We Also Request for Frank Assessment
Regrading Various Aspect to
the Book as Given Below

Database Management System

1. Who recommended you this book ? (Please tick in box)

□ Teacher □ Friends □ Bookseller

2. Name of the recommending teacher, designation and address.

....

....

3. What are the chapters where in the contents are not systematic or organised updated?

....

....

....

....

4. What contents of your syllabus, important from the point or view of examination, we have not covered in the book ?

....

....

....

5. Have you come across misprints/mistakes in the book ? if any, please specify the chapters and the page numbers.

....

....

....

6. Give your Suggestion and comments for the improvement of this book.

....

....

....

(Note : If need be please attach a separate sheet to write your views)

The best assessment will also be awarded each month. The award will be in the form of our publications as decided by our managerial Board.

You will be qualified for Lucky Draw Coupon Scheme, Whenever you will fill each detail given in both the pages (I & II). Please Post the filled up (Write Lucky Draw Coupon Schemes in bold Letters your name & Address on envelop) at your earliest to :

Shivani Publications
Ghuraiya Complex, Baradari Chowk,
Morar, Gwalior-474006 (M.P.)

卐 卐

वर
दे
वीणा
वादिनी

卐 卐

About Preparation of This Book

The Present Edition of this book DATABASE MANAGEMENT SYSTEM Stands out distinctly from others on account of the following salient features.

- This book Covers Complete New Syllabus as Prescribed by R.G.P.V. Bhopal.
- This book is according to New Scheme & Syllabus of Examinations.
- This book is thoroughly revised with a view to making it more student friendly.
- This book covers each and every topic in lucid and simple language with questions and answer form with easy solutions.
- This book has been presented on Teach Yourself technique without assuming any prior knowledge of the subject.
- This book has been presented with essentially elementary approach along with some Special tips on important topics and Diagrams.
- This book includes Complete Topics organised in to increasing degree of complexity, Nos of Numerical Problems with easy solution.
- All Questions set at examinations of R.G.P.V. Bhopal are included Chapter-wise with Full Solutions/Answers.
- We have referred to a number of books on DATABASE MANAGEMENT SYSTEM before writing of this book. Still we would whole heartely accept suggestions for improvement offered by the reader. We hope this book will meet all the requirements of the readers and come up to their expectations.

Syllabus

DATABASE MANAGEMENT SYSTEM (B.E., V-SEM)

UNIT : I : DBMS Concepts and architecture Introduction, Database approach v/s Traditional file accessing approach, Advantages of database systems, Data models, Schemas and instances, Data independence, Database Language and interfaces, Overall Database Structure, Functions of DBA and designer, ER data model : Entities and attributes, Entity types, Defining the E-R diagram, Concept of Generalization, Aggregation and Specialization. transforming ER diagram into the tables. Various other data models : object oriented data Model, Network data model, and Relational data model, Comparison between the three types of models.

UNIT : II : Relational Data models: Domains, Tuples, Attributes, Relations, Characteristics of relations, Keys, Key attributes of relation, Relational database, Schemas, Integrity constraints. Referential integrity, Intension and Extension, Relational Query Languages : SQL-DDL, DML, integrity constraints, Complex queries, various joins, indexing, triggers, assertions, Relational algebra and relational calculus, Relational algebra operations like select, Project, Join, Division, outer union. Types of relational calculus i.e., Tuple oriented and domain oriented relational calculus and its operations.

UNIT : III : Database Design : Introduction to normalization, Normal forms, Functional dependency, Decomposition, Dependency preservation and lossless join, problems with null valued and dangling tuples, multivalued dependencies. Query Optimization : Introduction, steps of optimization, various algorithms to implement select, project and join operations of relational algebra, optimization methods : heuristic based, cost estimation based.

UNIT : IV : Transaction Processing Concepts : – Transaction System, Testing of Serializability, Serializability of schedules, conflict & view serializable schedule, recoverability, Recovery from transaction failures. Log based recovery. Checkpoints deadlock handling. Concurrency Control Techniques – Concurrency Control, locking Techniques for concurrency control, time stamping protocols for concurrency control, validation based protocol, multiple granularity. Multiversion schemes, Recovery with concurrent transaction. Introduction to Distributed databases, data mining, data warehousing, Object Technology and DBMS, Comparative study of OODBMS Vs DBMS. Temporal, Deductive, Multimedia, Web & Mobile database.

UNIT : V : Study of Relational Database Management Systems through Oracle/PL SQL, MySQL : Architecture, physical files, memory structures, background process. Concept of table spaces, segments, extents and block. Dedicated server, multi threaded server. Distributed database, database links, and snapshot. Data dictionary, dynamic performance view. Security, role management, privilege management, profiles, invoker defined security model. SQL queries, Data extraction from single, multiple tables equi-join, non equi-join, self-join, outer join. Usage of like, any, all, exists, in Special operators. Hierarchical queries, in line queries, flashback queries. Introduction of ANSI SQL, anonymous block, nested anonymous block, branching and looping constructs in ANSI SQL. Cursor management: nested and parameterized cursors, Oracle exception handling mechanism. Stored procedures, in, out, in out type parameters, usage of parameters in procedures. User defined functions their limitations. Triggers, mutating errors, instead of triggers.

Price : Rs. 115.00 (Rs. One Hundred Fifteen Only)

Edition : 2019

Contents

DATABASE MANAGEMENT SYSTEM (B.E., V-SEM)

UNIT 1

DBMS CONCEPTS AND ARCHITECTURE INTRODUCTION, DATABASE APPROACH V/S TRADITIONAL FILE ACCESSING APPROACH, ADVANTAGES OF DATABASE SYSTEMS

Q.1. What is meant by DBMS ?

Ans. A database management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the ***database,*** contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.

Q.2. Differentiate the term data, information and knowledge.

(R.G.P.V., Dec. 2014, 2015)

Ans. Data is a collection of raw facts and figures. Whereas, information is the result of data processing which can be used to help people in decision making. But knowledge is a collection of interesting and useful information.

In data, there is no role of information whereas in information processing data is the most important element. In case of knowledge, information is the main element.

Q.3. Define the term data redundancy and inconsistency.

(R.G.P.V., Dec. 2011)

Or

What is data inconsistency ? ***(R.G.P.V., Dec. 2016)***

Ans. Since different programmers create the files and application programs over a long period, the various files are likely to have different formats and the programs may be written in several programming languages. Furthermore, the same information may be duplicated in several files. This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency, that is, the various copies of the same data may no longer agree.

Q.4. What are the problem caused by data redundancies ? Can data redundancy be completely eliminated when a database approach is used. ***(R.G.P.V., Dec. 2014)***

Ans. Redundancy of data means, having same set of data many times at different locations. For example, in case of college administration system. The identity of students are stored in every department like admission department, fees department and examination conducting unit etc.

This redundancy of data leads to various problems. Firstly, different units are storing the same data thus storage spaced is wasted. Secondly, in case any manipulation has to be done, it has to be done at every place, thus leading to duplication of efforts. Third and major one, it leads to inconsistency, i.e. it may be possible that two records in the two departments reveal different facts, and we cannot judge which one is correct. Those anomalies can destroy the effectiveness of the database.

The designed goal with the database approach forces to concentrate the data into a single logical structure, that will be used by every department. And the data in design, is related with another with the help of primary fact.

For example, in student database, the primary fact is roll number, the various data like name, address, and phone along with roll number is stored by admission unit, the other unit uses the roll no., for distinguishing every student from another.

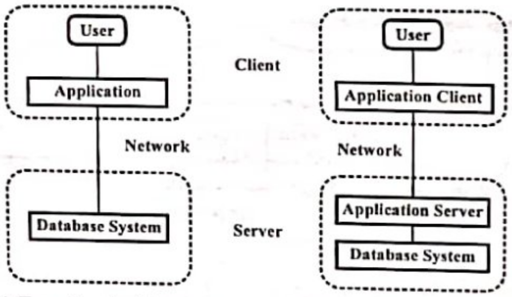
The relational database controls data redundancy by using common attributes that are shared by tables, called foreign keys. The proper use of foreign keys is crucial to controlling data redundancy. Although the use of foreign keys does not totally eliminate data redundancies, because the foreign key values can be repeated many times, the proper use of foreign keys minimizes data redundancies, thus minimizing the chance that destructive data anomalies will develop.

Q.5. Explain data dictionary. ***(R.G.P.V., Dec. 2013)***

Ans. The DBMS must provide a data dictionary function. The data dictionary is a database which contains "data about the data" (called metadata or descriptor) – that is, definitions of other objects in the system, instead of "raw data". In particular, all of the various schemas and mappings and all of the various security and integrity constraints will be stored, in both source and object form, in the dictionary.

Q.6. Differentiate between two tier and three tier client/server architecture. ***(R.G.P.V., Nov./Dec. 2007, Dec. 2013)***

Ans. Database applications are partitioned into two or three parts as shown in fig. 1.1. In a ***two-tier architecture***, the application is partitioned into a component that resides at the client machine, which invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.



(a) Two-tier Architecture ***(b) Three-tier Architecture***

Fig. 1.1 Two-tier and Three-tier Architectures

In contrast, in a three-tier architecture, the client machine acts as merely a front end and does not contain any direct database calls. Instead, the client end communicates with an application server, through a forms interface. The application server in turn communicates with a database system to access data. The ***business logic*** of the application, which says what actions to carry out under what conditions, is embedded in the application server, instead of being distributed across multiple clients. Three-tier applications are more appropriate for large applications, and for applications that run on the World Wide Web.

Q.7. Briefly explain about database system architecture.

Or ***(R.G.P.V., Dec. 2017)***

Explain the component modules of a DBMS and their interactions with the architecture. ***(R.G.P.V., June 2010)***

Or

Explain system structure of DBMS.

(R.G.P.V., June 2006, 2007, Dec. 2013)

Or

What do you mean by DBMS architecture ?

(R.G.P.V., June 2004, Nov./Dec. 2007, Dec. 2016)

Or

Discuss in detail about database system architecture with neat diagram. ***(R.G.P.V., May 2018)***

Ans. A database system is partitioned into module that deal with each of the responsibilities of the overall system. The functional components of a database system can be divided into the storage manager and the query processor components. Fig. 1.2 shows the structure of DBMS.

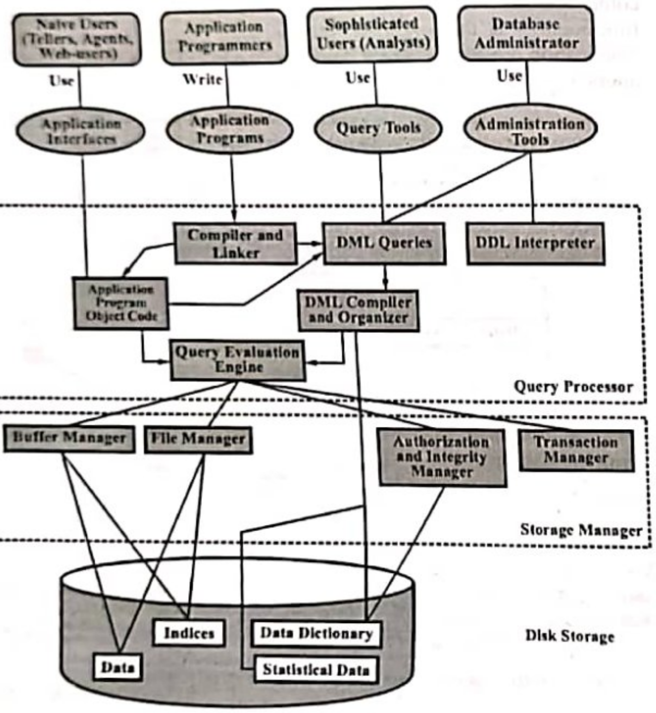


Fig. 1.2 System Structure

In fig. 1.2, the functional components of DBMS have query components and storage components as follows –

The Query Processor – The query processor components include –

(i) DDL Interpreter – Its function is to execute the low-level statements and records them in a set of tables that having metadata.

(ii) DML Compiler – Its function is to convert DML statements in source form of query language into necessary object form (i.e., low-level instructions) that query evaluation engine understands.

(iii) DML Precompiler – Its function is to convert DML statements embedded in application program to normal procedure calls in host language. To generate appropriate code, the precompiler must interact with DML compiler.

(iv) Query Evaluation Engine – Its function is to execute the low-level instruction generated by DML compiler.

Storage Manager – A storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.

The storage manager components include –

(i) Authorization and Integrity Manager – Tests for the satisfaction at integrity constraints and checks the authority of users to access data.

(ii) Transaction Manager – Ensures that the database remains in consistent state despite system failure and concurrent transaction executions proceed without conflicting.

(iii) Buffer Manager – Responsible for data fetching from disk storage into main memory and deciding what data to cache in memory.

(iv) File Manager – Manages allocation of space on disk storage and data structures used to represent information that stored on disk.

The storage manager implements several other data structures are needed as the part of physical system for implementation –

(i) Indices – These provide fast access to the data items that hold particular values.

(ii) Statistical Data – This store statistical information about data in database. To execute a query, this information is used by query processor.

(iii) Data Files – These are actual files that store the data in database i.e., these are database files

(iv) Data Dictionary – This stores metadata about each and every entity of the database along with security and integrity constraints.

Q.8. Define database management system (DBMS). What are the major component of this system ? Explain each component.

(R.G.P.V., Nov. 2018)

Ans. **DBMS** – Refer to Q.1.

Component of DBMS – Refer to Q.7.

Q.9. What are the different modules present ? Explain in detail.

(R.G.P.V., Nov. 2018)

Ans. Refer to Q.7.

Q.10. Explain the difference between a file-oriented system and a database oriented system. ***(R.G.P.V., June 2009)***

Or

How is the data-based approach different from the file based approach? ***(R.G.P.V., Dec. 2014)***

Ans. There are a number of characteristics to distinguish the database approach from the traditional approach of programming with files. In traditional file processing, each user defines and implements the files needed for a specific application as part of programming the application. For example, one user, the ***grade reporting office***, may keep a file on students and their grades. Programs to print a student transcript and to enter new grades into the file are implemented. A second user, the accounting office, may keep track of student's fees and their payments. Although both users are interested in data about students, each user has to maintain separate files and programs to manipulate these files because each user requires some data not available from the other user's files. This redundancy in defining and storing data causes wasted storage space and redundant efforts to maintain common data up-to-date. In the database approach, a single repository of data is maintained that is defined once and then is accessed by various users.

The main characteristics of the database approach versus the file-processing approach are as follows –

(i) Self-describing Nature of a Database System – The database system contains not only the database itself but also a complete definition or description of the database structure and constraints. This definition is stored in the system, ***catalog***, which contains information such as the structure of each file, the type and storage format of each data item and various constraints on the data. The information stored in the catalog is ***meta-data*** and it describes the structure of the primary database.

In traditional file processing, data definition is the part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs. Whereas file-processing software can access only specific databases, DBMS software can access diverse databases by extracting the data definitions from the catalog and then using these definitions.

(ii) Insulation between Programs and Data, and Data Abstraction – In traditional file processing, the structure of data files is embedded in the access programs, so any changes to the structure of a file may require changing all programs that access this file. By contrast, DBMS access programs do not require such changes. The structure of data files is stored in the DBMS catalog separately from the access programs. This property is called ***program-data independence***.

The characteristic that allows program-data independence and program-operation independence is called ***data abstraction***.

(iii) Multiple Views of the Data – A database has many users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored. Some users do not need to be aware of whether the data they refer to is stored or derived. A multiuser DBMS having variety of applications must provide facilities for defining multiple views.

(iv) Sharing of Data and Multiuser Transaction Processing – A multiuser DBMS must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be integrated and maintained in a single database. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

Q.11. List four significant differences between a file processing system and a DBMS. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.10.

Q.12. What are the main differences between a file processing system and a database management system ? Describe the overall system architecture of a database system. Also show the connection system. Also show the connection amongst its various components. ***(R.G.P.V., Dec. 2012)***

Ans. **Differences between a File Processing System and a Database Management System** – Refer to Q.10.

System Architecture of a Database System – Refer to Q.7.

Q.13. What are the advantages of DBMS over traditional file system ? ***(R.G.P.V., June 2007)***

Ans. The advantages of DBMS are as follows –

(i) Reduction of Redundancies – Centralized control of data by the DBA avoids unnecessary duplication of data and reduces the total amount of data storage required. It also eliminates the extra processing necessary to trace the required data in a large mass of data. Another advantage of avoiding duplication is the elimination of the inconsistencies present in redundant data files.

(ii) Shared Data – A database allows the sharing of data under its control by any number of application programs or users.

(iii) Integrity – Centralized control ensures that adequate checks are incorporated in the DBMS to provide data integrity. Data integrity means that the data contained in the database is accurate and consistent.

(iv) Security – Data is of vital importance to an organization and may be confidential. This confidential data must not be accessed by unauthorised persons. The DBA ensures that proper access procedures are followed, including proper authentication schemes for access to the DBMS and additional checks before permitting access to sensitive data. Different levels of security can be implemented for various types of data and operations. The enforcement of security can be datavalue dependent as well as data-type dependent.

(v) Conflict Resolution – The DBA resolves the conflicting requirements of various users and applications. In essence, the DBA chooses the best file structure and access method to get optimal performance for the response-critical applications, while permitting less critical applications to continue to use the database, although with a relatively slower response.

(vi) Data Independence – There are two types of data independence – physical data independence and logical data independence. Data independence is advantageous in the database environment since it allows for changes at one level of the database without affecting other levels.

Q.14. Compare database approach with traditional file accessing approach. Also, discuss the advantages of database systems. (R.G.P.V., Dec. 2009)

Ans. **Comparison** – Refer to Q.10.

Advantages of Database System – Refer to Q.13.

Q.15. What is DBMS and what are the components of DBMS ? What are the advantages of DBMS over file oriented approach ? (R.G.P.V., Dec. 2010)

Ans. **DBMS** – Refer to Q.1.

DBMS Components – Refer to Q.7.

Advantages of DBMS – Refer to Q.13.

DATA MODELS, SCHEMAS AND INSTANCES, DATA INDEPENDENCE, DATABASE LANGUAGE AND INTERFACES, OVERALL DATABASE STRUCTURE, FUNCTIONS OF DBA AND DESIGNER

Q.16. Define the term data models. (R.G.P.V., June 2010, Dec. 2011)

Ans. A model is an abstraction process that hides superfluous details while highlighting details pertinent to the applications at hand. Data model is a mechanism that provides the abstraction for database applications. Data modeling is used to represent entities of interest and their relationships in the database. It allows the conceptualization of the association between various entities and their attributes.

We can say that data model is a collection of conceptual tools to describe data, data relationships, data semantics and consistency constraints.

A number of models for data representation have been developed. Most data representation models, provide mechanisms to structure data for the entities being modeled and allow a set of operations to be defined on them. The models also enforce a set of constraints to maintain the integrity of the data. These models differ in their method of representing the associations amongst entities and attributes.

Q.17. Discuss main categories of data models. (R.G.P.V., Dec. 2006, Nov./Dec. 2007)

Or

Explain the various data models briefly with an example. (R.G.P.V., June 2016)

Ans. One data model can be distinguished from other on the basis of the way relationship among data, that is defined and the way the data is conceptually defined. There are many data models, chosen as per need of the application. These are fallen in following categories –

(i) Object-based logical models

(ii) Record-based logical models

(iii) Physical data models.

(i) Object-based Logical Models – These models are used in describing data at logical and view levels. They are characterized by the fact that they provide flexible structuring capabilities and allow data constraints to specify explicitly. This model emphasizes that, everything is object having a set of attributes. There are many data models in this category –

(a) Entity-relationship model (b) Object-oriented model

(c) Semantic data model (d) Functional data model.

(a) Entity-relationship Model – This model moves around three things – entity, attribute and relationship. This model is based on perception that consists of collection of objects called entities and every two entities are distinguished from other through their own set of properties. The relationship exists between these entities. E-R diagram graphically expresses the logical structure of database (schema) and it uses –

(1) Rectangles, to represent entity set

(2) Ellipses, to represent attributes

(3) Diamonds, to represent relationships among entity sets

(4) Lines, to show the links between entities and relationships.

(b) Object-oriented Model – This model is based on a collection of objects. Object has values stored in instance variable within the object. An object also contains bodies of code that operate on the object. These bodies of code are called ***methods***. This model introduces the concept of classes that contains objects having same type of values i.e., same set of attribute names and same methods. Values of attributes determine the object along with the methods i.e., set of instructions, that are used to modify those values. At any particular moment of time, object can be said to be the instance of the class. Two or more objects communicate each other by passing messages. The only way in which one object can access the data of another object is by invoking a method of that object. This action is called ***sending a message*** to the object. Thus, the call interface of the methods of an object defines that objects are externally visible parts. The internal parts of the object, the instance variables and methods code are not visible externally. The result is two levels of data abstraction.

(ii) Record-based Logical Models – These models are also used in describing data at logical and view levels. But in contrast to the object-based data models, they revolve around the records of the database and specify the overall structure of database, with the help of values of records. Record-based models are so named, since the database is structured in fixed-format records of several types.

Each record type defines a fixed number of fields or attributes and each field is usually of a fixed length. This simplifies the physical-level implementation of the database. The three most widely used record-based models are –

(a) Relational model (b) Network model (c) Hierarchical model.

(a) Relational Model – This is most popular among the various record-based models. This model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns and each column has unique name. Table is given the name relations, rows represent the records and the columns represent the attributes or properties.

In this model, data is handled on a conceptual rather than physical basis. This mechanism helps in processing entire files of data with single statement. The logical manipulation of data also makes feasible the creation of query languages more accessible to non-technical users.

customer_name	social_security	customer_street	customer_city	account_number
Johnson	192-83-7465	Alma	Palo Alto	A-101
Smith	019-28-3746	North	Rye	A-215
Hayes	677-89-9011	Main	Harrison	A-102
Turner	182-73-6091	Putnam	Stamford	A-305
Johnson	192-83-7465	Alma	Palo Alto	A-201
Jones	321-12-3123	Main	Harrison	A-217
Lindsay	336-66-9999	Park	Pittsfield	A-222
Smith	019-28-3746	North	Rye	A-201

For example, fig. 1.3 represents a sample of relational database comprising of two tables. One table shows bank customers and other shows the account that belong to those customers. It shows, for example, that customer Johnson with *social_security* number 192-83-7465, lives on Alma in Palo Alto and has two accounts, A-101 with a *balance* of $500 and A-201 with a *balance* of $900. Also, customers Johnson and Smith share account number A-201.

account_number	balance
A-101	500
A-215	700
A-102	400
A-305	350
A-201	900
A-217	750
A-222	700

Fig. 1.3 Relational Database Sample

(b) Network Model – Data in network model are represented by collection of records and relationships among data are represented by links, that can be viewed as pointers. A pointer is a physical address which identifies where next record can be found on the disk. Fig. 1.4 shows a network database using the same information as shown in fig. 1.3.

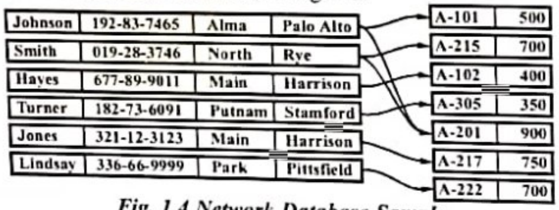


Fig. 1.4 Network Database Sample

(c) Hierarchical Model – It is very similar to network model, as in both of the models, data and relationships among data are represented by records and links respectively. In this model, records are organized as collection of trees rather than arbitrary graphs. Fig. 1.5 shows hierarchical database system.

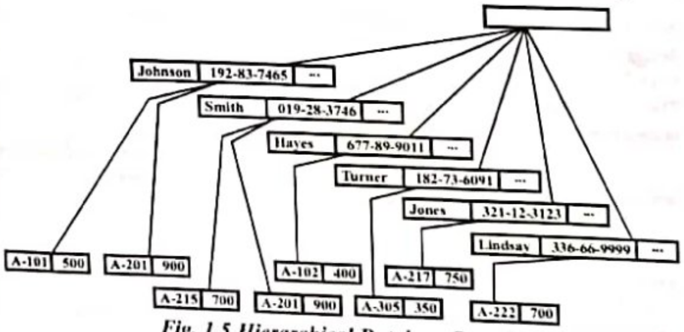


Fig. 1.5 Hierarchical Database Sample

(iii) Physical Data Models – This model is used to describe data at the lowest level i.e., to describe the behaviour of data at the disk level i.e., the way the data and data relationships are maintained while storing them on the disk. This decided the way the DBMS is going to use secondary storage devices, for storing and accessing database. The widely used these data models are –

(a) Unifying model (b) Frame memory model.

Q.18. Explain network and hierarchical model and compare them. ***(R.G.P.V., Dec. 2009)***

Ans. Refer to Q.17 (ii) (b) and (c).

Q.19. Explain the term database schema. ***(R.G.P.V., Nov./Dec. 2007, Dec. 2013)***

Or

What is schema and subschema ? Explain these two concepts through examples. ***(R.G.P.V., Dec. 2014)***

Or

What is schemas ? ***(R.G.P.V., June 2009, 2016)***

Ans. Schema is the logical description of entire database. That is, the overall design of the database is called ***database schema*** and it is changed rarely. The arrangement of all the files for an entire organization is an example of the schema.

Database systems have several schemas, partitioned according to the levels of abstraction. At lowest level is physical schema, at intermediate level is logical schema and at highest level is subschema or external schema. Generally, database systems support one physical schema, one logical schema and several subschemas.

Basically, internal schema describes how the data are actually stored on disk. Conceptual schema describes the structure of the database to database designer. External schema or subschema describes the structure of database to end users. An end user gets the view of the database via external schema. The arrangement of files pertaining to a specific subsystem of an organization is an example of the subschema.

Q.20. Describe the three-level architecture of DBMS. Also explain its importance in a database environment. ***(R.G.P.V., Dec. 2017)***

Ans. Fig. 1.6 shows the three-schema architecture. Its goal is to separate the user applications and the physical database. The architecture is divided into three levels – the external level, the conceptual level and the internal level. The view at each of these levels is described by a schema. A schema is an outline or a plan that describes the records and relationships existing in the view.

(i) Internal Level – The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

(ii) Conceptual Level – The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations and constraints. A high-level data model or an implementation data model can be used at this level.

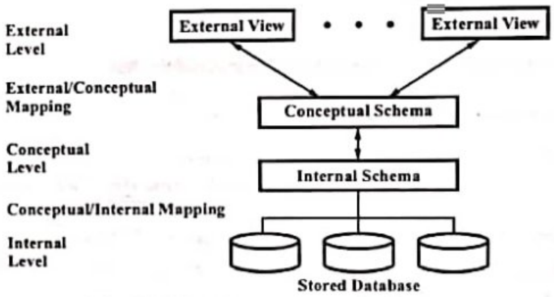


Fig. 1.6 The Three-schema Architecture

(iii) External or View Level – The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of database from that user group. A high-level data model or an implementation data model can be used at this level.

Q.21. Discuss the three-level architecture of DBMS. Explain how does it lead to data independence. ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.20.

Whenever we have a multiple-level DBMS, its catalog must be expanded to include information on how to map requests and data among the various levels. The DBMS uses additional software to accomplish these mappings by referring to the mapping information in the catalog. Data independence is accomplished because, when the schema is changed at some level, the schema at the next higher level remains unchanged, only the mapping between the two levels is changed. Hence, application programs referring to the higher-level schema need not to be changed.

Q.22. Discuss overall system structure of a database management system. Explain the difference between logical and physical schema. What is view level in this context ? ***(R.G.P.V., June 2011)***

Ans. Refer to Q.7 and Q.20.

Q.23. Define the term database instance.

Ans. Generally, the database is changed over time, as data is inserted, deleted or updated. The state of database at any particular moment of time i.e. the collection of data and information along with various objects definition is called the ***instance*** of the database.

In database terminology, the structure is a schema and the instance of schema is a database.

Q.24. What is the difference between a database schema and a database state ? ***(R.G.P.V., Dec. 2006, 2010)***

Ans. Refer to Q.19 and Q.23.

Q.25. Explain the term data independence.
(R.G.P.V., Nov./Dec. 2007, Dec. 2013)
Or
What is data independency ? ***(R.G.P.V., June 2007, Dec. 2016)***

Ans. The term "data independence" can be explained easily with the three-schema architecture as follows –

Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

Q.26. Write differences between the logical and physical data independence. ***(R.G.P.V., Dec. 2006)***

Ans. Logical data independence is the capacity to change the conceptual schema without having to change external schema or application programs.

In turn physical data independence is the capacity to change the internal schema without having to change the conceptual (or external) schemas.

Q.27. Write short note on data definition language (DDL).
(R.G.P.V., June 2010)

Ans. A database schema is specified by a set of definitions expressed by a special language called ***data definition language*** (**DDL**). After compilation of DDL statements, we get a set of tables that is stored in a special file called ***data dictionary*** or ***data directory***.

DDL allows us to create databases, tables and indexes. Furthermore, it allows changes to these objects and also their deletion. Some DDL commands are Create table, Drop table and Alter table. For example, to create our Book table, we can use the Create table command provided by SQL as –

```
CREATE TABLE BOOK
(Id              INTEGER,
Subject          CHAR (20)
Title            CHAR (30)
Author           CHAR (20)
Publication      CHAR (30))
```

The column, the form, the table are specified within the brackets. Along with every column we specify the type and size of data allowed for that column. The DBMS uses its own pre-specified length for integers. After executing the create table statement, only an empty structure called the template of the table is created.

Q.28. Write short note on data manipulation language (DML).
(R.G.P.V., June 2010)

Ans. A data manipulation language (DML) is a language that enables users to access or manipulate data as organized by appropriate data model. We have basically two types of languages.

(i) Procedural DMLs – This specifies, **what** data are needed and **how** to get those data.

(ii) Non-procedural DMLs – This specifies, **what** data are needed without specifying how to get those data.

Some DML commands are Select, Insert, Update and Delete. The Select statement allows viewing of the contents of a table in a variety of ways. All or only a few columns, rows or their combinations can be selected, based on the search criteria. SELECT * gives all the rows and all the colums of a table. The WHERE clause does the job of restriction. Only the rows satisfying one or more conditions can be retrieved by using it. For example,

```
SELECT Name, Author
FROM Book
WHERE Author = "Miller"
```

The above command gives the Name and Author from the Book table of Author Miller.

Q.29. Write short note on data control language (DCL).
(R.G.P.V., June 2010)

Ans. We must know how to give access to and revoke accesses from various tables to different users. This is very important in case of a multi-user database system. This can be achieved with the use of Data Control Language (DCL).

Suppose there are two persons working in the personnel department – Hari and Mohan. Only they should have access to the *Employee* table. We can secure access by using the GRANT and REVOKE statements provided by SQL as follows. First, we remove all accesses to the table by everybody.

```
REVOKE ALL ON Employee
FROM PUBLIC
```

Here, ALL means the ability to perform any DML operation such as SELECT, INSERT, UPDATE, or DELETE. Thus, we do not allow anyone (because we specify PUBLIC) to perform any operation on the *Employee* table. In other words, PUBLIC means *everybody*.

Now, we shall allow only Hari and Mohan the necessary authority to access then *Employee* table.

```
GRANT ALL
ON Employee
TO Hari, Mohan
```

This would allow Hari and Mohan to perform any DML operation on the *Employee* table. Suppose at some stage, Mohan is on leave and a third employee, David, temporarily takes his place. We need to provide only SELECT access to David on the *Employee* table. This can be done as follows –

```
GRANT SELECT
ON Employee
TO David
```

David cannot update the information in the *Employee* table in any manner. He can only see what it contains.

Q.30. What are DDL, DML and DCL ? Differentiate among the three and give one command for each of these. ***(R.G.P.V., Dec. 2015)***

Ans. Refer to Q.27, Q.28 and Q.29.

Q.31. Discuss the different types of user-friendly interfaces.

Ans. User-friendly interfaces provided by a DBMS may include the following –

(i) Menu-based Interfaces for Browsing – These interfaces present the user with lists of options, called menus, that lead the user through the formulation of a request. Menus eliminate the need to memorize the specific commands and syntax of a query language. Rather, the query is composed step-by-step by picking actions from a menu that is displayed by the system. Pull-down menus are used in window-based user interfaces. They are often used in browsing interfaces, which allow a user to look through the contents of a database is an exploratory and unstructured manner.

(ii) Forms-based Interfaces – A forms-based interface displays a form to each user. Users can fill out all or only certain form entries to insert new data, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions.

(iii) Graphical User Interfaces – A graphical user interface displays a schema to the user in diagrammatic form. The user can then specify a query by manipulating the diagram. In several cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, such as a mouse, to pick certain parts of the displayed schema diagram.

(iv) Natural Language Interfaces – These interfaces accept requests written in English or some other language and attempt to understand them. A natural language interface has its own schema, which is similar to the database conceptual schema. The natural language interface refers to the words in its schema, as well as to a set of standard words, to interpret the request. If the interpretation is successful the interface generates a high-level query corresponding to the natural language request and submits it to the DBMS for processing; otherwise a dialogue is started with the user to clarify the request.

(v) Interfaces for Parametric Users – Parametric users, such as bank tellers, have a small set of operations that they must perform repeatedly. System analysts and programmers design and implement a special interface for a known class of naive users. Usually, a small set of abbreviated commands is included to minimize the number of keystrokes required for each request. For instance, function keys in a terminal can be programmed to initiate the various commands. This allows the parametric user to proceed with a minimal number of keystrokes.

(vi) Interfaces for the DBA – Many database systems contain privileged commands that can be used only by the DBA's staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

Q.32. What is DBA ? What are the various roles of DBA ? ***(R.G.P.V., Dec. 2016)***

Or

What are the responsibilities of database administrator ? Explain them in brief. ***(R.G.P.V., Dec. 2006, 2008)***

Or

Explain the role of database administrator. ***(R.G.P.V., June 2007, 2008)***

Or

Write short note on role of DBA. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. The ***data administrator*** (DA) is a person who makes the strategic and policy decisions regarding the data of the enterprise and the ***database administrator (DBA)*** is the person, who provides the necessary technical support for implementing those decisions. Thus, DBA is responsible for the overall control of the system at a technical level. Also, one of the main reasons for using DBMS is, to have central control of both the data and the programs that access those data. The person who has such central control over the system is called the database administrator (DBA).

Some of the functions of database administrator are as follows –

(i) Schema Definition – The DBA creates the original database schema by executing a set of data definition statements in DDL.

(a) Defining the Conceptual Schema – It is the data administrator's job to decide exactly what information is to be held in the database. In other words, to identify the entities of interest to the enterprise and to identify the information to be recorded about those entities. This process is usually referred to as logical (sometimes conceptual database design). Once the data administrator has decided the contents of the database at an abstract level, the DBA will then create the corresponding conceptual schema, using the conceptual DDL. The object form of that schema will be used by DBMS in responding to access requests.

(b) Defining the Internal Schema – The DBA must also decide, how the data is to be represented in the stored database. This process is usually referred to as physical database design. Having done the physical design, the DBA must then create the corresponding storage structure definition, using the internal DDL.

(ii) Storage Structure and Access Method Definition – The DBA creates the appropriate storage structures and access methods by writing a set of definitions, which is translated by the data-storage and data-definition language compiler.

(iii) Defining Integrity Constraints – Data values stored in the database, must satisfy certain consistency constraints. For example, perhaps the number of hours an employee may work in 1 week, may not exceed a specified limit (say, 80 hours). Such a constraint must be specified explicitly by the database administrator.

(iv) Security – In large organization, many users access the database. But not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about various bank employees. They do not need access to information about customer accounts. So, DBA provides every user a unique ***password*** for system security.

(v) Granting of Authorization for Data Access – As explained above, granting of different types of authorization allows the database administrator to regulate which parts of the database various users can access. ie authorization information is kept in a special system structure that is nsulted by the database system, whenever access to the data is attempted in e system.

(vi) Internal Marketing – It is the business of DBA to ensure every er, that the database contains every useful data they want, in the desired rmat (conceptual schema). Effective internal marketing may reduce resistance change and data ownership problems.

(vii) Managing Data Dictionary – Data dictionary contains metadata, at describes the data and data processing units. It is useful tool for database lministration and is used throughout the database system life cycle.

(viii) Monitoring DBMS Performance – It is DBA, who is sponsible for designing DBMS in such a manner that is best for the ganization. He/she may need to have regular maintenance activities that isures the appropriate performance at all the time.

(ix) Selection of Hardware and Software – DBA is the only one ho decides what are the hardware and software requirements of the system. election is done, keeping in mind the financial condition of the firm and thus the manner that ensures maximized performance with lowest cost. Selection iteria includes that, whether the selected hardware and software enables the rm to fight in the market with competitors or not.

(x) Data Backup and Recovery – DBA must ensure that DBMS is cilitated with various backup procedures and recovery schemas. This is must ensure it due to any error, database fails, the data could be easily recovered and e system will be reformed in consistent state, with no or minimal loss of data.

Q.33. Briefly explain the functions of DBA. ***(R.G.P.V., Nov. 2018)***

Ans. Refer to Q.32.

Q.34. What are the responsibilities of a DBA ? Describe the three levels f data abstraction. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.32 and Q.20.

Q.35. Define the term domain constraints. ***(R.G.P.V., June 2010, Dec. 2011)***

Ans. Domain constraints are the most elementary form of integrity onstraint. They are tested easily by the system whenever a new data item is ntered into the database.

It is possible that several attributes can have the same domain. For example, ie attributes *customer-name* and *employee-name* can have the same domain – ie set of all person names. However, the domains of *balance* and *branch-ame* certainly ought to be distinct. It is perhaps less clear whether *customer-ame* and *branch-name* should have the same domain.

The *create domain* clause can be used to define new domains. For exan the statement –

create domain Dollars *numeric* (12, 2)

define the domains Dollars to be decimal numbers with a total of 12 d two of which are placed after the decimal point. Values of one domain ca cast i.e., converted to other domain. SQL also provides **drop domain** **alter domain** clauses to drop or modify domains that have been created ea

The **check** clause in SQL-92 permits domains to be restricted in pow ways that most programming language type systems do not permit. example, a **check** clause can ensure that an hourly wage domain allows values greater than a specified value (such as the minimum wage), as sh below –

```
create domain hourly-wage numeric(5, 2)
      constraint wage-value-test check(value > = 4.00)
```

The domain has a constraint that ensures that the hourly-wage is gre than 4.00. The clause **constraint** *wage-value-test* is optional, and is use give the name *wage-value-test* to the constraint. The name is used to indic which constraint an update violated.

The **check** clause can also be used to restrict a domain to not contain null values, as shown below –

```
create domain account-number char(10)
      constraint account-number-null-test check(value not nu
```

As another example, the domain can be restricted to contain only a specif set of values by using the **in** clause –

```
create domain account-type char(10)
      constraint account-type-test
            check(value in('Checking', 'Saving'))
```

E-R DATA MODEL – ENTITIES AND ATTRIBUTES, ENTITY TYPES, DEFINING THE E-R DIAGRAM, CONCEPT OF GENERALIZATION, AGGREGATION AND SPECIALIZATION, TRANSFORMING E-R DIAGRAM INTO THE TABLES, VARIOU OTHER DATA MODELS – OBJECT ORIENTED DATA MODEL, NETWORK DATA MODEL, AND RELATIONAL DATA MODEL, COMPARISON BETWEEN THE THREE TYPES OF MODELS

Q.36. Define the E-R model.

Ans. The Entity-Relationship (E-R) Model is a graphical representation the database logic and includes a detailed description of all entities, relationshi and constraints.

The E-R data model is based on a perception of a real world that consists a set of basic objects called ***entities***, and of ***relationships*** among these jects. It was developed to facilitate database design by allowing the ecification of an ***enterprise schema***, which represents the overall logical ucture of a database.

Q.37. What are entities and attributes ? Discuss them.

Ans. An entity is a thing in the real world with an independent existence. n entity may be an object with a physical existence – a particular person, car, ouse, or employee – or it may be an object with a conceptual existence – a mpany, a job, or a university course.

Each entity has attributes i.e., the particular properties that describe it. For ample, an employee entity may be described by the employee's name, age, ldress, salary and job. Each attribute of an entity set has a particular value. The t of possible values that a given attribute can have is called its ***domain***.

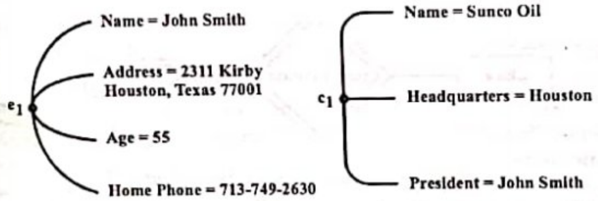


Fig. 1.7 Two Entities, an Employee e_1 and a Company c_1, and their Attribute Values

Fig. 1.7 shows two entities and the values of their attributes. The employee ntity e_1 has four attributes – Name, Address, Age and Homephone; their alues are "John Smith", "2311 Kirby, Houston, Texas 77001", "55", and 713-749-2630", respectively. The company entity c_1 has three attributes – ame, Headquarters, and President; their values are "Sunco Oil", "Houston", nd "John Smith", respectively.

Q.38. How strong and weak entity sets are represented in E-R diagram ? xplain with an example.

Or

Make comparison between the strong entity and weak entity.
(R.G.P.V., Nov./Dec. 2007, June 2009)

Or

When is the concept of weak entity used in data modelling ? Define the rms owner entity, weak entity, identifying relationship, partial key.
(R.G.P.V. Dec. 2011)

Or

What is weak entity set and strong entity set ? (R.G.P.V., Dec. 2015)

Ans. Sometimes an entity set may not have sufficient attributes to fo primary key. Such an entity set is termed a ***weak entity set***. On the other h if any entity set has sufficient attributes to form a primary key, it is said t ***strong entity set*** For example, consider an entity set ***payment***, which has t attributes *payment_number*, *payment_date* and *payment_amount*. Although, ***payment*** entity is different, payments from different loans may share the s payment number. So, only on the basis of *payment_number*, we cannot disting between different entities. Thus, this entity set does not have a primary key, a weak entity set. Let us take an example of ***loan*** entity, with attributes *loan_num* and *amount*. Here *loan_number* behaves as a primary key for ***loan*** entity. loan is a strong entity set. This is given in fig. 1.8.

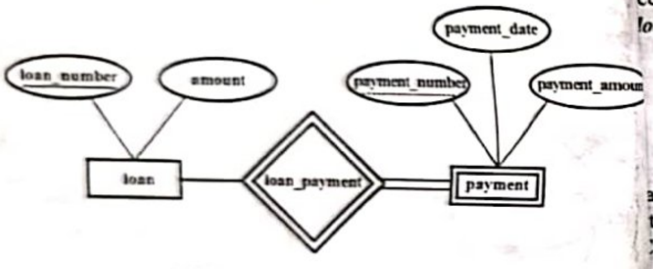


Fig. 1.8 E-R Diagram with a Weak Entity Set

The concept of weak and strong entity sets are very closely related to existence dependencies. A member of strong entity set is considered as domin entity, whereas a member of weak entity set is considered as subordinate ent

The primary key of a weak entity set is formed by the primary key of t strong entity set, on which the weak entity set is existence dependent, plus t weak entity set discriminator. The discriminator of a weak entity set is a set attributes that allow the distinction to be made among all those entities in t entity set that depends on one particular entity set. For example, the discrimina of the weak entity set ***payment*** is the attribute *payment_number*. Since *payment_number* uniquely identifies one single payment for that loan, T discriminator of a weak entity set is also called the ***partial key*** of the entity s

So, in the case of entity set ***payment***, its primary key is (*loan_numb* *payment_number*), where the *loan_number* identifies the dominant entity o ***payment*** and *payment_number* distinguishes ***payment*** entities within the sar ***loan***. The identifying dominant entity set is said to own the weak entity s that identifies. The relationship that associates the weak entity set with owner is the identifying relationship. In our example, fig. 1.8, ***loan_ payme*** is the identifying relationship for ***payment***.

In an E-R diagram, weak entity set is represented by a doubly outlined box and ıe corresponding identifying relationship by a doubly outlined diamond. In the fig. .8, weak entity set ***payment*** is shown in double outlined box and link between ***oan_ payment*** relationship set and ***payment*** entity set is also double, and ***loan_ ayment*** relationship set is also outlined doubly. Discriminator of weak entity set is ashed underlined as *payment_number* of entity set ***payment***, while the primary key f strong entity set is underlined with a solid line as *loan_number* in ***loan*** entity set.

Double lines in fig. 1.8 are used to identify the total participation. The articipation of weak entity set ***payment*** in the relationship ***loan_ payment*** is otal, meaning that every payment must be related via ***loan_ payment*** to some ccount. Arrow from relationship set (***loan_ payment***) to strong entity set ***loan***) indicates that, each payment is for a single loan.

Q.39. What do you mean by weak entity set ? ***(R.G.P.V., June 2016)***

Ans. Refer to Q.38.

Q.40. What do you mean by entity types and entity sets ?

Ans. An ***entity type*** defines a collection (or set) of entities that have the ame attributes. Each entity type in the database is described by its name and ttributes. Fig. 1.9 shows two entity types, named EMPLOYEE and OMPANY, and a list of attributes for each. The collection of all entities of a articular entity type in the database at any point in time is called an ***entity set***. he entity set is referred to using the same name as the entity type. For xample, EMPLOYEE refers to both a type of entity as well as the current set f all employee entities in the database.

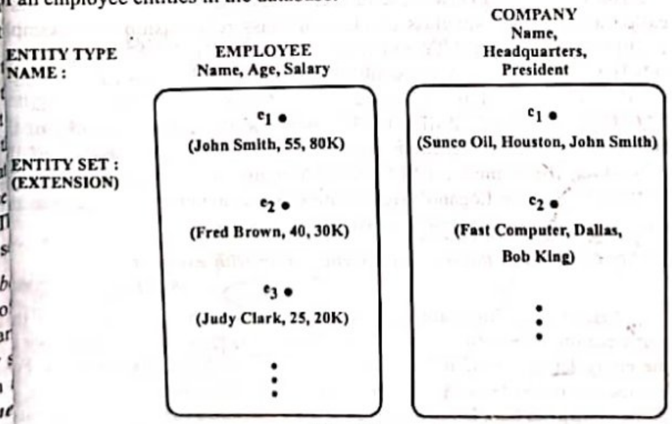


Fig. 1.9

Q.41. What is entity and attribute ? Explain the entity types.

(R.G.P.V., June 2

Or

Define entity and entity set. *(R.G.P.V., Dec. 2*

Ans. Refer to Q.37 and Q.40.

Q.42. Explain entity subtypes and supertypes.

Or

What is subclass ? When is subclass needed in data modelling ? Exp is-a relationship. *(R.G.P.V., Nov./Dec. 2*

Ans. An entity type is used to represent both a *type of entity*, and the e *set or collection of entities* of that exist in the database. For example, the e type EMPLOYEE describes the type of each employee entity and also re to the current set of EMPLOYEE entities in the COMPANY database. In m cases an entity type has numerous subgroupings of its entities that meaningful and need to be represented explicitly because of their significa to the database application. For example, the entities that are members of EMPLOYEE entity type may be grouped further into SECRETARY, ENGINE MANAGER, TECHNICIAN, SALARIED_EMPLOYEE, HOURLY_EMPLOY and so on. The set of entities in each of the latter groupings is a subse entities that belong to the EMPLOYEE. entity set, meaning that every en that is a member of one of these subgroupings is also an employee. Each these subgroupings is called a *subclass* of the EMPLOYEE entity type and EMPLOYEE entity type is called the *superclass* for each of these subclas

The relationship between a superclass and any one of its subclasse called a superclass/subclass or class/subclass relationship. For examp EMPLOYEE/SECRETARY and EMPLOYEE/TECHNICIAN are two cl subclass relationships. A class/subclass relationship is often called an IS (or IS-AN) relationship. We say "a SECRETARY IS-AN EMPLOYEE," TECHNICIAN IS-AN EMPLOYEE" and so forth. A member entity of subclass represents the same real-world entity as some member of superclass; for example, a SECRETARY entity 'Joan Logano' is also EMPLOYEE 'Joan Logano'. Hence, the subclass member is the same as entity in the superclass, but in a distinct specific role.

Q.43. Explain the term full participation with example.

(R.G.P.V., Dec. 20

Ans. Double lines are used in an E-R diagram to indicate that participation of an entity set in a relationship set is total; that is each entity the entity set occurs in at least one relationship in that relationship set. instance, consider the relationship ***borrower*** between customers and loans. double line from ***loan*** to ***borrower***, as in fig. 1.10 indicates that each loan m have at least one associated customer.

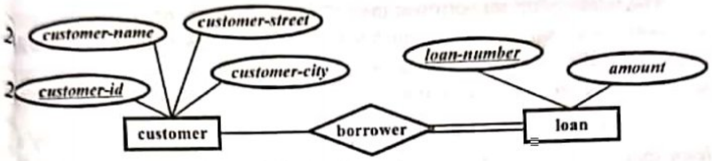


Fig. 1.10 Total Participation of an Entity Set in a Relationship Set

Q.44. Explain the representation of E-R diagram using a suitable xample.

Ans. The overall logical structure of a database can be expressed raphically by an E-R diagram. Because of its pictorial representation, it is asy to understand. Such a diagram consists of following main components –

(i) Rectangles represent entity sets.

(ii) Ellipses represent attributes.

(iii) Diamonds represent relationship sets.

(iv) Lines link attributes to entity sets and entity sets to relationship ets.

(v) Double ellipses represent multivalued attributes.

(vi) Dashed ellipses denote derived attributes.

(vii) Double lines indicate total participation of an entity in the elationship set.

(viii) Underlines denote the primary key attribute in an entity set.

(ix) Double rectangles represent weak entity sets.

Fig. 1.11 shows the entity-relationship (E-R) diagram consists of two ntity sets, customer and loan, related through a binary relationship set borrower. he attributes associated with customer are customer_id, customer-name, ustomer-street and customer-city. The attributes associated with loan are oan_number and amount. In fig. 1.11, attributes of an entity set that are nembers of the primary key are underlined.

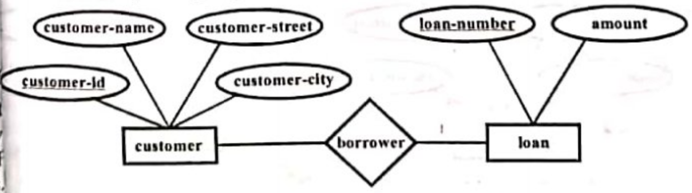


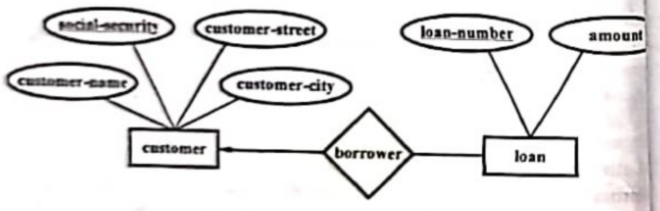
Fig. 1.11 E-R Diagram Corresponding to Customer and Loan

The relationship set borrower may be many-to-many, one-to-many, m
to-one and one-to-one. To distinguish between these types, either a directe
(→) or an undirected line (–) is drawn between the relationship set and the
set. Specification about directed and undirected lines are as follows –

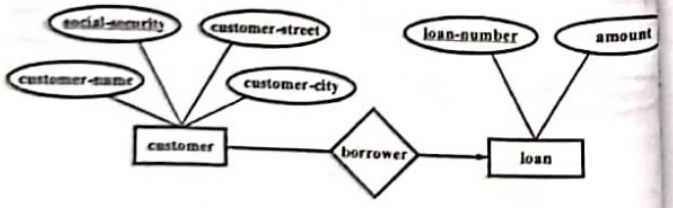
(i) A directed line from relationship set ***borrower*** to the enti
loan specifies that ***borrower*** is either one-to-one or many-to-one relatio
set, from ***customer*** to ***loan***; ***borrower*** cannot be many-to-many or on
many relationship set from ***customer*** to ***loan***.

(ii) Similarly, an undirected line from the relationship set ***born***
to the entity set ***loan*** specifies, that ***borrower*** is either a many-to-ma
one-to-many relationship set, from ***customer*** to ***loan***.

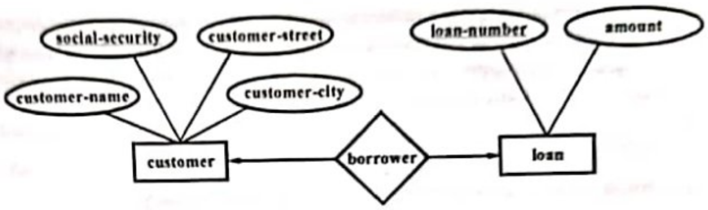
In fig. 1.11, ***borrower*** relationship set is many-to-many relationshi
the relationship set ***borrower*** were one-to-many from customer to loan,
the line from ***borrower*** to ***customer*** would be directed, with an arrow poi
to ***customer*** entity set as shown in fig. 1.12 (a). Similarly, if the ***borr***
relationship set were many-to-one from customer to loan, then line f
borrower to ***loan*** would be directed i.e., arrow pointing to ***loan*** entity se
shown in fig. 1.12 (b). Finally, if the ***borrower*** relationship set were to on
one, then both lines from ***borrower*** to ***customer*** and from ***borrower*** to l
would be directed i.e., arrow pointing to ***customer*** and ***loan*** entity sets
shown in fig. 1.12 (c).



(a) One-to-many Relationship



(b) Many-to-one Relationship



(c) One-to-one Relationship

Fig. 1.12 Relationships

If a relationship set has also some attributes associated with it, then we link these attributes to that relationship set. For example, in fig. 1.13, the descriptive attribute ***access-date*** has been attached to the relationship set depositor to specify the most recent data on which a customer accessed that account.

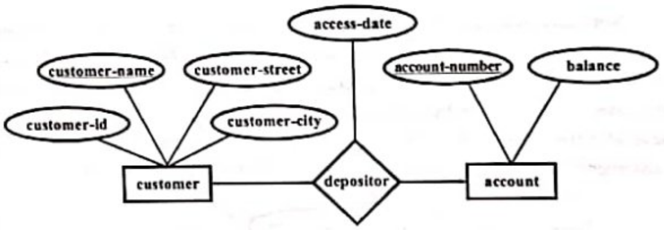


Fig. 1.13 E-R Diagram with an Attribute attached to a Relationship Set

Fig. 1.14 shows how composite attributes are represented in the E-R notation. Here, composite attribute name, with component attributes first-name, middle-name and last-name replaces the attribute customer-name of

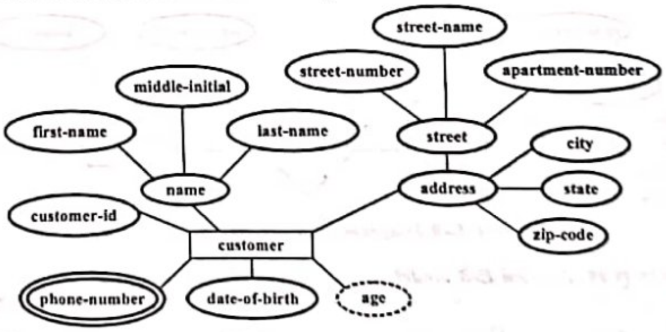


Fig. 1.14 E-R Diagram with Composite, Multivalued and Derived Attributes

customer. Also, a composite attribute address, whose component attributes are street, city, state and zip-code replaces the attributes customer-street and customer-city of customer. The attribute street is itself a composite attribute whose component attributes are street-number, street-name and apartment number. Fig. 1.14 also shows a multivalued attribute phone-number, depicted by double ellipses and desired attribute age, depicted by a dashed ellipse.

Roles of an E-R diagram can also be indicated by lines that connect diamonds to rectangles. Fig. 1.15 shows the role indicators, manager and worker, between the ***employee*** entity set and ***works-for*** relationship set.

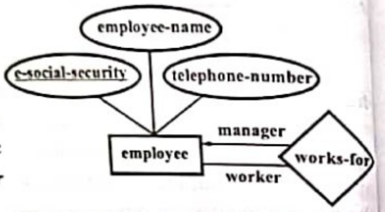


Fig. 1.15 E-R Diagram with Role Indicators

Normally, we have binary relationship between entity sets, but we also have examples in which three entities ***customer***, ***loan*** and ***branch***, related through the relationship CLB i.e., a ternary relationship set (see fig. 1.16). This diagram indicates that customer may have several loans and that loan may belong to several different customers. The arrow pointing to branch indicates that, each customer-loan pair is associated with a specific branch of bank.

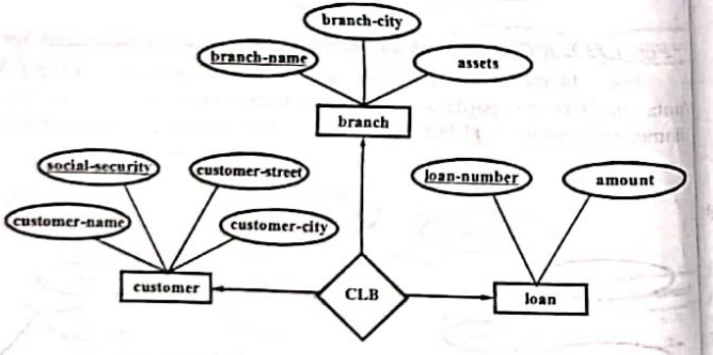


Fig. 1.16 E-R Diagram with Ternary Relationship

Q.45. Explain E-R model in detail with suitable example.
(R.G.P.V., Dec. 2017)

Ans. Refer to Q.36 and Q.44.

Q.46. A database is to be constructed to keep track of the teams and games of a sport league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game of each team and the result of the game.

Create an E-R diagram, completely with attributes, keys and constraints, for the above description. State any assumptions that you make.
(R.G.P.V., June 2009, Dec. 2010)

Ans. The E-R diagram for the universal hockey league (UHL) is given in fig. 1.17.

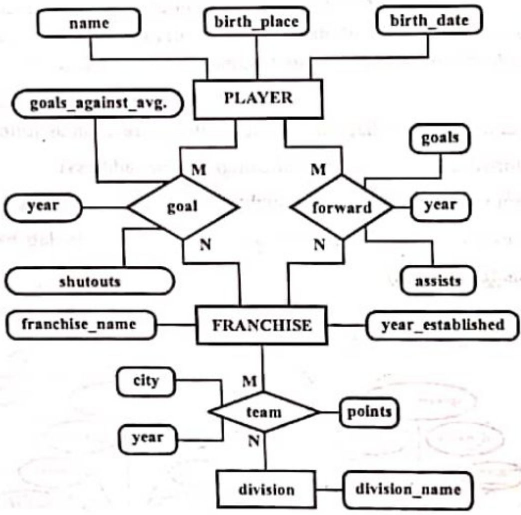


Fig. 1.17 E-R Diagram for the UHL Database

Q.47. Construct an E-R diagram for a hospital with a set of patients and a set of doctors. Associate with each patient a log of the various tests and examination conducted. Also show the tables for various entities with attributes.
(R.G.P.V., June 2011)

Or

Draw an E-R diagram for a hospital with a set of patients and a set of medical doctors. With each patient a log of the various conducted tests is also associated.
(R.G.P.V. Dec. 2011)

Or

Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted. ***(R.G.P.V., Dec. 2015)***

Or

Draw E-R diagram for hospital management system and convert into set of table schema. ***(R.G.P.V., Dec. 2016)***

Ans. Fig. 1.18 shows an E-R diagram for the given problem. In the fig.1.18, there are three entities patient, doctor, and test-examination. The entity patient has attributes p_id, p_name, age, weight, gender, address, illness. The entity doctor has attributes d_id, d_name, specialization, phone, address. The entity test_examination has attribute t_no, t_name, date, recommended_by, lab_name, result.

The tables for this diagram can be written as relation as follows –

Doctor (d_Id, d_name, specialization, phone, address)

Patient (p_id, p_name, age, weight, gender, illness, address)

Test_examination (t_no, t_name, date, recommended_by, lab_name, result)

Treats (d_id, p_id)

Observes (d_id, t_no)

Carried_on (t_no, p_id)

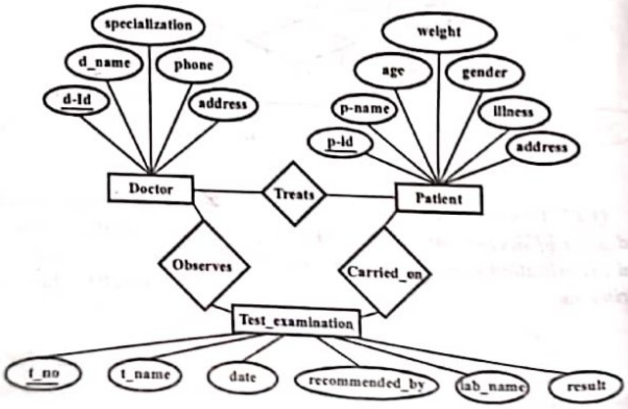


Fig. 1.18

Q.48. Draw an E-R diagram for a small marketing company database. Assume suitable data. ***(R.G.P.V., June 2016)***

Ans.

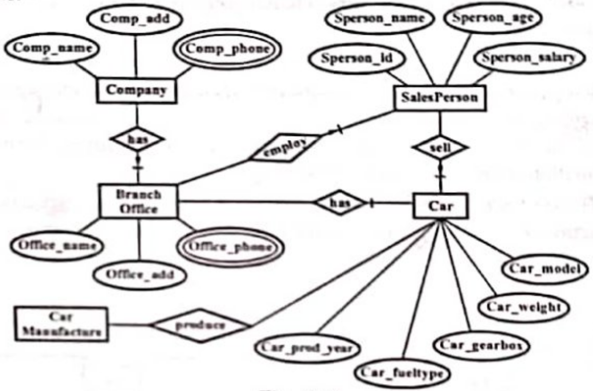


Fig. 1.19

Q.49. Construct an ER diagram of customer account relationship. Customer entity with attributes SS#, customer name, street, customer city and account entity with attributes account no. and balance. The customer account relationship with date attributes. ***(R.G.P.V., Dec. 2012)***

Ans. The ER diagram of customer account relationship is shown in fig. 1.20.

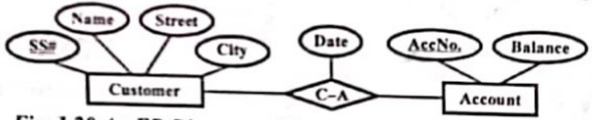


Fig. 1.20 An ER Diagram of Customer Account Relationship

Q.50. Differentiate between user defined and attribute defined specialization. ***(R.G.P.V., Nov./Dec. 2007)***

Ans. If all subclasses in a specialization have the membership condition on the same attribute of the superclass, the specialization itself is called an ***attribute-defined specialization*** and the attribute is called the defining attribute of the specialization. When we do not have a condition for determining membership in a class, the subclass is called ***user-defined***.

Q.51. Define the concept of aggregation. Give several examples of where this concept is useful. ***(R.G.P.V., Dec. 2014)***

Ans. One limitation of E-R model is that it cannot express relationships among relationships. For example, consider the ternary relationship works-on between a employee, branch and job. Now, suppose we want to reco managers for tasks performed by an employee at a branch. That is, we wa to record managers for (employee, branch, job) combinations. Let us assu that there is an entity set manager.

One approach for representing this relationship is to create a quarterna relationship ***manages*** between ***employee, branch, job*** and ***manager***. Th the resulting E-R diagram will be as shown in fig. 1.21. However, there redundant information in fig. 1.21, since every employee, branch, j combination in manages is also in works_on.

The best way to model such a situation is aggregation. Aggregation is abstraction through which relationships are treated as high-level entities. Th we regard the relationship set works_on as a higher-level entity set call works_on. Such an entity set is treated in the same manner as is any oth entity set. Thus, a binary relationship manages is created between works_ and manager to represent who manages what tasks as shown in fig. 1.22.

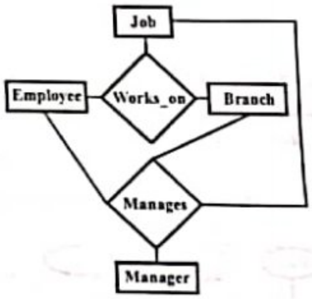


Fig. 1.21 E-R Diagram with Redundant Relationships

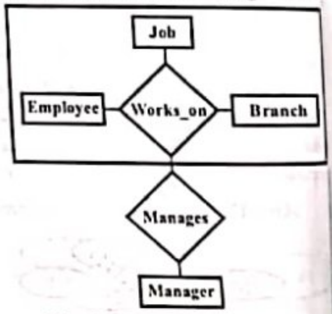


Fig. 1.22 E-R Diagram with Aggregation

Q.52. Discuss the concept of specialization and generalization i enhanced entity-relationship model using an example.

Or

Differentiate between specialization and generalization.

(R.G.P.V., Dec. 2003, June 2005, Nov./Dec. 200

Ans. Specialization is the process of defining a set of subclasses of entity type. This entity type is called the superclass of the specialization. T set of subclasses that form a specialization is defined on the basis of so distinguishing characteristic of the entities in the superclass. For example, t set of subclasses {secretary, engineer, technician} is a specialization of t superclass employee that distinguishes among employee entities based on t job type of each entity. There may have several specializations of the sa

entity type based on different distinguishing characteristics. For example, another specialization of the employee entity type may be the set of subclasses {Salaried_employee, Hourly_employee}. This specialization distinguishes among employees based on the method of pay.

Fig. 1.23 shows how we represent a specialization diagrammatically in an EER diagram. The subclasses that define a specialization are attached by line to a circle, which is connected to the superclass. The ***subset symbol*** on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship. Attributes that apply only to entities of a particular subclass – such as typing speed of secretary – are attached to the rectangle representing that subclass. These are called ***specific or local attributes*** of the subclass. Similarly, a subclass can participate in specific relationship types, such as the Hourly_employee class participating in the Belongs_to relationship in fig. 1.23.

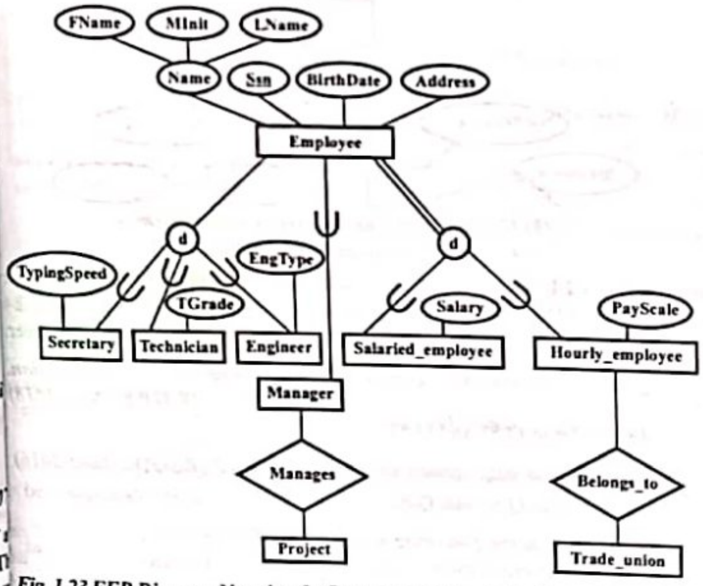
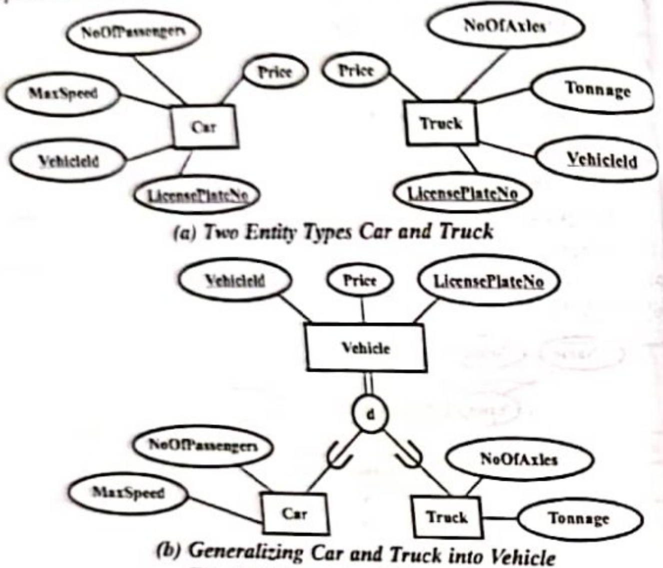


Fig. 1.23 EER Diagram Notation for Representing Specialization and Subclasses

Generalization is the reverse process of specialization in which we suppress the differences among several entity types, identify their common features, and generalize them into a single superclass of which the original entity types are special subclasses. For example, consider the entity types car and truck shown in fig. 1.24 (a). They can be generalized into the entity type vehicle shown in fig. 1.24 (b). Now, both car and truck are subclasses of the generaliz superclass vehicle.



(a) Two Entity Types Car and Truck

(b) Generalizing Car and Truck into Vehicle

Fig. 1.24 Examples of Generalization

In fig. 1.24, we can view {car, truck} as specialization of vehicle, rath than viewing vehicle as a generalization of car and truck. Similarly, in fig. 1.2 we can view employee as a generalization of secretary, technician and enginee

Q.53. Briefly explain the generalization, aggregation and specializatio

(R.G.P.V., Nov. 201

Ans. Refer to Q.51 and Q.52.

Q.54. What is aggregation and specialization ? (R.G.P.V., June 2016)

Ans. Refer to Q.51 and Q.52.

Q.55. Explain the following terms with example –
(i) Generalization ***(ii) Strong and weak entity***
(iii) Role indicators.

(R.G.P.V., Dec. 2005

Ans. (i) Generalization – Refer to Q.52.

(ii) Strong and Weak Entity – Refer to Q.38.

(iii) Role Indicators – Refer to Q.44.

Q.56. Design a generalization-specialization hierarchy for a motor-vehicle sales company. The company sells motorcycles which have an engine number and cost; cars which have a chassis number, and engine number, seating capacity and cost; trucks which have chassis number, an engine number and cost.

(R.G.P.V., Dec. 2010)

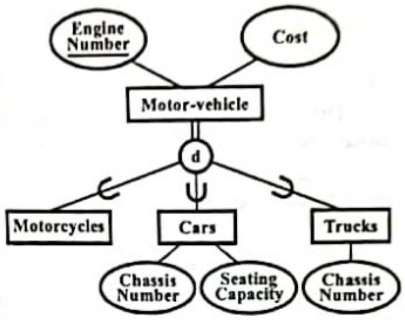


Fig. 1.25

Ans. A generalization-specialization hierarchy for the given motor-vehicle sales company is shown in fig. 1.25.

Q.57. Explain the followings –
(i) Mapping cardinalities (ii) Participation constraints
(iii) Attribute inheritance.

(R.G.P.V., Dec. 2013)

Ans. (i) Mapping Cardinalities – Mapping cardinalities can be defined as the number of entities to which another entity is associated via a relationship set. Binary relationship sets are described easily by using mapping cardinalities.

For a binary relationship set R between entity sets 1 and 2, the mapping cardinality can be one of the following –

(a) One to One – This is expressed as an association between an entity in set 1 to at most one entity in set 2, and an entity in set 2 to at most one entity in set 1.

(b) One to Many – This is expressed as an association between an entity in set 1 to any number (zero or more) of entities in set 2. However, an entity in set 2 can be associated with at most one entity in set 1.

(c) Many to One – This is expressed as an association between an entity in set 1 to at most one entity in set 2. However, an entity in set 2 can be associated with any number (zero or more) of entities in set 1.

(d) Many to Many – This is expressed as an association between an entity in set 1 to any number (zero or more) of entities in set 2, and an entity in set 2 to any number (zero or more) of entities in set 1.

(ii) Participation Constraints – The participation of an entity set in a relationship set is expressed as a total participation if each entity in entity set participates in at least one relationship in relationship set. If only some entities in entity set participate in relationship in relationship set, the participation of entity set in relationship set is expressed as a partial participation.

(iii) Attribute Inheritance – Attribute inheritance is an excellent property of the lower-and higher-level entities created by specialization and generalization. In this case, the lower-level entity sets inherit the attributes of the higher-level entity sets. A lower-level entity set also inherits participation in the relationship sets in which its higher-level entity participates.

The result is usually the same whether a predetermined portion of an E-R model was found out by specialization or generalization –

(a) A higher-level entity set with relationships and attributes that apply to all of its lower-level entity sets.

(b) Lower-level entity sets with different-2 characteristics that apply in the scope of a specific lower-level entity set.

Q.58. What do you mean by mapping cardinalities ? Explain various type of cardinalities. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.57 (i).

Q.59. Explain how an ER-schema is transformed to tables.

Or

Explain the steps for reduction of E-R model into relational model. ***(R.G.P.V., June 2008, Dec. 2010)***

Or

Explain the tabular representation of the following –

(i) Strong entity set (ii) Weak entity set

(iii) Relationship sets (iv) Generalization. ***(R.G.P.V., Dec. 2013)***

Ans. An ER-schema can be represented by tables as follows –

(i) Tabular Representation of Strong Entity Sets – Let E be a strong entity set with descriptive attributes $a_1, a_2, ..., a_n$. This entity can be represented by a table called E with n distinct column, each of which corresponds to one of the attributes of E. For instance, the entity set of loan having two attributes ***loan_number*** and ***amount*** (see fig. 1.11) is represented by a table called ***loan*** with two columns, as in fig. 1.26.

Loan_Number	Amount
L-11	900
L-14	1500
L-15	1500
L-16	1300
L-17	1000
L-23	2000
L-93	500

Fig. 1.26 The Loan Table

(ii) Tabular Representation of Weak Entity Sets – Let A be a weak entity set with attributes $a_1, a_2, ..., a_m$. Let B be the strong entity set on which A depends. Let the primary key of B consist of attributes $b_1, b_2, ..., b_n$. The entity set A is represented by a table called A with one column for each attribute of the set – $\{a_1, a_2, ..., a_m\} \cup \{b_1, b_2, ..., b_n\}$.

For instance, the entity set ***payment*** (in fig. 1.8) has three attributes ***payment_number, payment_date,*** and ***payment_amount***. The primary key of the ***loan*** entity set, on which ***payment*** depends, is ***loan_number***. Thus ***payment*** is represented by a table with four columns having the same name as attributes, as in fig. 1.27.

Loan_Number	Payment_Number	Payment_Date	Payment_Amount
L-11	53	7 June 2001	125
L-14	69	28 May 2001	500
L-15	22	23 May 2001	300
L-16	58	18 June 2001	135
L-17	5	10 May 2001	50
L-17	6	7 June 2001	50
L-17	7	17 June 2001	100
L-23	11	17 May 2001	75
L-93	103	3 June 2001	900
L-93	104	13 June 2001	200

Fig. 1.27 The Payment Table

(iii) Tabular Representation of Relationship Sets – Let R be a relationship set, let $a_1, a_2, ..., a_m$ be the set of attributes formed by the union of the primary keys of each of the entity sets participating in R, and let the descriptive attributes (if any) of R be $b_1, b_2, ..., b_n$. This relationship set is represented by a table called R with one column for each attribute of the set –

For example, consider the relationship set *borrower* in the E-R diagram of fig. 1.11. This relationship set involves the following two entity sets –

(a) *customer,* with the primary key *customer_id.*

(b) *loan,* with the primary key *loan_number.*

Since the relationship set has no attributes, the *borrower* table has two columns, labeled *customer_id* and *loan_number,* as shown in fig. 1.28.

customer_id	loan_number
019-28-3746	L-11
019-28-3746	L-23
244-66-8800	L-93
321-12-3123	L-17
335-57-7991	L-16
555-55-5555	L-14
677-89-9011	L-15
963-96-3963	L-17

Fig. 1.28 The borrower Table

(iv) Tabular Representation of Multivalued Attributes – Generally, attributes in an E-R diagram map directly into columns for the appropriate tables. However, multivalued attributes are an exception; new tables are created for these attributes. For a multivalued attribute M, we create a table T with a column C that corresponds to M and columns corresponding to the primary key of the entity set or relationship set of which M is an attribute. For example, consider the E-R diagram in fig. 1.29. The diagram includes the multivalued attribute *dependent_name*. For this multivalued attribute, we create a table *dependent_name*, with columns *dname*, referring to the *dependent-name* attribute of *employee* and *employee_id*, representing the primary key of the entity set *employee*. Each dependent of an employee is represented as a unique row in the table.

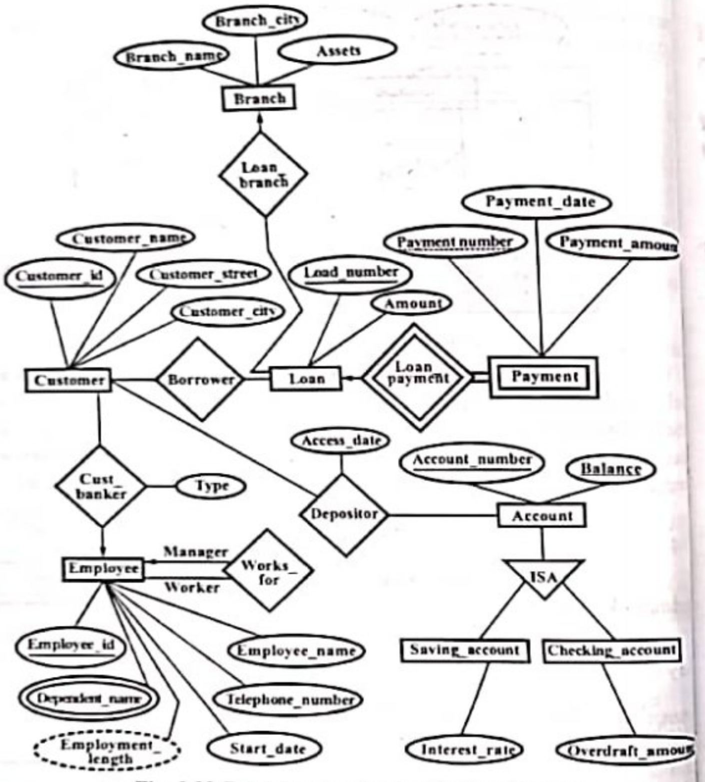


Fig. 1.29 E-R Diagram for a Banking Enterprise

(v) There are two ways for transforming to a tabular form an E-R diagram that includes generalization. Fig. 1.30 shows the generalization. We have chosen to simplify only the first tier of lower-level entity sets, i.e., *savings_account* and *checking_account.*

(a) Create a table for the higher-level entity set. For each lower-level entity set, create a table that includes a column for each of the attributes of that entity set plus a column for each attribute of the primary key of the higher-level entity set. Thus, there are three tables for fig. 1.30.

1. *account,* with attributes *account_number* and *balance.*

2. *savings_account,* with attributes *account_number* and *interest_rate.*

3. *checking_account,* with attributes *account_number* and *overdraft_amount.*

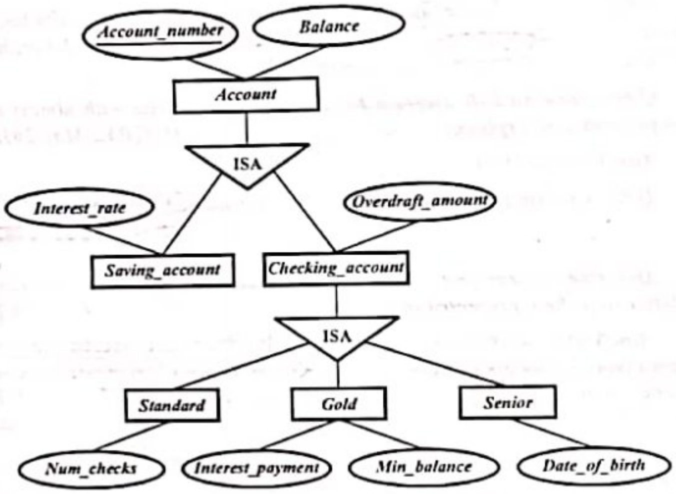


Fig. 1.30 Specialization and Generalization

(b) If the generalization is disjoint and complete, i.e., if no entity is a member of two lower-level entity sets directly below a higher-level entity set, and if every entity in the higher-level entity set is also a member of one of the lower-level entity sets. Then, an alternative representation is possible. Here, no table is created for the higher-level entity set. Instead, for each lower-level entity set, a table is created that includes a column for each of the attributes of that entity set plus a column for each attribute of the higher-level entity set. Then, there are two tables for the fig. 1.30 –

1. *savings_account,* with attributes *account_number, balance,* and *interest_rate.*

2. *checking_account,* with attributes *account_number, balance,* and *overdraft_amount.*

The *savings_account* and *checking_account* relations corresponding to these tables both have *balance* as the primary key.

If the second method were used for an overlapping generalization, some values like *balance* would be stored twice unnecessarily. Similarly, if the generalization were not complete, i.e., if some accounts were neither savings nor checking accounts, then such accounts could not be represented with the second method.

(vi) Tabular Representation of Aggregation – Transforming an E-R diagram containing aggregation to a tabular form is straightforward. Consider the diagram of fig. 1.22. The table for the relationship set *manages* between the aggregation of *works_on* and the entity set *manager* includes a column for each attribute in the primary keys of the entity set *manager* and the relationship set *works_on*. It would also include a column for any descriptive attributes, i they exist, of the relationship set *manages*. We then transform the relationshi sets and entity sets within the aggregated entity.

Q.60. Draw an E-R diagram for a banking enterprise with almost al components and explain. ***(R.G.P.V., May 2018***

Ans. Refer to Q.59.

Q.61. Explain the network database model with an example. ***(R.G.P.V, Dec. 2002***

Or

Describe records and sets in network model. How can one-to-many relationships be represented in this model ? ***(R.G.P.V., Dec. 2002***

Ans. The network data model represents data for an entity set by a logical record type. A network database consists of a collection of records connected to one another through links. In many respects, a record is similar to an E-R model. Each record is a collection of fields (attributes), each of which has only one data value. A link is an association between two records.

Let us consider a database that represents a customer-account relationship in a banking system. Two record types, customer and account are here. We can define record type customer as follows –

```
type customer = record
     customer-name : string;
     customer-street : string;
     customer-city : string;
 end
```

The record type account can be defined as follows –

```
type account = record
account-number : string;
balance : integer;
end
```

Fig. 1.31 shows a database sample, in which Hayes has account A-102, Johnson has accounts A-101 and A-201, Turner has account A-305.

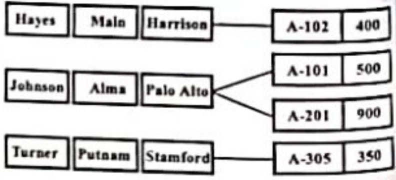


Fig. 1.31 Database Sample

To represent the design of a network database, a data-structure diagram is a schema. Such a diagram involves two components i.e., boxes correspond to record types and lines correspond to links. This diagram serves the same purpose as E-R diagram i.e., specifies overall logical structure of the database. Let us consider on fig. 1.32.

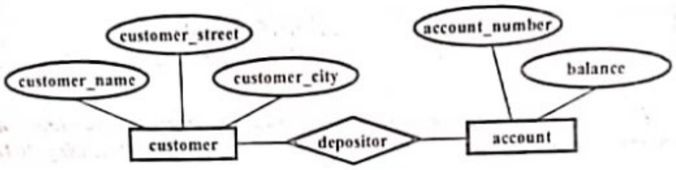


Fig. 1.32 E-R Diagram

Here, E-R diagram consists of two entity sets i.e., ***customer*** and ***account***, related through many-to-many relationship *depositor* with no descriptive attributes. Diagram specifies, a customer can have several accounts but an account belongs only one customer. Its corresponding data-structure diagram is shown in fig. 1.33.

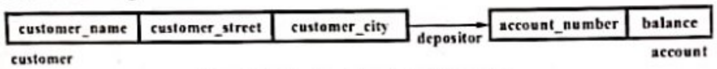


Fig. 1.33 Data-structure Diagram

Here, record type customer corresponds to entity set ***customer*** and it has three fields i.e., customer_name, customer_street and customer_city. The record type account corresponds to entity set ***account*** and it has two fields, account_number and balance. The relationship ***depositor*** is replaced with link depositor.

If relationship ***depositor*** were one-to-many from ***customer*** to ***account***, then link depositor would have an arrow pointing to record type customer. Similarly, if relationship ***depositor*** were one-to-one, then link depositor would have two arrows, one pointing to recored type account and other one pointing to record type customer.

Q.62. Explain the concept of relational data model in brief.

Ans. Relational data model is based on the collection of tables. The user of the database system may query these tables, insert new tuples, delete tuples and update tuples. To express these operations, there are several languages.

The relational data model, like all data models, consists of three basic components –

(i) A set of domains and a set of relations
(ii) Operations on relations
(iii) Integrity rules.

In relational database systems, attributes correspond to fields. For given application an attribute may only be allowed to take a value from a set of permissible values. This set of allowable values for the attribute is the domain of the attribute. A domain may be structured or unstructured.

The unstructured (atomic) domains are general sets, such as sets of integers, real numbers, character strings and so on. These are sometimes referred to as application-independent domains. Structured (composite) domains can be specified as consisting of nonatomic values. The domain for attribute address, for instance, that specifies street number, street name, city, state and zip or postal code is considered as composite domain.

An entity type having n attributes can be represented by an ordered set of these attributes called an n-tuple. If these n attributes take values from domains $D_1, D_2, \ldots, D_n$. The representation of entity must then be a member of the set $D_1 \times D_2 \times \ldots \times D_n$, as resulting set of this cartesian product has all possible ordered n-tuples. Conceptually, a relation is represented as a table. Each column of table represents an attribute and each row represents a tuple of the relation. In relational model, we represent the entity by a relation and use a tuple to represent an instance of the entity. Different instances of an entity type are distinguishable.

Codd defined a "relationally complete" set of operations and the collection of these, that take one or more relations as their operand(s), forms the basis of relational algebra. To manipulate relations, there are number of operations. Relations can be derived from other relations or a number of relations can be combined to define a new relation. The transformation of relations is useful in getting results from database.

Relational model involves two integrity rules. These implicitly or explicitly define the set of consistent database states, or changes of states or both. Other integrity constraints can be specified in terms of dependencies during database design. First integrity rule is concerned with primary key values. If any attribute of a primary key were permitted to have null values then attributes in key must be nonredundant, the key cannot be used for unique identification of tuples. This rule is also referred to as entity rule. Second integrity rule is concerned with foreign keys i.e., with attributes of a relation having domains that are those of the primary key of another relation. This rule is also referred as referential integrity rule.

Fig. 1.34 shows representation of a relation as a table.

Q.63. Why is the hierarchical data model considered inflexible ?

(R.G.P.V., Dec. 2015)

Ans. The hierarchical model provides a straightforward and natural method of implementing a one-to-many relationship. This structure is difficult to implement and extra indexing or data duplication is required. In the hierarchical model, the links are hard coded into the data structure, that is, the link is permanently established and cannot be modified. The hard coding makes the hierarchical model very inflexible.

APPLICANT –

Name	Age	Profession
John Doe	55	Analyst
Mirian Taylor	31	Programmer
Abe Malcolm	28	Receptionist
Adrian Cook	33	Programmer
Liz Smith	32	Manager

Fig. 1.34 Example Representation of a Relation as a Table

UNIT 2

RELATIONAL DATA MODELS – DOMAINS, TUPLES, ATTRIBUTES, RELATIONS, CHARACTERISTICS OF RELATIONS, KEYS, KEY ATTRIBUTES OF RELATION, RELATIONAL DATABASE, SCHEMAS, INTEGRITY CONSTRAINTS, REFERENTIAL INTEGRITY, INTENSION AND EXTENSION

Q.1. Discuss the terms attribute and domain in relational model.

Ans. An object or entity is characterized by its properties or attributes. In conventional file systems the term field refers to the smallest item of data with same practical meaning i.e., a field is used to capture some specific property of the object. In relational database systems, attributes correspond to fields. For each attribute there is a set of permitted values, called the domain of that attribute.

A domain D is a set of atomic values. It means that each value in the domain is indivisible. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain to help in interpreting its values.

Some examples of domains are as follows –

(i) USA_phone_numbers – The set of 10-digit phone numbers valid in the united states.

(ii) Local_phone_numbers – The set of 7-digit phone numbers valid within a particular area code in the United States.

The preceding are called logical definitions of domain. A data type or format is also specified for each domain. For example, the data type for the domain USA_phone_numbers can be declared as a character string of the form (ddd) ddd-dddd, where each d is a numeric digit and the first three digits form a valid telephone area code. The data type for employee_ages is an integer number between 15 and 80. Thus, a domain is given a name, data type, and format.

A relation schema R, denoted by R $(A_1, A_2, \ldots, A_n)$, is made up of a relation name R and a list of attributes $A_1, A_2, \ldots, A_n$. Each attribute A_i is the name of a role played by some domain D in the relation schema R. D is called the domain of A_i and is denoted by dom (A_i). A relation schema is used describe a relation. R is called the name of this relation. The degree o relation is the number of attributes n of its relation schema.

Q.2. What do you mean by attributes ? *(R.G.P.V., Dec. 201*

Ans. Refer to Q.1.

Q.3. Discuss the correspondence between ER model constructs and relation model constructs. Show how each ER model construct can be mapped relational model and also discuss alternative mappings. (R.G.P.V., Nov. 201

Ans. The correspondence between ER model constructs and relation model constructs –

S.No.	*ER Model*	*Relational Model*
(i)	Entity type	"Entity" relation.
(ii)	1 : 1 or 1 : N relationship type	Foreign key.
(iii)	M : N relationship type	"Relationship" relation and tw foreign keys.
(iv)	n ary relationship type	"Relationship" relation and foreign keys.
(v)	Simple attributes	Attributes
(vi)	Composite attributes	Set of simple component attribute
(vii)	Multivalued attributes	Relation and foreign key
(viii)	Value set	Domain
(ix)	Key attribute	Primary key or secondary key.

ER model construct can be mapped to relational model by combining th primary keys of the entities involved in a relation and it's attributes, if any.

Mapping an EER Schema to an ODB Schema – To design the typ declarations of object classes for an ODBMS from an EER schema that contain neither categories nor n-ary relationship with n > 2, is relatively easy. Mappin from EER to ODL is given as follows –

Step I – Create an ODL class for each EER entity type or subclass. I this mapping multivalued attributes are declared by using the set, bag or li constructors. In this composite attributes are mapped into a tuple constructo

Step II – Into ODL classes which are participating in the relationship add relationship properties or reference attributes. These may be created i one or both directions.

Depending on the cardinality ratio of the binary relationship, the relationship properties or reference attributes may be single valued or collectio types; single valued for binary relationships in the 1 : 1 or N : 1 directions and collection types for relationships in the 1 : N or M : N direction.

If relationship attributes exist, a tuple constructor (struct) can be used to create a structure of the form <reference, relationship attributes >, which may be included instead of the reference attribute.

Step III – Include appropriate operations for each class. These are not available from the EER schema and must be added to the database design by referring to the original requirements.

Step IV – An ODL class that corresponds to a subclass in the EER schema inherits (via EXTENDS) the type and methods of its superclass in the ODL schema.

Step V – Weak entity types can be mapped in the same way as regular entity types. An alternative mapping is possible for weak entity types that do not participate in any relationships except their identifying relationship.

Step VI – Categories in an EER schema are difficult to map to ODL. It is possible to create a mapping similar to the EER-to-relational mapping by declaring a class to represent the category and defining 1 : 1 relationships between the category and each of its superclasses.

Step VII – An n-ary relationship with degree n > 2 can be mapped into a separate class, with appropriate references to each participating class. These references are based on mapping a 1 : N relationship from each class that represents a participating entity type to the class that represents the n-ary relationship.

Q.4. What do you mean by the term tuple ? Explain briefly.

Ans. In relational model terminology, a row in a table, is known as a ***tuple***. An entity type having n attributes can be represented by an ordered set of these attributes called an ***n-tuple***. If these n attributes take values from the domains $D_1, D_2, ... , D_n$. The representation of the entity must then be a member of the set $D_1 \times D_2 \times .....\times D_n$, as the resulting set of this cartesian product contains all the possible ordered n-tuples.

A tuple is just like a record in conventional file systems and is used for handling entities and relationships between entities. Tuples are generally denoted by lowercase letters such as r, s, t, ... of the alphabet. An n-tuple t can be specified as

$$t = (a_1, ..., a_n)$$

where each a_i for $1 \le i \le n$ is a value in the D_i, and is the value of the attribute A_i in the tuple t. The order of the attributes is significant and fixed in the tuple representation. However, if we associate the attribute names with the corresponding values, we can relax the ordering requirement. Thus, the sets $\{(A_1 : a_1), (A_2 : a_2)\}$ and $\{(A_2 : a_2), (A_1 : a_1)\}$ are same. Therefore, a tuple can be viewed as a mapping from attribute names to values in the domains of the attributes.

Now, a tuple can be represented in a number of ways as follows –

$t = (a_1, ..., a_n)$ attribute value order must be constant

$$\left.\begin{array}{l} t=(a_1,....,a_n|A_1,....,A_n) \\ t=(|A_1,....,A_n|\ |a_1,....,a_n|) \\ t=((A_1:a_1),....,(A_n:a_n)) \\ t=((A_1|a_1),....,(A_n|a_n)) \end{array}\right\} \begin{array}{l}\text{attribute value can be}\\ \text{deduced from relative}\\ \text{ordering of the names}\\ \text{of the attribute}\end{array}$$

In the above formulations the a_i's are values drawn from D_i, the doma of A_i. The value of a tuple t over an attribute A_i is denoted as t [A_i], i.e., t[A_i] = It is known as the projection of the tuple t over A_i.

Q.5. What do you mean by a relation ? Discuss it.

Ans. A relation consists of a homogeneous set of tuples. A relation (relation state) r of the relation schema R (A_1, A_2, ..., A_n), also denoted r (R), is a set of n-tuples r = {t_1, t_2, ..., t_m}. Each n-tuple t is an ordered l of n values t = < v_1, v_2, ..., v_n >, where each value v_i, $1 \leq i \leq n$, is element of dom (A_i) or is a special ***null*** value.

The above definition of relation can be restated as follows –

A relation r (R) is a mathematical relation of degree n on the domains do (A_1), dom (A_2), ..., dom (A_n), which is a subset of the cartesian product the domains that define R –

$$r(R) \subseteq (dom(A_1) \times dom(A_2) \times .... \times dom(A_n))$$

The cartesian product specifies all possible combinations of values fro the underlying domain. Hence, if the number of values or cardinality of domain D is denoted by | D |, and all domains are finite, then total number tuples in the cartesian product is

$$| dom (A_1) | * | dom (A_2) | * .... * | dom (A_n) |$$

Q.6. Define degree of relation with example.

Or

Give the proper definition of the degree of relation.

(R.G.P.V., Dec. 2008, June 200

Ans. The ***degree***, also known as ***arity***, of a relation is number of attribute n of its relation schema.

Example – The relational schema of university students is given below i which there are 7 attributes. So degree of relation for this schema is 7.

STUDENT (Name, SSN, Home Phone, Address, Office Phone, Age, GPA)

Q.7. What is weak entity set and strong entity set and degree of a relatio with example ?

(R.G.P.V., May 201

Ans. Refer to Q.38 (Unit-I) and Q.6.

Q.8. What are the main properties of relation ?

Ans. The relation has some important properties which are as follows –

(i) There are no Duplicate Tuples – This property follows from the fact that the body of the relation is a mathematical set (of tuples) and set in mathematics does not include duplicate elements. And infact, it is useless to have same set of attributes more than once in the given relation. This property of relation clearly says that relation is different from table. It is so because tables allow the duplication of records while relation does not.

(ii) Tuples are Unordered, Top to Bottom – This property also follows from the fact that the body of the relation is a mathematical set, sets in mathematics are not ordered. Relations are independent of the order of tuples in it. Logically, it states that does not give preference to any one tuple over others. Thus, we cannot have numbering of the tuples. The relation gives equal emphasize in each of its tuples.

This property leads to another definition of relation. An alternative definition of a relation can be given, making the ordering of values in a tuple unnecessary. In this definition, a relation schema R = {A_1, A_2, A_3, ..., A_n} is a set of attributes and a relation (R) is a finite set of mapping r = {t_1, t_2, ..., t_m}, where each tuple t_i is a mapping from R to D and D is the union of the attribute domains, that is, D = dom (A_1) ∪ dom (A_2) ∪ .... ∪ dom (A_n). In this definition, t[A_i] must be in dom (A_i) for $1 \leq i \leq n$ for each mapping t in r, each mapping t_i is called a tuple. According to this definition, a tuple can be considered as a set of (<attribute>, <value>) pairs where each pair gives the value of the mapping from an attribute A_i to a value V_i from dom (A_i).

This property gives another distinction between the table and relation. In case of the table, it is mandatory for the database management system to give numbering to the records of the table. It is done to uniquely identify each record in the database and possess top-down ordering among the records. It is usually done on the basis of first-come, first-serve. The record stored earlier is shown first and the others follow them.

(iii) Attributes are Unordered, Left to Right – This property follows from the fact that the headings of a relation is also a set of attributes.

Ordering of attributes is also managed by the front end tool. It is who need to decide which attribute has to be placed and where. It is as per the requirement of the end user. This feature adds one more difference between relation and table, As in case of table, specific ordering of attributes (columns or fields) is done and is again usually on the basis of first come first serve, from left to right in the table.

(iv) Each Tuple Contains Exactly One Value for Each Attribute – This property states that each tuple is an atomic value, that is not divisible into components. Within the framework of the basic relational model. Hence, composite and multivalued attributes are not allowed. Thus relational model follows the process of normalization to overcome this problem.

Multivalued attributes must be represented by separate relations and composite attributes are represented by their component attributes.

But sometimes, the value of some attributes within a particular tuple may be unknown or may not be applied to that tuple. A special value, called NULL, is used for these cases. In general we can have several types of null values, such as "value unknown", "value exist but not available" or "attribute does not apply to this tuple".

Q.9. What do you understand by the term key ?

Ans. Entities have properties, called attributes, which associate a value from a domain of values for that attribute with each entity in the entity set. Usually the domain for an attribute will be a set of integers, real numbers or character strings, but we do not rule out other type of values. For example, the entities in the entity set of persons may be said to have attributes such as name (a character string), height (a real number) and so on.

The selection of relevant attributes for entity sets is another critical step in the design of a real-world model. An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a ***key*** for that entity set. In principle, each entity set has a key, since we hypothesized that each entity is distinguishable from all others. But if we do not choose, for an entity set, a collection of attributes that includes a key, then we shall not be able to distinguish one entity in the set from another. Often an arbitrary serial number is supplied as an attribute to serve as a key. For example, an entity set that included only U.S. nationals could use the single attribute "social-security-number" as a key.

Q.10. Define the term key attribute.

Ans. An entity type usually has an attribute whose values are distinct for each individual entity in the collection. Such an attribute is called a ***key attribute***, and its values can be used to identify each entity uniquely. For instance, for the PERSON entity type, a key attribute is social security number.

Q.11. Discuss the foreign key with suitable example.

Ans. A foreign key is a set of attributes of one relation R_2, whose values are required to match values of some candidate key of some other relations R_1.

For example, two relations *customer* (cust_id, cust_name, cust_add) and *order* (order_id, order_date, cust_id) where we store customer details and order details in two separate relations for reducing redundancy. The attribute *'customer_id'* in relation *'order'* is the foreign key. It allows the association of order with the respective entity in relation *'customer'*.

Q.12. Discuss the candidate key, primary key, super key, composite key and alternate key. ***(R.G.P.V., June 2010)***

Ans. (i) Candidate Key – A table can have more than one set of columns that could be chosen as the key. These are called candidate keys. For example, consider the relation Salespeople containing the following columns-Snum, Sname, Region and commission. From the list of columns it may appear that apart from Snum, Sname can also be a key. This assumption will prove right as long as we always have unique salesperson names. However, if we cannot make this assumption, Sname cannot be a candidate key.

(ii) Primary Key and Alternate Key – A relation can have more than one candidate keys. In this case, at least one of them should be chosen as the primary key and others are then called alternate keys.

For example, if we consider a relation ELEMENT with attributes NAME, SYMBOL and ATOMIC. Then we can choose SYMBOL as the primary key. NAME and ATOMIC# which are definitely candidate keys are considered as alternate keys.

In the cases where relation have only one candidate key, that only key can be considered as primary key.

(iii) Super Key – A super key is a set of columns that uniquely identifies every row in a table while a key is a minimal set of such columns. For example, consider a relation Employee containing just two columns-Emp_id, Emp_name. Then the two columns {Emp_id, Emp_name} together make up the super key. However, it is not a key because it is not a minimal set of columns.

(iv) Composite Key – There are situation when a single column cannot constitute a key. This is because a single column uniquely identify every row in a table. Instead, we need to have two or more columns together in order to identify every row in the table uniquely. A key consisting of two or more columns is called as a composite key.

For example, consider the relation Supplier (sup_id, part_id, Qty) tells us which supplier sells which part. As we know neither the sup_id nor the part_id can identify a row in the relation uniquely because a single supplier supply more than one part and also, a single part is supplied by more than one supplier. However, the two of them i.e., sup_id and part_id, together can easily identify any row in the table uniquely. Hence, it is a composite key.

Q.13. Explain various key constraints with example. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.11 and Q.12 (ii).

Q.14. Give the proper definition of null value. ***(R.G.P.V., June 2009)***

Ans. An attribute takes a null value when an entity does not have a value for it. The null value may indicate not applicable, i.e., the value does not exist for the entity. For instance, one may have no middle name.

Q.15. Explain relational databases and relational database schemas.

Ans. A relational database consists of a collection of tables, each of which is assigned a unique name. Each table has similar structure as in E-R databases. A row in a table represents a relationship among a set of values since a table is a collection of such relationship, there is a close correspondence between the concept of table and mathematical concept of relation, from which the relational data model takes its name. Actually, a relational database contains many relations, with tuples in relations that are related in various ways. A relational database schema S is a set of relation schemas S = {R_1, R_2, ..., R_m} and a set of integrity constraints. A relational database state DB of S is set of relation states DB = {r_1, r_2,...., r_m} such that each r_i is a state of R_i and such that the r_i relation states satisfy the specified integrity constraints.

For understanding database schema in a better way, we must differentia between the database schema or the logical design of the database, and database instance, which is a snapshot of the data in the database at a give instant in time. The concept of relation schema corresponds to the programmin language notion of type definition. It is convenient to give a name to a relatio schema. We use the convention of using lowercase names for relations an names beginning with an uppercase letter for relation schemas.

Following this notion, we use account-schema to denote the relatio schema for relation account.

Thus,

Account-schema = (branch_name, account_number, balance)

We denote the fact that account is a relation on Account-schema by account (Account-schema) as in fig. 2.1. In general, a relation schema comprises a list of attributes and their corresponding domains.

branch_name	*account_number*	*balanc*
Downtown	A-101	500
Mianus	A-215	700
Perryridge	A-102	400
Round Hill	A-305	350
Brighton	A-201	900
Redwood	A-222	700
Brighton	A-217	750

Fig. 2.1 The Account Relation

Q.16. Explain the characteristics of relation. Also explain the relationa databases. ***(R.G.P.V., Dec. 2014, 2015)***

Ans. Refer to Q.8 and Q.15.

Q.17. Describe entity integrity and referential integrity. Give an exampl of each. ***(R.G.P.V., Nov./Dec. 2007, Dec. 2010)***

Or

State two integrity rules. ***(R.G.P.V., Dec. 2015)***

Or

Explain integrity constraints.

(R.G.P.V., Dec. 2003, June 2007, June 2016)

Or

What do you mean by integrity constraints ?

(R.G.P.V., June 2006, Dec. 2016)

Ans. The term integrity refers to the accuracy or correctness of data in the database. Integrity constraints ensure that the changes made to the database by authorized users do not result in a loss of data consistency. Thus, the integrity constraints guard against accidental damage to the database. A database might be subject to any number of integrity constraints of arbitrary complexity.

Integrity constraints are specified on a database schema and are expected to hold on every database state of that schema. In addition to domain and key constraints, the relational model includes two general integrity rules – entity integrity and referential integrity. These integrity rules implicitly or explicitly define the set of consistent database states, or changes of state or both.

Integrity Rule 1 (Entity Integrity) – Integrity rule 1 is concerned with primary key values. The entity integrity constraint states that no primary key value can be null. This is because the primary key is used to identify individual tuples in a relation. Having null values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples have null for their primary keys, we cannot be able to distinguish them.

P :

Id	Name
101	Jones
103	Smith
104	Lalonde
107	Evan
110	Drew
112	Smith

(a) Relation without Null Values

P :

Id	Name
101	Jones
@	Smith
104	Lalonde
107	Evan
110	Drew
@	Lalonde
@	Smith

(b) Relation with Null Values

Fig. 2.2

Consider the relation P(P) in fig. 2.2 (a). The attribute Id is the primary key for P (P). If null values (represented as @) are permitted, then the two tuples < @, smith > are indistinguishable, even though they may represent two different instances of the entity type employee. Similarly, the tuples < @, Lalonde > and < 104, Lalonde >, are also indistinguishable and may be referring to the same person.

Integrity rule 1 specifies that instances of the entities are distinguishable and thus no prime attribute value may be null. This rule is also referred to as the entity rule. This rule can be formally stated as –

If attribute A of relation R(R) is a prime attribute of R(R), then A cannot accept null values.

Integrity Rule 2 (Referential Integrity) – Integrity rule 2 is concerned with foreign keys i.e., with attributes of a relation having domains that are those of the primary key of another relation. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.

Consider the example of employee and their managers. Since each employee has a manager and as managers are also employees, we may represent managers by their employee numbers, if the employee number is a key of the relation employee. Thus, manager attribute represents the

Emp#	Name	Manager
101	Jones	@
103	Smith	110
104	Lalonde	107
107	Evan	110
110	Drew	112
112	Smith	112

Fig. 2.3 Foreign Keys

employee number of the manager. Note that it is referring to the primary k of the same relation. An employee can only have a manager who is also employee, the chief executive officer (CEO) of the company can have him or herself as the manager or may take null values.

Q.18. Define the term referential integrity.(R.G.P.V., Dec. 2008, 20

Or

Define referential integrity constraints ? (R.G.P.V., Dec. 201

Ans. Refer to Q.17.

Q.19. What are integrity constraints ? Define the term primary k constraint and foreign key constraint. (R.G.P.V., Dec. 201

Ans. Refer to Q.17, Q.12 (ii) and Q.11.

Q.20. Explain the following keys with suitable example –

(i) Primary key (ii) Secondary key (iii) Foreign key (iv)Super k Explain referential integrity and integrity constraints.

(R.G.P.V., June 201

*Ans. (i) **Primary Key*** – Refer to Q.12 (ii).

(ii) Secondary Key – Refer to Q.12 (ii).

(iii) Foreign Key – Refer to Q.11.

(iv) Super Key – Refer to Q.12 (iii).

Referential Integrity and Integrity Constraints – Refer to Q.17.

Q.21 Discuss different types of keys. For each case, give a suitab example. What is foreign key constraint ? Why is such constraint importan

(R.G.P.V., Nov. 201

Ans. Refer to Q.11, Q.12 and Q.17.

The referential integrity constraint is specified between two relations an is used to maintain the consistency among tuples of the two relations.

Q.22. How the foreign key is used to maintain the referential integrity Explain by taking one example. (R.G.P.V., June 200

Ans. The concept of foreign key is useful to define referential integrity more formally. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2. A set of attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following two rules –

(i) The attributes in FK have the same domain(s) as the primary key attributes PK of R_2; attributes FK are said to reference to refer to the relation R_2.

(ii) A value of FK in a tuple t_1 of the current state r_1 (R_1) either occurs as a value of PK for some tuple t_2 in the current state r_2 (R_2) or is null. In the former case, we have t_1 [FK] = t_2 [PK], and we say that the tuple t_1 references or refers to the tuple t_2. R_1 is called the referencing relation and R_2 is the referenced relation.

In a database of many relations, there are many referential integrity constraints. To specify these constraints, it is first necessary to have a clear understanding of the meaning or role that each set of attributes plays in the various relation schemas of the database. Referential integrity constraints arise from the relationships among the entities represented by the relation schemas. For example, consider the database shown in fig. 2.4. In the employee relation, the attribute DNO refers to the department for which an employee works. Hence, we designate DNO to be a foreign key of EMPLOYEE, referring to the DEPARTMENT relation. This means that a value of DNO in any tuple t_1 of the EMPLOYEE relation must match a value of the primary key of DEPARTMENT – the DNUMBER attribute – in some tuple t_2 of the DEPARTMENT relation, or the value of DNO can be null if the employee does not belong to a department. In fig. 2.4 the tuple for employee 'John Smith' references the tuple for the 'Research' department indicating that 'John Smith' works for this department.

FNAME	LNAME	SSN	ADDRESS	SEX	SUPERSSN	DNO
John	Smith	123456789	731 Fondren, Houston, TX	M	333445555	5
Franklin	Wong	333445555	638 Voss, Houston, TX	M	888665555	5
Alicia	Zelaya	999887777	3321 Castle, Spring, TX	F	987654321	4
Jennifer	Wallace	987654321	291 Berry, Bellaire, TX	F	888665555	4
Ramesh	Narayan	666884444	975 Fire Oak, Humble, TX	M	333445555	5
Joyce	English	453453453	5631 Rice, Houston, TX	F	333445555	5
Ahmad	Jabbar	987987987	980 Dallas, Houston, TX	M	987654321	4
James	Borg	888665555	450 Stone, Houston, TX	M	Null	1

EMPLOYEE

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPARTMENT

Fig. 2.4 One Possible Relational Database State Corresponding to the Schema COMPANY

A foreign key can also refer to its own relation. For example, the attribute SUPERSSN in EMPLOYEE refers to the supervisor of an employee; this is another employee, represented by a tuple in the EMPLOYEE relation. Hence, SUPERSSN is a foreign key that references the EMPLOYEE relation itself.

Q.23. Define intension and extension, take suitable example of ea

Or

Differentiate between the intension and extension.

(R.G.P.V., June 2008, Dec. 2008, June 2009, Dec. 20

Ans. We know that a set is well-defined collection of objects and i represented by a list of elements (called members) or by the specification some membership condition.

The *intension* of a set defines the permissible occurrences by specifyi a membership condition.

The *extension* of the set specifies one of numerous possible occurrenc by explicitly listing the set members.

These two methods of defining a set are illustrated as follows –

Intension of set G = {g | g is an odd positive integer less than 20}

Extension of set G = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}

Q.24. Explain the following with examples –

(i) Super key ***(ii) Primary key***

(iii) Alternate key ***(iv) Extensions and intensions.***

(R.G.P.V., Dec. 201

Ans. ***(i) Super key*** – Refer to Q.12 (iii).

(ii) Primary key – Refer to Q.12 (ii).

(iii) Alternate key – Refer to Q.12 (ii).

(iv) Extensions and Intensions – Refer to Q.23.

RELATIONAL QUERY LANGUAGES – SQL-DDL, DML, INTEGRITY CONSTRAINTS, COMPLEX QUERIES, VARIOUS JOINS, INDEXING, TRIGGERS, ASSERTIONS

Q.25. What do you mean by relational query languages ?

Ans. A query language is a language in which a user requests informatio from the database.

Query languages can be categorized as being either procedural or non procedural. In procedural language, user instructs the system to perform a sequence of operations on the database to compute the desired result. In a non-procedural language, the user describes the information desired without giving a specific procedure for obtaining that information. The relational algebra is a procedural, whereas the tuple relational calculus and domain relational calculus are non-procedural.

In relational query languages, we are concerned with queries. A complete data-manipulation language includes not only a query language, but also a language for database modification. Such language includes command to insert and delete tuples as well as commands to modify parts of existing tuples. Very often, non-procedural query languages are used, as the database management systems are so powerful, that contain code for almost all type of requests that user may do. Usually, these DBMS have a procedural interface also, to be on the safer side.

For example, we have SQL as non-procedural query language but most DBMS like ORACLE provides language called PL/SQL which provides some basic facilities of procedural language along with the powerful advantages of SQL. While in case of non-procedural languages, we have a separate set of instructions and keywords to handle distinct operations. We have data definition language (DDL) for defining relation schemas, data manipulation language (DML) for manipulating the tuples in the relation, and DCL and embedded SQL are used as the supporting features.

Q.26. What do you understand by the term 'SQL' ? Explain.

Ans. SQL stands for structured query language. It is used to communicate with the database. Whenever any data is to be accessed from the database, we can do it through the SQL, and user can access the required data from the database, can manipulate data, can define the data in the database through SQL. SQL is supposed as structured query language for relational database management systems (RDBMS).

SQL uses the combination of relational algebra and relational calculus constructs. It has many other capabilities besides querying the database. It includes the features of defining the structure of the data, for modifying data in the database, and for specifying security constraints.

SQL has established as standard relational database language, and its original version was developed at IBM's San Jose Research Laboratory. It is also called SEQUEL and was implemented as part of the system R project. The purpose of this project was to validate the feasibility of relational model and to implement a DBMS based on this model. The results of this project are well documented in database literature. Additionally, to contributing to the concept of query compilation and optimization and concurrency control mechanisms, the most salient result of this project was the development of SQL.

SQL is a nonprocedural language. Users describe in SQL what they want to do and the SQL language compiler automatically generates a procedure to navigate the database and performs the desired task.

Q.27. What is SQL ? Which data models implements this language ? Write a SQL server program segment to write rules and defaults. How can you connect and disconnect them to a database ? ***(R.G.P.V., June 2011)***

Ans. SQL – Refer to Q.26.

Relational model implements this language.

The JDBC standard defines an API that Java programs can use to com to database servers. Fig. 2.5 shows a Java program that uses the JD interface. The program before opening a connection loads the appropri drivers for the database by using *Class.forName*, then open a connection t execute SQL statements. The parameter to the *getConnection* call speci the protocol to be used to communicate with the database (jdbc:oracle:thi the machine name where the server runs (aura.bell-labs.com), the port num uses for communication (2000). The parameter also specifies which sche on the server is to be used (bankdb). Note that JDBC specifies only the A not the communication protocol. A JDBC driver may support multiple protoc and we must specify one supported by both the database and the driver. U identifier and password are the two other arguments to *getConnection*.

```
public static void JDBCexample(String dbid, String userid, String passwd)
{
    try
    {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection conn=DriverManager.getConnection("jdbc:oracle:
                thin:@ aura.bell-labs.com:2000:bankdb", userid, passw
        Statement stmt=conn.createStatement( );
        try
        {
            stmt.executeUpdate("insert  into  table_name  values('val1
                                                    'val2', 'val3')"
        }
        catch(SQLException sqle)
        {
            System.out.println("Could not insert tuple:"+sqle);
        }
        ResultSet rset=stmt.executeQuery("Write Query");
        while(rset.next( ))
        {
            System.out.println(rset.getString("attribute_name") + " "
                                    rset.getFloat(attribute_number)
        }
        stmt.close( );
        conn.close( );
    }
    catch(SQLException sqle)
    {
        System.out.println("SQLException:" +sqle);
    }
}
```

Fig. 2.5 JDBC Code

Then, the program creates a statement handle on the connection and uses it to execute an SQL statement and get back results. The try and catch construct permit us to catch any exceptions that arise on calling JDBC calls and shows an appropriate message to the user. The program can update a query by using stmt.executeUpdate

The stmt.executeQuery is used to execute a query and retrieve the set of rows in the result into a ResultSet and fetch them one tuple at a time using the next( ) function on the result set. The values of attributes in a tuple can be retrieved in two ways – using the name of the attribute and using the position of the attribute.

Q.28. What are the various characteristics of SQL ? Discuss five aggregate functions with a suitable example. ***(R.G.P.V., Dec. 2011)***

Ans. **Characteristics of SQL** – The main characteristics of the SQL language is that it is a declarative or nonprocedural language. This implies that the programmer does not need to specify step-by-step all the operations that the computer needs to carry out to obtain a particular result. Instead, the programmer indicates to the DBMS what needs to be accomplished and then lets the system decide on its own how to obtain the desired result. The statements or commands that compose the SQL language are generally divided into two major categories or data sublanguages. Each sublanguage is concerned with a particular aspect of the language. One of these sublanguages known as the data definition language or DDL and the other is known as data manipulation language or DML. The SQL language can be used interactively or in its embedded form.

Aggregate Functions – Aggregate functions are functions that take a collection of values as input and return a single value. In SQL, we have five aggregate functions –

(i) Average : avg	(ii) Minimum : min
(iii) Maximum : max	(iv) Count : count
(v) Total : sum.	

Here, sum and avg functions work only for numbers, while min, max, and count work equivalently well for alphabets i.e., string.

(i) Average: avg – This function takes as input, a collection of numbers and returns a single number, which is average of all the numbers.

For example, suppose we want to find average of amount of orders in order table, then query can be written as –

```
select avg(amt)
from orders
```

This query will result a single attribute named avg(amt) having only one value that is average of all amounts. If we wish to get avg with heading of field as average then query should be written as follows –

```
select avg(amt) average
from orders
```

Now, we will have only one attribute with heading average and only one value that is average of amount from order field.

We can also use group by clause. The group by clause is useful in circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples.

For example,

```
select snum, avg(amt)
from orders
group by snum
```

This query will result average amount for each snum.

We can also use having clause in SQL, for applying some c restrictions.

For example,

```
select snum, avg(amt)
from orders
group by snum
having avg(amt) > 1200
```

The predicates in the 'having' clause are applied after the formatio groups, so the aggregate functions may be used.

This query will give us only snum which contains average amount m than 1200. The 'having' clause is used only with the aggregate functions

(ii) ***Minimum : min*** – This function is used to find minim number from a group of numbers. For example, suppose we want to find amount with minimum value from orders table –

```
select min(amt)
from orders
```

This query will give us with the minimum amount i.e., 18.69.

(iii) ***Maximum : max*** – This function is used to find maxim number from a given group of numbers. For example, suppose we wis find maximum amount from orders table –

```
select max(amt)
from orders
```

This query will result with only one attribute that contains only one va i.e., the maximum amount in order table. In our example is 9891.88.

(iv) ***Count : count*** – This function is used to return *total* num of values in a given attribute, means total count. For example, if we wan find total number of snum in orders table then

```
select count(snum)
from orders
```

Here, count counts the number of snum in the table and displays the to number of snum. In our example, this value will be 10.

But, here some of the snum are repeating. For example, 1001, 10 1003, etc. So, if we wish to count one snum only once, then we can u distinct in the following way –

```
select count(distinct snum)
from orders
```

In our example, this value is 5.

If we wish to find total orders in orders table, then we can also write as –

```
select count(*)
from orders
```

This will result a value 10.

(v) ***Total : sum*** – This function is used to return the sum of numbers. This function just adds all the numbers given to it as input and return the sum of these numbers. For example, if we want to find the sum of amounts of orders from orders table then –

```
select sum(amt)
from orders
```

If we want to find the sum of amount for each snum separately, then we have to write as

```
select snum, sum(amt)
from orders
group by snum
```

This will give us snum with total of amounts for each snum separately. If a 'where' clause and a 'having' clause appear in the same query, the predicate in 'where' clause is applied first. The tuples satisfying the 'where' predicate are then placed into groups by the group by clause. The 'having' clause if present is then applied to each group, the groups that do not satisfy the having clause predicate are removed. The remaining groups are used by the select clause to generate the tuples of the result of the query.

Q.29. What is aggregate functions of SQL ? ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.28.

Q.30. Write the commands of DDL. ***(R.G.P.V., June 2016)***

Ans. The three main commands of DDL are –

(i) ***Create Table*** – For creating a table or view.

(ii) ***Drop Table*** – It is used for dropping a relation i.e., when drop command is used it delets all the tuples along with the schema of that relation.

(iii) ***Alter Table*** – For adding attribute to an existing i.e., for altering a relation.

Q.31. Explain data definition language in reference to SQL.

Ans. Data definition language (DDL) provides commands for defining relation schemas, deleting relations, creating indices, and modifying relation schemas.

The DDL portion consists of those declarative constructs of PL/1 that are needed to declare database objects – The DECLARE (DCL) statement itself, certain PL/1 data types, possibly special extensions to PL/1 to support new objects that are not handled by existing PL/1.

SQL DDL allows specification of a set of relations along with t information about each relation, including –

(i) The schema of each relation.

(ii) The domain of values associated with each attribute.

(iii) The integrity constraints.

(iv) The set of indices to be maintained for each relation.

(v) The security and authorization information for each relation.

(vi) The physical storage structure of each relation on disk.

SQL DDL can be discussed on the basis of following topics –

(i) Domain Types in SQL – SQL provides a variety of in-bu domain types. It also provides a facility to user to define their own data type.

While declaring these data types, we also have to specify length of variabl as given below –

(a) char(n) – It is used for character strings having fixed leng n. The value of n is to be given by user.

(b) varchar(n) – This is also used to specify character string but it has variable length with user specified maximum length n. It's full for is character varying, is equivalent to varchar(n).

(c) int – It is used to specify integer (a finite subset of th integers that is machine-dependent). Its full form is integer.

(d) smallint – It is used to specify small integer (a machin dependent subset of the integer domain type).

(e) numeric(p, d) – It is used to specify number having fixed point with user-specified precision. It has total p digits (plus a sign), in whic d digits are right to decimal and (p–d) digits are left to the decimal point. Fo example, the numeric (5, 2) allows 321.45 to be stored exactly, but neithe 3214.5 nor 32.145 can be stored exactly in a field of this type.

(f) real, double Precision – It is used to specify real number i.e., floating point and double-precision floating-point numbers, with machine dependent precision.

(g) float(n) – It is used to specify floating point numbers. It has user specified precision of at least n digits.

(h) date – It is used to specify calender date. It contains (four digits) year, month and day of the month. By default format of oracle is "DD-MM-YY".

(i) time – It is used to specify time of day, in hours, minutes and seconds.

(j) long – It stores in the ASCII text format. It has capacity upto 4GB.

(k) BLOB – It is binary large object.

(*l*) CLOB – It is character large object.

(m)NCLOB – It is multiset character large object. It is also called nationalised character set.

SQL also provides a data type called interval and allows computations based on dates and times, and on intervals.

For example, if A and B are of type date, then A–B is an interval whose value is the number of days from date A to date B.

Similarly, adding or subtracting an interval to a date or time gives back a date or time, respectively.

(ii) Schema Definition in SQL – In DDL, we have mainly three types of commands –

(a) create Command – We define an SQL relation using the create table command –

```
create table < tablename >
(column1 datatype [constraints],
column2 datatype [constraints],.....);
```

In above syntax, symbol < > specifies that a value is to be passed there by user, and in above example, <tablename> indicates that user should give the name of relation there.

Then names of columns should be written, separated by commas and followed by data type (i.e., the domain type of values in the domain of attribute).

The allowed constraints include primary key ($A_{j1}, A_{j2}, \ldots, A_{jm}$) and check (P) and foreign key and Not Null, unique key etc.

(b) drop Command – This command is used to drop the relation with its structure and the data. Its syntax is –

```
drop table <table name>
or
drop table r
```

Here, r is the name of relation.

If we want to remove rows from the table, then we can use delete command for it.

(c) alter Command – This command is used to add the attributes or to modify the existing attributes. The form of the alter command is as follows –

```
alter table < table name> add <A> <D>
```

Here A is the name of attribute to be added and D is the data type of the attribute A.

Q.32. What are database languages ? Why were the DDL and DML called data sublanguage ? Also explain functional dependency.

(R.G.P.V., Dec. 2012)

Ans. **Database Languages** – In many DBMSs where no strict separation of levels is maintained, the DBA and database designers use one language, known as the data definition language (DDL) to define both the conceptual and the internal schemas for the database. The DBMS will have a DDL compiler wh function is to process DDL statements in order to recognize descriptions of t schema constructs and to store the schema description in the DBMS catalo

DDL Sublanguage – The DDL is required to specify the concept schema only when a clear separation is maintained between the conceptu and internal levels. Another language, the storage definition language (SDL), used to specify the internal schema. Either one of these languages may speci the mappings between the two schemas. For a true three-schema architectu we would require a third language, the view definition language (VDL), specify user views and their mappings to the conceptual schema, but in mo DBMSs the DDL is used to define both conceptual and external schemas.

DML Sublanguage – A data manipulation language is provided by the DBM for manipulating operations such as retrieval, insertion, deletion, and modificati of the data. There are two main types of DMLs, namely, high-level nonprocedural DML and low-level or procedural DML. A high-level DML ca be used on its own to specify complex database operations in a concise mann A low-level DML needs to use programming language constructs, such as loopin to retrieve and process each record from a set of records.

Whenever DML commands, whether high-level or low-level, are embedd in a general-purpose programming language, that language is called the ho language and the DML is called the data sublanguage.

Functional Dependency – Refer to Q.17 (Unit-III).

Q.33. What is structured query language ? How the DDL and DML a different ? Explain. ***(R.G.P.V., Dec. 201***

Ans. Refer to Q.26 and Q.32.

Q.34. What is the difference between procedural DML and non procedural DML ? ***(R.G.P.V., Dec. 201***

Ans. Refer to Q.32.

Q.35. What is NULL ? Give an example to illustrate testing for NULL in SQL. ***(R.G.P.V., June 2008, Dec. 2008, 2010***

Ans. SQL allows the use of null values to indicate absence of informatio about the value of an attribute. The keyword null is used in a predicate to tes for a null value. Thus, to find all loan numbers that appear in the *loan* relatio with null values for amount, we write

select ***loan_number*** from ***loan*** where ***amount*** is null;

The predicate *is not null* tests for the absence of a null value.

The use of a null value in arithmetic and comparison operations cause several complications. The result of an arithmetic expression is null if any o the input values in null. SQL treats as unknown the result of any compariso involving a null value.

Q.36. Describe the different types of file organization. Explain using a sketch of each of them with their advantages and disadvantages. ***(R.G.P.V. June 2010, Dec. 2011)***

Ans. A file is organized logically as a sequence of records. These records are mapped onto disk blocks. Files are provided as a basic construct in operating system. Although blocks are of a fixed size determined by the physical properties of the disk and by the operating system record sizes vary.

One approach to mapping the database to files is to use several files, and to store records of only one fixed length in any given file. An alternative is to structure files such that we can accommodate multiple lengths for records. Files of fixed-length records are easier to implement than are files of variable-length records.

(i) Fixed-length Records – Suppose a file of account records of bank database shown in fig. 2.6. Each record of a file can be defined as –

```
type deposit = record
        acc_no : char(12)
        b_name : char(20)
        amount : real;
    end
```

Let each character occupies 1 byte and a real occupies 8 bytes, our account record is 40 bytes long. A simple approach is to use the first 40 bytes for the first record, next 40 bytes for the next and so on. But, this approach suffer from two problems –

(a) To delete a record from this structure is difficult. The space occupied by the deleted record must be filled with some other record of the file or we must have a way of marking deleted records so that they can be ignored.

(b) Unless the block size happens to be a multiple of 40, some records will cross block boundaries. That is, part of the record will be stored in one block and part in another. It would thus require two block accesses to read or write such a record.

rec 0	A-601	Round Hill	100
rec 1	A-501	Perryridge	200
rec 2	A-201	Brighton	750
rec 3	A-101	Mianus	450
rec 4	A-502	Perryridge	800
rec 5	A-102	Brighton	850
rec 6	A-503	Perryridge	600
rec 7	A-401	Downtown	300
rec 8	A-301	Redwood	250

Fig. 2.6 Account Records File

rec 0	A-601	Round Hill	100
rec 1	A-501	Perryridge	200
rec 2	A-201	Brighton	750
rec 3	A-101	Mianus	450
rec 5	A-102	Brighton	850
rec 6	A-503	Perryridge	600
rec 7	A-401	Downtown	300
rec 8	A-301	Redwood	250

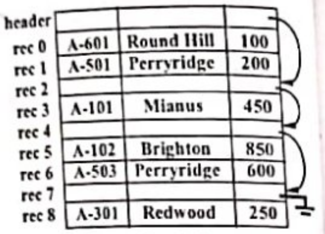
Fig. 2.7 Account Records File On Deleting Record 4

On deleting a record, we could move the record that came after it into the space formerly occupied by the deleted record and so on until every

record following the deleted record has been moved ahead (see fig. 2.
This approach requires moving a large number of records. But, it might
easier to move the final record of the file into the space occupied by t
deleted record (see fig. 2.8).

rec 0	A-601	Round Hill	100
rec 1	A-501	Perryridge	200
rec 2	A-201	Brighton	750
rec 3	A-101	Mianus	450
rec 8	A-301	Redwood	250
rec 5	A-102	Brighton	850
rec 6	A-503	Perryridge	600
rec 7	A-401	Downtown	300

Fig. 2.8



header			
rec 0	A-601	Round Hill	100
rec 1	A-501	Perryridge	200
rec 2			
rec 3	A-101	Mianus	450
rec 4			
rec 5	A-102	Brighton	850
rec 6	A-503	Perryridge	600
rec 7			
rec 8	A-301	Redwood	250

Fig. 2.9

It is undesirable to move records to occupy the space freed by a delet
record, since doing so needs extra block accesses. Since insertion tend to
more frequent than deletions, it is acceptable to leave open the space occupi
by the deleted record and to wait for a subsequent insertion before reusing t
space. A simple marker on a deleted record is not sufficient, since it is hard
find this available space when an insertion is being done. Therefore, we nee
to introduce an additional structure.

We assign a fixed number of bytes as a file header at the beginning of th
file. The header will contain a variety of information about the file. For no
all we need to store there is the address of the first record whose contents a
deleted. We use this first record to store the address of the second availab
record and so on. Thus, the deleted record form a linked list called free lis
Fig. 2.9 depicts the account record file with the free list after deleting t
records 2, 4 and 7.

When a new record is inserted, we use the record pointed through th
header and update the header pointer to point to the next available record
When no space is remain, we append the new record to the end of fil
Fixed-length records files are simple to implement because the space mad
available by a deleted record is similar to the space needed to insert
record.

(ii) Variable-length Records – In a database systems, variable-lengt
records exist in various ways–storage of multiple record types in a file, recor
types that permit repeating fields and record types that permit variable length
for one or more fields. Assume a different representation of the accou
information file in which we use one variable-length record for each branc
name and for all the account information for that branch. The format of th
record is defined as –

```
type acc_list = record
    b_name : char(20);
    acc_info : array[1....∞] of
        record;
            acc_no : char(12);
            amount : real;
        end
end
```

Here, acc_info is defined as an array with an arbitrary number of elements. There is no limit on how large a record can be. There exist different techniques for implementing variable-length records.

(a) Byte-string Representation – In this method, a special end-of-record (⊥) symbol is appended to the end of each record. Afterwards, we store each record as a string of consecutive bytes. A byte-string representation of variable-length records is shown in fig. 2.10. A variant method of byte-string representation, rather than employing end-of-record symbols, stores the record length at the starting of each record.

0	Mianus	A-101	450	⊥				
1	Brighton	A-201	750	A-102	850	1		
2	Redwood	A-301	250	⊥				
3	Downtown	A-401	300	⊥				
4	Perryridge	A-501	200	A-502	800	A-603	600	⊥
5	Round Hill	A-601	100	⊥				

Fig. 2.10 Byte-string Representation

Some disadvantages of byte-string representation are as follows –

(1) To reuse space occupied via a deleted record is not easy. However, techniques exist to manage insertion and deletion, they lead to a large number of small fragments of disk storage that are wasted.

(2) In general, there is no space for records to grow longer. When a variable length record becomes longer, it must be moved–movement is costly if pointer to the record are stored elsewhere in the database, since the pointers must be located and updated.

Hence, the byte-string representation is not usually employed for implementing variable-length records. Although, there is a modified form of the byte-string representation known as slotted-page structure.

(b) Fixed-length Representation – In a file system, an another way to implement variable-length record is to use one or more fixed-length records to represent one variable-length record. This can be done in two ways –

(1) List Representation – Variable-length records a represented through fixed length records, chained together by pointers.

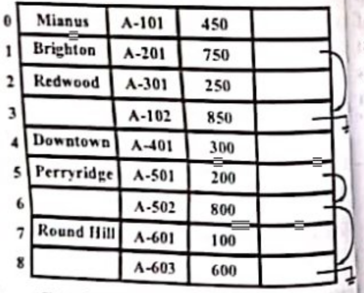
(2) Reserved Space – When there is a maximum reco length that is never exceeded, we can use fixed-length records of that lengt Unused space is filled with a end-of-record symbol.

Fig. 2.11 shows the file when we permitted a maximum of three accoun per branch for reserved-space method. In the file, a record is of the account-li type but with the array having exactly three elements. The branches with less tha three accounts have records with null fields for which we use the symbol ⊥.

0	Mianus	A-101	450	⊥	⊥	⊥	⊥
1	Brighton	A-201	750	A-102	850	⊥	⊥
2	Redwood	A-301	250	⊥	⊥	⊥	⊥
3	Downtown	A-401	300	⊥	⊥	⊥	⊥
4	Perryridge	A-501	200	A-502	800	A-603	600
5	Round Hill	A-601	100	⊥	⊥	⊥	⊥

Fig. 2.11 Reserved-space Method

When records have a length close to the maximum, then the reserved-space method is useful. Otherwise, a significant amount of space may be wasted. In our example, some of the branches can have more accounts compared to others. This situation is handled by linked list. To represent the file through the linked list method, we add a pointer field. Fig. 2.12 shows the resulting structure.



0	Mianus	A-101	450	
1	Brighton	A-201	750	
2	Redwood	A-301	250	
3		A-102	850	
4	Downtown	A-401	300	
5	Perryridge	A-501	200	
6		A-502	800	
7	Round Hill	A-601	100	
8		A-603	600	

Fig. 2.12 File Using Linked List

A disadvantage of the linked list structure is that we waste space in al records except the first in a chain. The first record needs to have the branch-name value, but subsequent records do not. This wasted space is significan because we expect that each branch has a large number of accounts. To handle this problem, we use two different types of blocks –

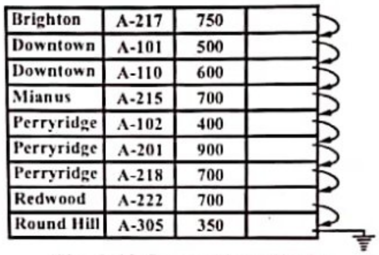
(1) Anchor Block – It contains the first record of a chain

(2) Overflow Block – It contains records other than those that are the first record of chain.

Q.37. What do you mean by indexing ? Also write types of indexing.

Ans. The idea behind an ordered index access structure is similar to that behind the index used in a textbook, which lists important terms at the end of the book in alphabetical order along with a list of page numbers where the term appears in the book. We can search an index to find a list of addresses – page numbers in this case – and use these addresses to locate a term in the textbook by searching the specified pages.

For a file with a given record structure consisting of several fields (or attributes), an index access structure is defined on a single field of a file, called, an *indexing field* (or *indexing attribute*). The index stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value. The values in the index are ordered so that we can do a binary search on the index.



Brighton	A-217	750	
Downtown	A-101	500	
Downtown	A-110	600	
Mianus	A-215	700	
Perryridge	A-102	400	
Perryridge	A-201	900	
Perryridge	A-218	700	
Redwood	A-222	700	
Round Hill	A-305	350	

Fig. 2.13 Sequential File for Account Records

There are several types of ordered indexes.

A file may have several indices, on different search key. If the file records are in sequential ordered, the index whose search key specifies the sequential order of the file is the ***primary index*** means index on a primary key. It is also called *clustering index*. Indices whose search key specifies an order different from the sequential order are called ***secondary indices*** or *nonclustering indices*.

(i) Primary Index – Here we assume that all files are ordered sequentially on some search key. They are designed for application that require both sequential processing of the entire file and random access to individual records. Fig. 2.13 shows a sequential file of account records for bank example, as branch_name is a search key.

(ii) Secondary Indices – Generally, a secondary index on a candidate key looks just like a dense primary index, except that the records are not stored sequentially. However, secondary indices may be structured differently from primary indices. If the search key of a primary index is not a candidate key. It suffices if the index points to the first record with a particular value for the search key, since the other records can be fetched by sequential scan of the file. It is given in fig. 2.14.

If the search key of a secondary index is not a candidate key, it is not enough to point to just the first record with each search key vlaue. The remaining record with the same search key value could be anywhere in the file, since the records are ordered by the search key of the primary index rather than by the

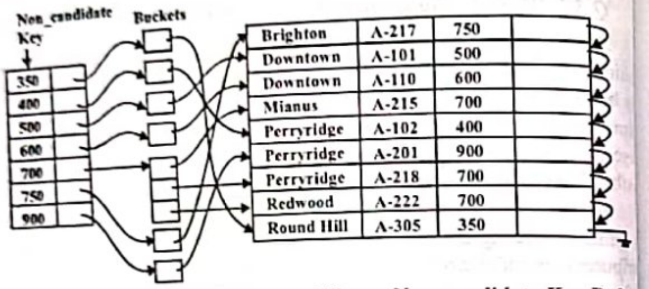


Fig. 2.14 Secondary Index on Account File, on Non-candidate Key Balan

search key of the secondary index. So, secondary index must contain pointe to all the records. Secondary level can uses an extra level of indirection implement secondary indices on search key that are not candidate key. Tl pointer of secondary index do not point to the file directly, instead, each poi to a bucket that contains pointers to the file. It is clear from fig. 2.14 that, t structure of a secondary index that uses an extra level of indirection on t account file, search key as balance.

In principle, a database system can decide automatically what indices create. However, because of the space cost of indices, as well as the effect indices on update processing, it is not easy to automatically make the rig choices about what indices to maintain. Therefore, most SQL implementatio provide the programmer control over creation and removal of indices throu data-definition-language commands.

We create an index by **create index** command, which takes the form

create index <index-name> on <relation-name> (<attribute-list>

Q.38. Write short note on triggers. ***(R.G.P.V., June 2011, Nov. 201***

Or

What is SQL triggers ? ***(R.G.P.V., June 201***

Or

Explain the trigger feature of ORACLE. ***(R.G.P.V., Dec. 2008, 201***

Ans. A trigger is a procedural SQL code that is automatically invoked the RDBMS upon the occurrence of a given data manipulation event.

The syntax of a trigger in Oracle is –

```
CREATE OR REPLACE TRIGGER trigger_name
[BEFORE/AFTER] [DELETE/INSERT/UPDATE OF column_name] O
table_name
```

```
[FOR EACH ROW]
[DECLARE]
    [variable_name data type [: = initial_value]]
BEGIN
    PL/SQL instructions;
...
END;
```

Thus, a trigger definition contains following parts –

(i) The Triggering Timing – BEFORE or AFTER. This timing indicates when the trigger's PL/SQL code executes. In this case, before or after the triggering statement is completed.

(ii) The Triggering Event – The statement that causes the trigger to execute (INSERT, UPDATE or DELETE).

(iii) The Triggering Level – There are two types of triggers – statement level triggers and row-level triggers.

A *statement-level trigger* is assumed if you omit the FOR EACH ROW keywords. This type of trigger is executed once, before or after the triggering statement is completed. This is the default case.

A *row-level trigger* requires use of the FOR EACH ROW keywords. This type of trigger is executed once for each row affected by the triggering statement (In other words, if you update 10 rows, the trigger executes 10 times).

(iv) The Triggering Action – The PL/SQL code enclosed between the BEGIN and END keywords. Each statement inside the PL/SQL code must end with a semicolon ";".

It is useful to remember that –

(i) A trigger is invoked before or after a data row is inserted, updated, or deleted.

(ii) A trigger is associated with a database table.

(iii) Each database table may have one or more triggers.

(iv) A trigger is executed as part of the transaction that triggered it.

Oracle recommends triggers for –

(i) Auditing purposes

(ii) Automatic generation of derived column values

(iii) Enforcement of business or security constraints

(iv) Creation of replica table for backup purposes.

Q.39. What are triggers ? Explain with suitable example. How can th help in building robust database ? ***(R.G.P.V., Dec. 201***

Ans. **Triggers** – Refer to Q.38.

Triggers are useful mechanism for altering or to perform certain tas automatically when certain conditions are met. For example, suppose th instead of allowing negative account balance, the bank treats overdrafts setting the account balance to zero, and creating a loan in the amount of overdraft. This new loan is given a loan number identical or same as acco number of the overdrawn account.

For this example, the condition for executing the trigger is an update the account relation that results in the negative balance value.

Suppose Pawan withdraw some money from his account that resulted to be negative the account balance. Now let t denote the account tuple wit negative value of balance, then the actions to be taken are as follows –

(i) Insert a new tuple r in the loan relation with –
r[branch_name] = t[branch_name]
r[loan_number] = t[account_number]
r[amount] = – t[balance]

(ii) Insert a new tuple u in the borrower relation with –
u[customer_name] = "Pawan"
u[loan_number] = t[account_number]

(iii) Set t[balance] to 0.

Now there is SQL query to invoke the above example –

```
create trigger overdraft after update on account T
(if new T.balance <0)
then (insert into loan values
(T.branch_name, T.account_number, – new T.balance)
insert into borrower
(select customer_name, account_number
from depositor
where T.account_number = depositor.account_number)
update account U
set U.balance = 0
where U.account_number = T.account_number))
```

Q.40. What is assertion ?

Ans. An assertion is a predicate expressing a condition that we wish t database always to be satisfied. Referential-integrity constraints and dom constraints are special form of assertions. We have to pay more attention these forms of assertion, because they easily tested and applied to wide ran of database applications. Even there are many constraints that cannot expressed by using only these special forms. For example –

(i) The sum of all loan amounts for each branch must be less than the sum of all account balance at the branch.

(ii) Every loan have at least one customer who maintains an account with a minimum balance of $500.00.

The assertion can be written in the following form –

create assertion <*assertion_name*> **check** <*predicate*>

Whenever an assertion is created, the system tests it for validity and if the assertion is valid then any modification to the database is allowed. The assumption should be used with great care.

If we want to delete any assertion the following command is used –

drop assertion <assertion_name>

NUMERICAL PROBLEMS

Prob.1. Let the following relational schema be given –

Employee (SSN, name, age, dno)

Salary (SSN, salary)

Works_on (Projects#, SSN)

Project (Projects#, project_name, location)

For each of the following queries give an expression in SQL –

(i) Display the names of projects at "Delhi".

(ii) Find the project_name of employee whose salary is greater than 10000.

(iii)Retrieve the name and SSN of employees working on project# A 100.

(R.G.P.V., June 2010)

Sol. (i) The SQL query is as follows –

select project_name from Project
where location = "Delhi"

(ii) select project_name from Project where Projects# in (select Projects# from Works_on, Salary where Works_on.SSN = Salary.SSN having Salary > 10000);

(iii) select name, SSN from Employee,
where SSN in(select SSN from Works_on where Projects# = A 100);

***Prob.2.** Consider the relations –*

EMP (ENO, ENAME, AGE, BASIC)
WORK_ON (ENO, DNO)
DEPT (DNO, DNAME, CITY)

Express the following queries in SQL –

***(i)** Find names of employees whose basic pay is greater th average basic pay.*

***(ii)** Find the sum of the basic pay of all the employees, th maximum basic pay, the minimum basic pay and the average basic pay.*

(R.G.P.V., Dec. 201

***Sol.** (i)*
```
select ENAME from EMP
where BASIC > (select avg(BASIC) from EMP)
```

(ii)
```
select sum(BASIC) "SUM", max(BASIC) "MAX",
min(BASIC) "MIN", avg(BASIC) "AVG" from EMP
```

***Prob.3.** Consider the following relations with keys underlined –*

Street (<u>name</u>, location, city)
House (<u>number</u>, street_name)
Lives (<u>name</u>, house_number)

Define the above relations as tables in DDL SQL making real wor assumptions about the type of the fields. Define the primary keys and th foreign keys. ***(R.G.P.V., June 200***

Sol.
```
create table Street
    (name varchar2 (25) not null,
        location varchar2 (30),
        city varchar2 (15),
        primary key (name));
create table House
    (number int not null,
        street_name varchar2 (25) not null,
        primary key (number),
        foreign key (street_name) references Street (name));
create table Lives
    (name varchar2 (25) not null,
        house_number int not null,
        primary key (name),
        foreign key (house_number) references House (number));
```

***Prob.4.** Consider the following relations with keys underlined –*

Street (<u>name</u>, location, city)
House (<u>number</u>, street_name)
Lives (<u>name</u>, house_number)

***(i)** Define the above relations as tables in SQL making real world assumptions about the type of fields. Define the primary keys and foreign key.*

***(ii)** For the above relations answer the following queries in SQL –*

***(a)** Get the names of persons who live in the street named 'Mahatma Gandhi'.*

***(b)** Get the house numbers street wise.*

***(c)** Get the numbers of houses which are not occupied.*

(R.G.P.V., June 2008)

***Sol. (i) Tables in SQL** –* Refer to Prob.3.

***(ii)** (a)* The SQL statement is as follows –
```
select name from House, Lives where
    House. number = Lives. house_number AND
        street_name = 'Mahatma Gandhi';
```

(b) The SQL statement is as follows –
```
select number, street-name from House order by street_name;
```

(c) The SQL statement is as follows –
```
select number from House where number not in
    (select house_number from Lives);
```

***Prob.5.** Consider the following employee database –*

Employee (Emp_name, Street, City)
Works (Emp_name, Company_name, Salary)
Company (Company_name, City)
Manages (Emp_name, manager _name)

Write expressions in SQL for the following queries –

***(i)** Find all employees who live in the city where the company for which they work is located.*

***(ii)** Find all employees who live in the same city and on the same street as their managers.*

***(iii)** Find all employees in the database who do not work for ABC corporation.*

***(iv)** Find all employee who earn more than every employee of ABC corporation.*

(v)** Find all company names located in every city in which ABC corporation is located.* ***(R.G.P.V., Dec. 2006)

Sol. (i) The SQL expression is as follows –

```
select Emp_name from Employee, Works, Company
where Employee.Emp_name = Works.Emp_name
and Works.Company_name = Company.Company_name
and Employee.City = Company.City;
```

(ii) The SQL expression is –

```
select Emp_name from Employee
where Employee.street AND Employee.city IN
(select Street, City from Employee, Manages
where Emp_name = Manager_name
```

(iii) The SQL statement is –

```
select Emp_name from Employee where
Emp_name NOT IN
(select Emp_name from Works where Company_name = 'ABC
```

(iv) The SQL statement is

```
select Emp_name from Works where salary >
(select max(salary) from Works where Company_name = 'ABC
```

(v) The SQL statement is as follows –

```
select Company_name from Company where City IN
(select City from Company where Company_name = 'AB
corporation');
```

Prob.6. Consider the following Relational Database –

Employee (employee_name, street, city)
Works (employee_name, company_name, salary)
Company (company_name, city)
Manages (employee_name, Manager_name)

Write SQL for the following queries –

(i) Find the names of all employees in this database who live i the same city as the company for which they work.

(ii) Find the names of all employees who live in the same city an on the same street as do their Manager.

(iii) Find the names of all employees in this database who do n work for the company 'FBC'

(R.G.P.V., Dec. 2008, June 200

Sol. (i) Refer to Prob.5 (i)

(ii) Refer to Prob.5 (ii)

(iii) The SQL expression is as follows –

```
select employee_name from Employee where employee_name NOT I
(select employee_name from Works where company_name = 'FBC')
```

Prob.7. Consider the employee data. Give an expression in SQL for the following query –

Employee (employee-name, street, city)
Works (employee-name, company-name, salary)
Company (company-name, city)
Manages (employee-name, manager-name)

(i) Find the name of all employees who work for State Bank.

(ii) Find the names and cities of residence of all employees who work for State Bank.

(iii) Find all employee in the database who donot work for State Bank.

(iv) Find all employee in the database who earn more than every employee of UCO Bank.

(R.G.P.V., Dec. 2016)

Sol. The SQL queries are as follows –

(i)

```
Select employee-name from Works
where company-name = "State Bank"
```

(ii)

```
Select employee-name, city from Employee, Works
where Employee.employee-name = Works.employee-name
and company-name = "State Bank"
```

(iii)

```
Select employee-name from Employee
where employee-name NOT IN
(select employee-name from Works
where company-name = "State Bank")
```

(iv)

```
Select employee-name from Works
where salary > (select max (salary) from Works
where company-name = "UCO Bank")
```

Prob.8. Consider the following database with primary keys underlined –

Employee database
Empl (E_no, E_name, No)
Dept (D_no, D_name)
Emp_sal (E_no, Basic, DA, HRA, Tot-sal)

Represent the following queries in SQL –

(i) Get the names of all employees whose total salary is more than Rs. 10,000.

(ii) Get Dept. names of employees who earn more than Rs. 50,000 (Tot-salary).

(iii) Get the names of employees whose HRA is nil.

(iv) Write SQL DDL for the above given employee database, where primary key are underlined.

(R.G.P.V., Dec. 2009)

Sol. (i) Select E_name from Empl where E_no IN
(Select E_no from Empl_sal where Tot_sal > 10,000)

(ii) Select D_name from Dept where D_no in
(Select NO from Empl where E_no IN
(Select E_no from Empl_sal where Tot_sal > 50,000))

(iii) Select E_name from Empl where E_no IN
(Select E_no from Empl_sal where HRA = 0.00)

(iv) The SQL DDL for the given employee database are as follo

```
create table Empl
(E_no        INT              NOT NULL
E_name       VARCHAR(25)
NO           INT              NOT NULL
PRIMARY KEY (E_no));

create table Dept
(D_no        INT              NOT NULL
D_NAME       VARCHAR(20)
PRIMARY KEY (D_no)
Unique (D_name)               FOREIGN KEY (D_no)
                              References Empl (NO));

create table Emp_sal
(E_no        INT              NOT NULL
Basic        DECIMAL (8, 2)
DA           DECIMAL (4, 2)
HRA          DECIMAL (6, 2)
Tot_Sal      DECIMAL(10, 2)
PRIMARY KEY (E_no)
FOREIGN KEY (E_no) References Empl (E_no));
```

Prob.9. For the relations PROJECT, EMPLOYEE and ASSIGN_T database, express the following queries in SQL –

PROJECT (Project #, Project_Name, chief_Incharge)
EMPLOYEE (Emp #, Empname)
Assign_to (Project #, Emp #)

(i) Get details of employees working on all projects.

(ii) Get employee numbers of employees who work on at least those projects that employee 107 works on.

(iii) Get the names and employees who are not assigned any projec

(iv) Get all pairs of employee numbers such that the two employe are working on the same project.

(v) Get the employee names of all employees who work on project P

(R.G.P.V., June 200

Sol. (i) The SQL expression is as follows –
select empname, project # from EMPLOYEE, Assign_to
where employee.emp # = Assign_to.emp #;

(ii) The SQL expression is as follows –
select emp # from EMPLOYEE, Assign_to where employee.emp # in (select project # from assign_to where emp # = 107);

(iii) The SQL expression is as follows –
select emp #, empname from EMPLOYEE where emp # NOT IN (select emp # from Assign_to);

(iv) The SQL expression is as follows –
select emp #, Project # from EMPLOYEE, Assign_to
where EMPLOYEE.emp # = Assign_to.emp #
group by project #;

(v) The SQL expression is as follows –
select emp #, empname where emp # in (select emp # from Assign_to where project # = 'P7');

Prob.10. Consider the following database with primary key underlined.

Employee (<u>ENO</u>, DOB, Name, Address, Sex, Salary, Dept-no)
Department (Dept-no, Dept-Name)

For each of the following queries give expression in SQL

(i) Retrieve the names of employees in department-5.

(ii) Retrieve the names of all employees who are not in department-5.

(iii) Retrieve the average salary of all female employees.

(iv) Write SQL DDL statements of above database.

(R.G.P.V., Dec. 2013)

Sol. (i) The SQL expression is as follows –
Select Name from Employee where Dept-No. = 5;

(ii) The SQL expression is as follows –
Select Name from Employee where Dept-No. != 5;

(iii) The SQL expression is as follows –
Select avg(Salary) from Employee where Sex = 'Female';

(iv) The SQL, DDL statements are as follows –

```
create table Employee
(ENO         INT              NOT NULL,
DOB          DATE,
Name         VARCHAR(20),
Address      VARCHAR(30),
Sex          VARCHAR(6),
Salary       DECIMAL(10, 2),
Dept-No         INT             NOT NULL,
PRIMARY KEY (ENO));

create table Department
(Dept-No.       INT             NOT NULL,
Dept-Name  VARCHAR(15)          NOT NULL,
PRIMARY KEY(Dept-No.),
UNIQUE (Dept_Name));
```

Prob.11. Consider the following database (supplier part database) wi primary key underlined –

S(SNO, SName, city, status)

P(PNO, PName, city, weight, color)

SP(SNO, PNO, Qty) relational algebra

Give expression in SQL for the following question –

(i) Get SNames for suppliers who do not supply part 'P2'.

(ii) Double the status of 'LONDON' suppliers.

(iii) Delete all 'PARIS' suppliers and corresponding shipments.

(iv) Get SNames for suppliers who supply at least one 'red' part.

(v) Get all pairs of suppliers who are located in same city.

(vi) Get SNames for suppliers who supply at least one part supplie by supplier 'S2'. *(R.G.P.V., June 200*

Sol. (i) The SQL expression is as follows –

```
Select distinct S.SName
from S
where NOT EXISTS
    (select *from SP
    where S.P.SNO = S.SNO
    AND SP.PNO = 'P2');
```

(ii) The SQL expression is as follows –

```
update S
set status = 2*status
where city = 'London';
```

(iii) The SQL expression is as follows –

```
Delete from SP
where city = 'Paris' AND SP.SNO IN (select SNO from S wher
S.SNO = SP.SNO);
```

(iv) The SQL expression is as follows –

```
select distinct S.SName
From S
where S.SNO in
(select SP.SNO
from SP
where SP.PNO in
(select P.PNO from P where P.color = 'Red'));
```

(v) The SQL expression is as follows –

```
select A.SNO as SA, B.S# as SB
from S as A, S as B
where A.city = B.city
AND A.SNO < B.SNO;
```

(vi) The SQL expression is as follows –

```
select SName from S, SP
where S.SNO = SP.SNO and PNO in (select PNO from SP where
SNO = 'S2')
```

Prob.12. Consider the following database with primary keys underlined –

Employee (ENO, DOB, ADDRESS, SEX, SALARY, NAME, DNO SUPER END).

Department (DNO, DName, MGRENO, MGR start date)

Dept-location (DNO, Dlocation)

Project(PNAME, PNO, Plocation, DNO)

Works-ON(ENO, PNO, hours)

Dependent (ENO, dep-name, sex, bdate, relationship)

For each of the following queries give expressions in SQL and relational algebra –

(i) Retrieve the names of all employees in department 5.

(ii) List the names of all employees who have no dependents.

(iii) Retrieve the names of all employees who do not work on any project.

(iv) Retrieve the average salary of all female employee.

(v) Find names of all employees who work on at least one project located in Hyderabad.

(vi) Write SQL, DDL statements for the above database.

(R.G.P.V., Nov./Dec. 2007)

Sol. (i) The SQL expression is as follows –

```
select NAME from Employee where DNO = 5;
```

Its corresponding relational algebra expression is as follows –

$\Pi_{NAME}(\sigma_{DNO = 5}(EMPLOYEE))$

(ii) The SQL expression is as follows –

```
select NAME from Employee
where not exists
(select *from Dependent where Employee.ENO = Dependent.E
```

Its corresponding relational algebra expression is as follows –

ALL_EMPS ← Π_{ENO} (EMPLOYEE)

EMP_WITH_DEPS(ENO) ← Π_{ENO} (DEPENDENT)

EMP_WITHOUT_DEPS ← (ALL_EMPS_EMPS_WITH_DE

RESULT ← Π_{NAME} (EMP_WITHOUT_DEPS*EMPLOYE

(iii) The SQL expression is as follows –

```
select NAME from employee
where not exists(select * from works-on
                    where employee.ENO = works-on.EN
```

Its corresponding relational algebra expression is as follows –

$\Pi_{ENO \cdot NAME}(Employee)\_\Pi_{ENO}$ (works_ON)

(iv) The SQL expression is as follows –

```
select avg(salary)from employee where SEX = 'Female';
```

Its corresponding relational algebra expression is as follows –

$\mathcal{G}_{avg(salary)}$ ($\sigma_{SEX = 'Female'}$ (EMPLOYEE))

(v) The SQL expression is as follows –

```
select NAME from employee, project where ENO in
(select * from project, works-on where
project.PNO = works-on.PNO
AND Plocation = 'Hyderabad');
```

Its corresponding relational algebra expression is as follows –

Π_{Name} ($\sigma_{Project \cdot PNO = works\_on.PNO}$

($\sigma_{Plocation = "Hyderabad"}$ (Project X works_on)))

(vi) The SQL, DDL statements are as follows –

```
Create table Employee
(ENO         INT              NOT NULL,
 DOB         DATE,
 ADDRESS     VARCHAR(30),
 SEX         VARCHAR(6),
 SALARY      DECIMAL(10, 2)
 NAME        VARCHAR(30),
 DNO         INT              NOT NULL,
 SUPEREND    INT              NOT NULL,
PRIMARY KEY (ENO));
create table Department
 (DNO        INT          NOT  NULL,
  DNAME      VARCHAR(15)  NOT  NULL,
  MGRENO     CHAR(9)      NOT  NULL,
  MGRSTARTDATE  DATE,
Primary Key (DNO),
Unique (DName),
FOREIGN KEY(MGRENO) References Employee (ENO));
create table dept_location
  (DNO       INT            NOT  NULL,
  Dlocation  VARCHAR(15)    NOT  NULL,
  Primary key (DNO));
create table Project
  (PNAME     VARCHAR(15)   NOT NULL,
   PNO       INT           NOT NULL,
   Plocation VARCHAR(15),
   DNO       INT           NOT NULL,
  Primary key (PNO),
  UNIQUE (PNAME)
FOREIGN KEY (DNO) REFERENCES DEPARTMENT (DNO));
create table Works_ON
  (ENO       CHAR(9)          NOT  NULL
   PNO       INT              NOT  NULL
   HOURS     DECIMAL (3, 1)   NOT  NULL,
Primary key (ENO, PNO),
FOREIGN key (ENO) References Employee (ENO),
FOREIGN key (PNO) References PROJECT(PNO));
```

```
create table Dependent
    (ENO           CHAR(9)           NOT NULL,
     dep_name      VARCHAR(15)       NOT NULL,
     sex           VARCHAR(6),
     Bdate         DATE,
     Relationship  VARCHAR(8),
  PRIMARY KEY (ENO, dep_name),
  FOREIGN KEY (ENO) References Employee (ENO));
```

***Prob.13.** Consider the following relations (primary keys are underlin*

(i) account (acc-no, balance, branch-name)

(ii) depositor (acc-no, cust-no)

(iii) customer (cust-no, name, city)

(iv) loan (loan-no, amt, branch-name)

(v) borrower (cust-no, loan-no)

Solve the following queries using SQL.

(i) Find all customer-no and loan-no who have a loan at t 'Perryridge' branch.

(ii) Find all customer who have an account but no loan at the ba

(iii) Find branch-name and average account balance where averg account balance is greater than 1000.

(R.G.P.V., Nov. 201

Sol. (i) Select distinct customer-name, borrower.loan-number fr borrower, loan where borrower.loan-number = loan.loan-number and branc name = "Perryridge"

(ii) Select customer-name from depositor except select custome name from borrower;

(iii) Select avg (balance).branch-name from account where balan > 1000 group by branch-name;

RELATIONAL ALGEBRA AND RELATIONAL CALCULUS, RELATIONAL ALGEBRA OPERATIONS LIKE SELECT, PROJECT, JOIN, DIVISION, OUTER UNION, TYPES OF RELATIONAL CALCULUS i.e., TUPLE ORIENTED AND DOMAIN ORIENTED RELATIONAL CALCULUS AND ITS OPERATIONS

Q.41. What do you understand by the term relational algebra ? Explai

Or

Describe the relational algebra. ***(R.G.P.V., Dec. 201***

Ans. Relational algebra is a collection of operations to manipulate relatio

The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result. It specifies the operations to be performed on existing relations to derive result relations – Furthermore, it defines the complete scheme for each of the result relations.

The relational algebra operations are usually divided into two groups. One group includes set operations from mathematical set theory; these are applicable because each relation is defined to be a set of tuples. Set operations include union, intersection, set difference and cartesian product. The other group consists of operations developed specifically for relational databases; these include select, project and join, among others.

This concept of relational algebra is given by Codd in 1970. One of the major feature of relational algebra is that it results into a relation, which then can further be used as operand, by next higher query. So, we can have nested queries into a single query to solve the complicated problems. It can have intermediate relations but output is always one relation, there is no place for intermediate relations in the relational database. There are many operations used.

The simplified definitions of these operators are as follows –

(i) ***Restrict/Select*** – It returns a relation containing all tuples from a specified relation that satisfy a specified condition.

(ii) ***Project*** – It returns a relation containing all tuples that remain in a specified relation after specified attributes have been removed.

(iii) ***Product*** – It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations.

(iv) ***Union*** – It returns a relation containing all tuples that appear in either or both of two specified relations.

(v) ***Intersect*** – It returns a relation containing all tuples that appear in both of two specified relations.

(vi) ***Difference*** – It returns a relation containing all tuples that appear in the first and not the second of two specified relations.

(vii) ***Join*** – It returns a relation containing all possible tuples that are a combination of two tuples, one from each of two specified relations, such that the two tuples contributing to any given combination have a common value for the common attributes of the two relations.

(viii) ***Divide*** – It takes two unary relations one binary relation and returns a relation containing all tuples from one unary relation that appear in the binary relation matched with all tuples in the other unary relation.

Q.42. Explain the select operation of relational algebra.

Ans. The ***select*** operation is used to select a subset of the tuples from a relation that satisfy a selection condition (or predicate). It means that select operation can be considered as a filter that keeps only those tuples that sa a qualifying condition. The symbol used for this operation is lowercase G letter sigma (σ) to denote selection. Predicate appears as a subscript to σ example, to select those tuples of the loan relation where the branc "Perryridge", we can write as –

$$\sigma_{\text{branch\_name = "Perryridge"}}\text{(loan)}$$

In general, operators we can use for comparison in the selection pred are =, ≠, <, ≤, >, ≥. We can also combine several predicates into a la predicate using the connectives and (∧) and or (∨), and not (¬). For exam to find those tuples pertaining to loans of more than $1200 made by Perryridge branch, we write

$$\sigma_{\text{branch\_name = "Perryridge"} \wedge \text{amount} > 1200}\text{(loan)}.$$

So, by above discussion, we can say that select operation has follow properties.

Notation : σ_p (R)

Purpose : Pick rows according to some criteria

Input : A table R

Output : Has the same columns as R, but only those rows of l that satisfies predicate P.

Q.43. Define the project operation of relational algebra.

Ans. If we think of a relation as a table, the project operation se certain columns from the table and discards the other columns. The gen form of the project operation is

$$\Pi_{<\text{ attribute list }>}\text{(R)}.$$

where Π (pi) is the uppercase Greek letter to denote the project operation a <attribute list> is a list of attributes from the attributes of relation R. The res of the project operation has only the attributes specified in <attribute list> a in the same order as they appear in the list. Hence, its degree is equal to t number of attributes in <attribute list>.

For example, suppose we want to list all loan numbers and the amount of t loans, the query to list all numbers and the amount of the loan can be written a

$$\Pi_{\text{loan\_number, amount}}\text{(loan)}.$$

So by above explanation, project operation can be given as –

Notation : Π_L(R)

Purpose : Pick the listed columns to output.

Input : A relation R

Output : A relation that has only those columns listed in L.

Q.44. Define the cartesian product operation of relational algebra and give an example. ***(R.G.P.V., June 2009)***

Ans. The cartesian product operation is denoted by a cross (×) symbol. It allows us to combine information from any two relations. We write cartesian product of two relations R_1 and R_2 as $R_1 \times R_2$. The cartesian product of any two relations R_1 (of degree m) and R_2 (of degree n) yields a relation $R_1 \times R_2$ of degree m + n. This product relation has all the attributes that are present in relations R_1 and R_2 and the tuples in $R_1 \times R_2$ are also possible combinations of tuples from R_1 and R_2.

So, if cardinality of R_1 is x and cardinality of R_2 is y, then the cardinality of $R_1 \times R_2$ is xy.

For example, let we want to find the names of all customers who have a loan at the perryridge branch. We need the information in both the *loan* relation and the *borrower* relation to do so, we write

$$\sigma_{\text{branch\_name = "Perryridge"}}\text{(borrower} \times \text{loan)}$$

But the above query executes only when cartesian-product results first, followed by restriction operation.

So, cartesian product can be given as –

Notation : $R_1 \times R_2$

Purpose : Pairs rows from two tables

Input : Two relations R_1 and R_2

Output : For each row in R_1 and each row in R_2, output a row R_1R_2; the output relations has the attribute of R_1 and the attributes of R_2. It is a binary operation.

Q.45. Explain the use of union operator with example.

Or

Define the union in context of SQL. ***(R.G.P.V., Dec. 2006, 2017)***

Ans. The union operator is the usual set union of two relations R and S, which are union-compatible. Two relations are called union compatible, if they satisfy following two conditions –

(i) The relation R and relation S must be of same arity. That is, they must have the same number of attributes.

(ii) The domains of the ith attribute of R and the ith attribute of S must be the same, for all i.

Here, R and S are temporary relations that are the result of relational-algebra expressions.

For example, consider a query to find the names of all bank customers who have either an account or a loan or both. Since, *customer* relation does not have complete information i.e., information about account or loan, bu has only the information of customer i.e. *customer_name*, *customer_street*, *customer_city*.

So, we also need relation *borrower* and *depositor*, that have informa about *loan_number*, *account_number* and *customer_name*.

Query for finding names of all customers who have loan in the bank c be given as –

$$\Pi_{customer\_name} \text{ (borrower)}$$

Similarly, query to find name of all customers who have account in t bank can be given as –

$$\Pi_{customer\_name} \text{ (depositor)}$$

Now, for finding the required result i.e., the customer name who h either an account or a loan at the bank, we have to union above two queri Union operation is denoted by symbol ∪.

Expression needed is given as –

$$\Pi_{customer\_name} \text{ (borrower)} \cup \Pi_{customer\_name}\text{(depositor)}$$

Q.46. Define set intersection operation of relational algebra.

Ans. Set intersection operation returns a relation having all tuples th appear in both of two specified relations. Set-intersection operation is denot by symbol '∩'.

For example, if we wish to find all customers who have both a loan a an account. Using set-intersection, we can write

$$\Pi_{customer\_name} \text{ (borrower)} \cap \Pi_{customer\_name} \text{ (depositor)}$$

Q.47. Define the set difference operation of relational algebra.

(R.G.P.V., Dec. 200

Ans. The set difference operation is denoted by '–'. It allows us to fin tuples that are in one relation but are not in another. The expression R – produces a relation containing those tuples in R but not in S. The set differen operator is a binary operator.

For example, we can find all customers of the bank who have an accou but not a loan by writing

$\Pi_{customer\_name}$ (depositor) – $\Pi_{customer\_name}$ (borrower).

So, by above discussion we have –

Notation	: R – S
Purpose	: To produce single relation removing the commo tuples among R and S.
Input	: Two relations R and S with identical schema
Output	: Relation that has same schema as R and S.

Q.48. What is union compatibility ? Why do the union, intersection and set-difference operations require that the relations on which they are applied are union compatible ? ***(R.G.P.V., Nov./Dec. 2007, Dec. 2013)***

Ans. Several set theoretic operations are used to merge the elements of two sets in various ways, including union, intersection, and set difference. These are binary operations; that is, each is applied to two sets. When these operations are adapted to relational databases, the two relations on which any of the above three operations are applied must have the same type of tuples. This condition is called ***union compatibility***. Two relations R (A_1, A_2,......, A_n) and S (B_1, B_2, ..., B_n) are s7aid to be union compatible if they have the same degree n, and if dom (A_i) = dom (B_i) for $1 \le i \le n$. It means that the two relations have the same number of attributes and that each pair of corresponding attributes have the same domain.

The three operations union, intersection, and set difference can be defined on two union compatible relations R and S as follows –

(i) Union – The result of this operation, denoted by R∪S, is a relation that includes all tuples that are either in R or in S or in both R and S. Duplicate tuples are eliminated.

(ii) Intersection – The result of this operation, denoted by R∩S, is a relation that includes all tuples that are in both R and S.

(iii) Set Difference – The result of this operation, denoted by R – S, is a relation that includes all tuples that are in R but not in S.

Q.49. Explain the rename operation by taking suitable example.

(R.G.P.V., Dec. 2006)

Ans. The rename operation is used to give the name to the results of relational algebra expressions. It is denoted by the lowercase Greek letter rho (ρ). Given a relational algebra expression E, the expression

$$\rho_x \text{ (E)}$$

returns the result of expression E under the name x.

A relation R by itself is a relational algebra expression. The rename operation can also be used to rename a relation R to get the same relation under a new name.

Thus, another form of the rename operation is as follows. Assume that a relational algebra expression E has arity n. Then, the expression

$$\rho_x \text{ (}A_1, A_2, \text{.......}, A_n\text{) (E)}$$

returns the result of expression E under the name x, and with the renamed attributes A_1, A_2, ..., A_n.

For example, we consider a query "find the largest account balance in the bank". For this purpose, we first compute a temporary relation consisting of those balances that are not the largest, and then to take the set differe between the relation $\Pi_{balance}$ (account) and the temporary relation just compu to obtain the result. To compute the temporary relation, we need to comp the values of all account balances. We can do this by computing the carte product account × account and forming a selection to compare the value any two balances appearing in one tuple. First, we need to devise a mechan to distinguish between two balance attributes. We shall use the rename reference to the account relation, thus, we can reference the relation tw without ambiguity. The temporary relation that consists of the balances, t are not the largest, can now be written as –

$$\Pi_{account.balance} (\sigma_{account.balance < d.balance} (\text{account} \times \rho d\ (\text{accou}$$

This expression gives those balances in the account relation for whic larger balance appears, somewhere in the account relation (renamed as The result contains all balances except the largest one.

The query to find the largest account balance in the bank can be wri as follows –

$$\Pi_{balance} (\text{account}) - \Pi_{account.balance}$$
$$(\sigma_{account.balance < d.balance} (\text{account} \times \rho d\ (\text{account})))$$

So, by above discussion we can say that, for a rename operation

Notation : $\rho_s(R)$ or $\rho_{(A_1, A_2 ... A_m)}(R)$
Purpose : Rename a relation and/or this columns
Input : Relation R
Output : Relation S, with tuples of R under same or differ named attributes.

Q.50. Explain the basic relational algebra operations with the symb used and example for each. ***(R.G.P.V., June 20***

Or

Discuss the different relational algebra operations. ***(R.G.P.V., Nov. 201***

Ans. In the list of fundamental operations, select, project and rename a called the unary operations because they operate on one relation. The oth three operations union, set difference and cartesian product operate on th pair of relations and are, therefore, called binary operations.

(i) ***The Select Operation*** – Refer to Q.42.
(ii) ***The Project Operation*** – Refer to Q.43.
(iii) ***The Union Operation*** – Refer to Q.45.
(iv) ***The Set Difference Operation*** – Refer to Q.47.
(v) ***The Cartesian Product Operation*** – Refer to Q.44.
(vi) ***The Rename Operation*** – Refer to Q.49.

Q.51. What do you mean by query language ? Explain following operations by taking suitable examples.
(i) Select (ii) Project (iii) Cartesian product (iv) Rename (v) Union
Also explain super key and candidate key. ***(R.G.P.V., Dec. 2012)***

Ans. **Query Language** – Refer to Q.25.

(i) ***Select*** – Refer to Q.42.
(ii) ***Project*** – Refer to Q.43.
(iii) ***Cartesian Product*** – Refer to Q.44.
(iv) ***Rename*** – Refer to Q.49.
(v) ***Union*** – Refer to Q.45.

Super Key – Refer to Q.12 (iii).
Candidate Key – Refer to Q.12 (i).

Q.52. Explain the assignment operation of relational algebra.

Ans. The assignment operation is denoted by ← sign. It works like assignment in a programming language. The evaluation of an assignment does not result in any relation being displayed to the user. The result of the expression to the right of the ← is assigned to the relation variable on the left of the ←. This relation variable may be used in subsequent expressions.

With the assignment operation, a query can be written as a sequential program consisting of a series of assignment followed by an expression whose value is displayed as the result of query. For relational algebra queries, assignment must be made to a temporary relation variable. Assignments to permanent relations constitute a database modification.

For example, if bank manager wants to know the customer_name who have account, then the relational expression would be –

$$\text{result} \leftarrow \Pi_{customer\_name} (\text{depositor})$$

The degree and cardinality of the resultant relation will depend on the expression on RHS.

Q.53. Define the natural join operation of relational algebra and give an example. ***(R.G.P.V., June 2008)***

Or

Define the natural join in context of SQL. ***(R.G.P.V., Dec. 2017)***

Ans. The natural join is a binary operation that allows us to combine certain selections and a cartesian product into one operation. It is denoted by the join symbol ⋈. The natural join operation forms a cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes.

Natural join is not as difficult as it looks. For example, "to find the names of all customers who have a loan at the bank and find the amount of the loan."

We can express this query by following expression –

$$\Pi_{customer\_name,\ loan\_number,\ amount}\ (borrower \bowtie loan)$$

Since, the schemas for *borrower* and *loan* have the attribute *loan_num* in common, the natural-join operation considers only pairs of tuples that h same value on *loan_number*. It combines each such pair of tuples into a si tuple on the union of the two schemas i.e., customer_name, branch_na loan_number, amount. After performing the projection, the resultant rela can contains customer_name, loan_number and amount.

Now, the natural join can be formally defined as follows – If r (R) a (S) are two relations, then the natural join of r and s, denoted by r⋈s, i relation on schema R∪S defined as follows –

$$r \bowtie s = \pi_{R \cup S}(\sigma_{r.A_1} = s.A_1 \wedge r.A_2 = s.A_2 \wedge ....$$
$$\wedge r.A_n = s.A_n r \times s)$$

where $R \cap S = \{A_1, A_2, ..., A_n\}$.

Q.54. Explain JOIN. Differentiate between join and cartesian prod Give various join types and condition. ***(R.G.P.V., June 20***

Or

Explain various types of join with example. Also, differentiate betw natural join and equi join. ***(R.G.P.V., Dec. 20***

Or

Discuss join operation and its types with example. (R.G.P.V., Dec. 20

Ans. The JOIN operation, denoted by ⋈, is used to combine rela tuples from two relations into single tuples. To illustrate join, suppose that want to retrieve the name of the manager of each department. To get manager's name, we need to combine each department tuple with the employ tuple whose SSN value matches the MGRSSN value in the department tup This is done by using the join operation, and projecting the result over t necessary attributes as follows –

$$DEPT\_MGR \leftarrow DEPARTMENT \bowtie_{MGRSSN = SSN} EMPLOYEE$$

$$RESULT \leftarrow \Pi_{DNAME,\ LNAME,\ FNAME}\ (DEPT\_MGR)$$

The first operation is shown in fig. 2.15.

DEPT_MGR	DNAME	DNUMBER	MGRSSN	•••	FNAME	MINIT	LNAME	SSN	•
	Research	5	333445555	•••	Franklin	T	Wong	333445555	•
	Administration	4	987654321	•••	Jennifer	S	Wallace	987654321	•
	Headquarters	1	888665555	•••	James	E	Borg	888665555	•

Fig. 2.15 Illustrating the JOIN Operation

The general form of JOIN operation on two relations $R(A_1, A_2, ..., A_n)$ and $S(B_1, B_2, ..., B_m)$ is –

$$R \bowtie_{<join\ operation>} S$$

The result of the JOIN is a relation Q with n + m attributes $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$ in that order; Q has one tuple for each combination of tuples – one from R and one from S – whenever the combination satisfies the join condition. This is the main difference between CARTESIAN PRODUCT and JOIN – in JOIN, only combinations of tuples satisfying the join operation appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result. The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples. Each tuple combination for which the join condition evaluates to true is included in the resulting relation Q as a single combined tuple.

A general join condition is of the form

<condition>AND<condition>AND....AND<condition>

where each condition is of the form $A_i\ \theta\ B_j$, A_i is an attribute of R, B_j is an attribute of S, A_i and B_j have the same domain, and θ(theta) is one of the comparison operators $\{=, <, \leq, >, \geq, \neq\}$. A JOIN operation with such a general join condition is called a **THETA JOIN.** Tuples whose join attributes are null do not appear in the result.

The most common JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUIJOIN. Where only comparison operator used is not equal to is called an NON-EQUIJOIN. Note that in the result of an EQUIJOIN there have one or more pairs of attributes that have identical values in every tuple of DEPT_MGR because of the equality join condition specified on these two attributes. Because one of each pair of attributes with identical values is superfluous, a new operation called NATURAL JOIN, denoted by *, was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition. The NATURAL JOIN requires that the two join attributes have the same name in both relations. If this is not the case, a renaming operation is applied first.

Q.55. Discuss the selection, projection and join operator of relational algebra with a suitable example. ***(R.G.P.V., Dec. 2011)***

Ans. **Selection Operator** – Refer to Q.42.

Projection Operator – Refer to Q.43.

Join Operator – Refer to Q.54.

Q.56. Explain theta join with an example.

Or

What are the various types of inner join operators ? Why theta joi required ? ***(R.G.P.V., June 2006, Dec. 2(***

Ans. The theta join operation is an extension to the natural-join opera This operation is intended to join two relations together on the basis of s comparison operator other than equality. Let relations A and B satisfy requirements for cartesian product i.e., they have no attribute names in comm Let A have an attribute X and let B have an attribute Y, and let X, Y, a satisfy the requirements for restriction. Then, the θ join of relation A on attri X with relation B on attribute Y is defined to be the result of evaluating expression.

(A TIMES B) WHERE X θ Y

In other words, it is a relation with the same heading as the cartes product of A and B, and with a body consisting of the set of all tuples t s that t appears in that cartesian product and the condition "X θ Y" evaluate true for that tuple t.

If θ is ">", the θ-join is called the ***greater than join.*** For example, supp we wish to compute the greater-than join of relation S on city with P on city shown in fig. 2.17 then the relational expression is as follows –

((S RENAME CITY AS SCITY) TIMES

(P RENAME CITY AS PCITY))

WHERE SCITY > PCITY

The result of this expression as shown in fig. 2.16.

S#	SNAME	STATUS	SCITY	P#	PNAME	COLOR	WEIGHT	PCIT
S_2	Jones	10	Paris	P_1	Nut	Red	12.0	Lond
S_2	Jones	10	Paris	P_4	Screw	Red	14.0	Lond
S_2	Jones	10	Paris	P_6	Cog	Red	19.0	Lond
S_3	Blake	30	Paris	P_1	Nut	Red	12.0	Lond
S_3	Blake	30	Paris	P_4	Screw	Red	14.0	Lond
S_3	Blake	30	Paris	P_6	Cog	Red	19.0	Lond

Fig. 2.16 Greater-than Join of Suppliers and Parts on Cities

If θ is " = ", the θ-join is called an ***equijoins.*** It follows from the defini that the result of an equijoin must include two attributes with the property the values of those two attributes are equal in every tuple in the relation. If of those two attributes is projected away and the other renamed appropria the result is the natural join.

S

S#	SNAME	STATUS	CITY
S_1	Smith	20	London
S_2	Jones	10	Paris
S_3	Blake	30	Paris
S_4	Clark	20	London
S_5	Adams	30	Athens

P

P#	PNAME	COLOR	WEIGHT	CITY
P_1	Nut	Red	12.0	London
P_2	Bolt	Green	17.0	Paris
P_3	Screw	Blue	17.0	Rome
P_4	Screw	Red	14.0	London
P_5	Cam	Blue	12.0	Paris
P_6	Cog	Red	19.0	London

Fig. 2.17 The Suppliers and Parts Database

Similarly, if θ is "<", the θ-join is called the ***less than join.*** For example, to compute the less than join of relations S on city with P on city as shown in fig. 2.17, the relational expression is as follows –

((S RENAME CITY AS SCITY) TIMES

(P RENAME CITY AS PCITY))

WHERE SCITY < PCITY

The result of this expression is shown in fig. 2.18.

S#	SNAME	STATUS	SCITY	P#	PNAME	COLOR	WEIGHT	PCITY
S_1	Smith	20	London	P_2	Bolt	Green	17.0	Paris
S_1	Smith	20	London	P_3	Screw	Blue	17.0	Rome
S_1	Smith	20	London	P_5	Cam	Blue	12.0	Paris
S_2	Jones	10	Paris	P_3	Screw	Blue	17.0	Rome
S_3	Blake	30	Paris	P_3	Screw	Blue	17.0	Rome
S_4	Clark	20	Paris	P_3	Screw	Blue	17.0	Rome
S_5	Adams	30	London	P_2	Bolt	Green	17.0	Paris
S_5	Adams	30	London	P_3	Screw	Blue	17.0	Rome
S_5	Adams	30	London	P_5	Cam	Blue	12.0	Paris

Fig. 2.18 Less-than Join of Suppliers and Parts on Cities

Q.57. Explain the use of division relational operator with example. ***(R.G.P.V., Dec. 2006, 2008)***

Or

Explain division operation of relational algebra. How division operator can be represented using traditional set operators ? ***(R.G.P.V., Dec. 2009)***

Or

Define the division operation of relational algebra and give an example of each. ***(R.G.P.V., June 2008, 2009)***

Ans. The division operation is useful for a special kind of queries include the phrase "for all". It is denoted by ÷.

It takes two unary relations and one binary relation and returns a rela containing all tuples from one unary relation that appear in the binary rela matched with all tuples in the other unary relation.

For example, suppose we wish to find all customers who have an acco at all the branches located in Brooklyn. We can obtain all the branches loc in Brooklyn by the expression –

$$R_1 = \Pi_{branch\_name}(\sigma_{branch\_city = \text{"Brooklyn"}}(branch))$$

Now, the second step is to find all (customer_name, branch_name) p for which the customers have an account at a branch by writing –

$$R_2 = \Pi_{customer\_name,\ branch\_name}(depositor \bowtie account)$$

Now, at last step we need to find customers who appear in R_2 with ev branch name in R_1. The operation that provides exactly those customers is divide operation. We formulate the query by writing –

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_c\ldots \text{"Brooklyn"}}(branch))$$

The result of this expression is a relation that has the sche (customer_name) and that contains the tuple Jay.

Q.58. Explain select, project, join and division with example.
(R.G.P.V., June 20

Ans. Refer to Q.42, Q.43, Q.53 and Q.57.

Q.59. List the operations of relational algebra and the purpose of e
(R.G.P.V., Dec. 20

Or

Explain various traditional or fundamental operators of relatio algebra. Explain division operation and rewrite this operator using traditio operators.
(R.G.P.V., June 200

Ans. Refer to Q.42, Q.43, Q.44, Q.45, Q.46, Q.47, Q.49, Q.52, Q and Q.57.

Q.60. Discuss the outer join operation of extended relational algeb

Ans. The outer join operation is an extension of the join operation to d with missing information. Suppose that there are two relations with following schemas containing data on full-time employees –

employee (emloyee_name, street, city)
ft_works (employee_name, branch_name, salary)

Fig. 2.19 shows two relations employee and ft_works.

employee_name	street	city
Coyote Rabbit Smith William	Toon Tunnel Revolver	Hollywood Carrotville Death Valley Seattle

employee_name	branch_name	salary
Coyote Rabbit Gates William	Mesa Mesa Redmond Redmond	1500 1300 5300 1500

Fig. 2.19 The employee and ft_works Relations

If we want to generate a single relation with all the information about full_time employee. The approach is to use the natural join operation as follows –

$$employee \bowtie ft\_works$$

The result of this operation is shown in fig. 2.20.

employee_name	street	city	branch_name	salary
Coyote Rabbit William	Toon Tunnel Seaview	Hollywood Carrotville Seattle	Mesa Mesa Redmond	1500 1300 1500

Fig. 2.20 The Result of employee ⋈ ft-works

Here, we have lost the street and city information about smith, since the tuple describing smith is absent from the ft_works relation.

Similarly, the branch_name and salary information about Gates have been lost, since the tuple describing Gates is absent from the employee relation.

This loss of information can be avoided using outer join operation. There are three forms of outer join operation – left outer join, right outer join, and full outer join. All three forms of outer join compute the join, and add extra tuples to the result of the join.

(i) ***Left Outer Join*** – It is denoted by ⟕ symbol. It takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join.

Fig. 2.21 shows the result of the expression employee ⟕ ft_works.

employee_name	street	city	branch_name	salary
Coyote Rabbit William Smith	Toon Tunnel Seaview Revolver	Hollywood Carrotville Seattle Death Valley	Mesa Mesa Redmond Null	1500 1300 1500 Null

Fig. 2.21 Result of employee ⟕ ft_works

(ii) ***Right Outer Join*** – It is denoted by ⟖ symbol. It pads tuples from the right relation that did not match from the left relation with null values and adds them to the result of the natural join.

Fig. 2.22 shows the result of the expression employee ⟕ ft_works

employee_name	street	city	branch_name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Gates	Null	Null	Redmond	5300

Fig. 2.22 Result of employee ⟕ ft_works

(iii) Full Outer Join – The full outer join is denoted by ⟗ sym performs functions of both left and right outer joins. This means that it p tuples from the left relation that did not match any from the right relation well as tuples from the right relation that did not match any from the l relation, and adds them to the result of the join.

Fig. 2.23 shows the result of expression employee ⟗ ft_works.

employee_name	street	city	branch_name	salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrotville	Mesa	1300
William	Seaview	Seattle	Redmond	1500
Smith	Revolver	Death Valley	Null	Null
Gates	Null	Null	Redmond	5300

Fig. 2.23 Result of employee ⟗ ft_works

Q.61. Explain natural join, outer join, full outer join, left outer j and theta join with examples. ***(R.G.P.V., Dec. 20***

Or

Explain the join operator, its relevance and its various types. ***(R.G.P.V., Dec. 201***

Or

Discuss the various type of join operations. Why are these join required ***(R.G.P.V., Dec. 201***

Ans. Refer to Q.53, Q.60 and Q.56.

Q.62. Explain view in relational algebra. ***(R.G.P.V., June 200***

Or

Define views. ***(R.G.P.V., Dec. 201***

Ans. Any relation that is not part of the logical model, but is made visib to a user as a virtual relation, is called a ***view***. It is possible to support a lar number of views on top of any given set of actual relations.

We define a view by using the create view statement. To define a vie we must give the view a name and must state the query that computes t view. The form of the create view statement is

create view v as <query expression>

where <query expression> is any legal relational-algebra query expression. The view name is represented by v.

Q.63. What is a view ? Create a view of EMP table named DEPT 20, to show the employees in department 20 and their annual salary. ***(R.G.P.V., Dec. 2010)***

Ans. View – Refer to Q.62.

The given view is created as follows –

```
create view DEPT 20 as
select employee, annual_salary
from EMP
where employee_dept = 20
```

Q.64. What is the difference between view and table ? ***(R.G.P.V., June 2016)***

Ans. View is a virtual table based on the result set of an SQL statement. A view contains rows and columns just like a real table. The fields in a view are fields from one or more real tables in the database. In views, rows are not explicitly stored in the database but are computed as needed from a view definition. A table is a two dimensional view of the database where you store your data. It is a set of related pieces of information stored in rows and columns. The difference between a view and a table is that views are definitions built on top of other tables (or views), and do not hold data themselves. If data is changing in the underlying table, the same change is reflected in the view.

Q.65. Describe the relational calculus. (R.G.P.V., June 2005, Dec. 2016)

Ans. Relational calculus is a formal query language where we write one ***declarative*** expression to specify a retrieval request and hence there is no description of how to evaluate a query; a calculus expression specifies what is to be retrieved rather than how to retrieve it. Therefore, the relational calculus is considered to be a ***nonprocedural*** language. This differs from relational algebra, where we must write a sequence of operations to specify a retrieval request. Hence, it is considered as a ***procedural*** way of stating a query. It is possible to rest algebra operations to form a single expression. However, a certain order among the operations is always explicitly specified in a relational algebra expression. This order also influences the strategy for evaluating the query.

Any retrieval that can be specified in the algebra can also be specified in the relational calculus and vice versa. In other words, the expressive power of the two languages is identical.

The fact is, the algebra and the calculus are logically equivalent. For every algebraic expression there is an equivalent calculus one, for every calculus expression there is an equivalent algebraic one. There is a one-to-one correspondence between the two. Thus, the difference between them is really just a difference of style – the calculus is arguably closer to natural langua the algebra is like a programming language.

Relational calculus is based on a branch of mathematical logic cal *predicate* calculus. The idea of using predicate calculus as the basis fo query language was originated by Kuhns. The concept of a relational calcu – i.e., an applied predicate specifically tailored to relational databases – w first proposed by Codd. A language explicitly based on that calculus cal data sublanguage ALPHA was presented by Codd. ALPHA was ne implemented, but a language called QUEL was implemented and had bee competitor to SQL.

Q.66. Differentiate between relational calculus and relational algebr
(R.G.P.V., Dec. 201

Ans. Refer to Q.65.

Q.67. What do you mean by the tuple relational calculus ? What are operations ?

Or

Write short note on tuple calculus. ***(R.G.P.V., June 20***

Or

Explain with respect to tuple calculus –
(i) tuple variable (ii) range relation (iii) atom formula (iv) expressi (v) ∃ (Existential quantifier) and ∀ (universal quantifier).
(R.G.P.V., Nov./Dec. 20

Ans. The tuple relational calculus is a nonprocedural query language describes the desired information without giving a specific procedure f obtaining that information. It is based on specifying a number of *tup variables*. Each tuple variable usually ***ranges over*** a particular datab relation, meaning that the variable may take as its value any individual tup from that relation.

A query in the tuple relational calculus is expressed as

{t | COND (t)}

where t is a tuple variable and COND(t) is a conditional expression involving The result of such a query is the set of all tuples t that satisfy COND(t). F example, to find the employee whose salary is above $ 50,000, the tuple calcu expression is as follows –

{t | employee (t) and t.salary > 50000}

The condition employee (t) specifies that the range relation of tuple variab *t is employee*. Each employee tuple t that satisfies the condition t.salary 5000 is retrieved.

If we want to select only some of the attributes – say the first and last names, then the query is written as follows –

{t.fname, t.lname | employee (t) and t.salary > 50000}

Informally, the following information is specified in a tuple calculus expression –

(i) For each tuple variable t, the range relation R of t. This value is specified by a condition of the form R(t).

(ii) A condition to select particular combinations of tuples. As tuple variables range over their respective range relations, the condition is evaluated for every possible combination of tuples to identify the selected combinations for which the condition evaluates to true.

(iii) A set of attributes to be retrieved i.e., the requested attributes. The values of these attributes are retrieved for each selected combination of tuples.

Expressions and Formulas in Tuple Relational Calculus – A general *expression* of the tuple relational calculus is of the form

$$\{t_1.A_1, t_2.A_2, ..., t_n.A_n \mid \text{COND } (t_1, t_2, ..., t_n, t_{n+1}, t_{n+2}, ..., t_{n+m})\}$$

where $t_1, t_2, ..., t_n, t_{n+1}, ..., t_{n+m}$ are tuple variables, each A_i is an attribute of the relation on which t_i ranges, and COND is a ***condition*** or ***formula*** of the tuple relational calculus. A formula consists of predicate calculus atoms, which can be one of the following –

(i) An atom of the form $R(t_i)$, where R is a relation name and t_i is a tuple variable. This atom identifies the range of the tuple variable t_i as the relation whose name is R.

(ii) An atom of the form $t_i.A$ *op* $t_j.B$, where op is one of the comparison operators in the set $\{=, >, \geq, <, \leq, \neq\}$, t_i and t_j are tuple variables, A is an attribute of the relation on which t_i ranges, and B is an attribute of the relation on which t_j ranges..

(iii) An atom of the form $t_i.A$ op c or c op $t_j.B$, where op is one of the comparisons operators in the set $\{=, >, \geq, <, \leq, \neq\}$, t_i and t_j are tuple variables, A is an attribute of the relation on which t_i ranges, B is an attribute of the relation on which t_j ranges and c is a constant value.

Each of the atoms evaluates to either true or false for a specific combination of tuples. This called the ***truth value*** of an atom. For atoms of type1, if the tuple variable is assigned a tuple that is a member of the specified relation R, the atom is true; otherwise it is false. For atoms of type 2 and 3, if the tuple variables are assigned to tuples such that t values of the specified attributes of the tuples satisfy the condition, th the atom is true.

A ***formula*** or ***condition*** consists of one or more atoms connecte through the logical operators ***and***, ***or***, and ***not***, is defined recursively a follows –

(i) Every atom is a formula

(ii) If F_1 and F_2 are formulas, then so are (F_1 and F_2), (F_1 or F_2), not (F_1) and not (F_2). The truth values of these four formulas are derive from their component formulas F_1 and F_2 as follows –

(a) (F_1 and F_2) is true if both F_1 and F_2 are true, otherwise fals

(b) (F_1 or F_2) is false if both F_1 and F_2 are false; otherwise tru

(c) not (F_1) is true if F_1 is false, it is false if F_1 is true.

(d) not (F_2) is true if F_2 is false, it is false if F_2 is true.

The Existential (∃) and Universal (∀) Quantifiers – Two speci symbols called quantifiers can appear in formulas; these are the univers quantifier (∀) and the existential quantifier (∃).

A tuple variable t is ***bound*** if it is quantified. It means that it appears in a (∃t) or (∀t) clause; otherwise, it is ***free***. Formally, a tuple variable can b defined in a formula as free or bound according to the following rules –

(i) An occurrence of a tuple variable in a formula F that is an atom is free in F.

(ii) An occurrence of a tuple variable t is free or bound in a formul made up logical connectives – (F_1 and F_2), (F_1 or F_2), not (F_1), and not (F_2) – depending on whether it is free or bound in F_1 or F_2.

(iii) All free occurrences of a tuple variable t in F are bound in formul F' of the form F' = (∃t) (F) or F' = (∀t)F. The tuple variable is bound to the quantifier specified in F'.

Now, the rules for the definition of a formula are as follows –

(i) If F is a formula, the so is (∃t) (F), where t is tuple variable. The formula (∃t) (F) is true if the formula F evaluates to true for at least one tuple assigned to free occurrences of t in F; otherwise (∃t) (F) is false.

(ii) If F is a formula, then so is (∀t) (F), where t is a tuple variable. The formula (∀t) (F) is true if the formula F evaluates to true for every tuple assigned to free occurrences of t in F; otherwise (∀t) (F) is false.

The (∃) quantifier is called an existential quantifier because a formula (∃t) (F) is true if there exists some tuple that makes F true. For the universal quantifier, (∀t) (F) is true if every possible tuple that can be assigned to free occurrences of t in F is substituted for t, and F is true for every such substitution. It is called the universal (or for all) quantifier because every tuple in the universe of tuples must make F true to make the quantified formula true.

Q.68. What do you mean by domain relational calculus ? Explain its operations in brief.

Ans. Another type of relational calculus is called the domain relational calculus or simply domain calculus. The key difference between tuple calculus and domain calculus is that the range variables of the tuple calculus range over relations, while the range variables of the domain calculus range over domains.

An expression in the domain relational calculus is of the form

$$\{<x_1, x_2, \ldots\ldots\ldots, x_n> \mid P\ (x_1, x_2, \ldots\ldots, x_n)\}$$

where $x_1, x_2, \ldots\ldots, x_n$ are domain variables that range over domains (of attributes) and P is a condition or formula composed of atoms. An atom in the domain relational calculus can be one of the following forms –

(i) $<x_1, x_2, \ldots\ldots, x_n> \in r$, where r is a relation on n attributes and $x_1, x_2, \ldots\ldots, x_n$ are domain variables or domain constants.

(ii) $x\ \theta\ y$, where x and y are domain variables and θ is comparison operator in the set $\{=, >, \geq, <, \leq, \neq\}$.

(iii) $x\ \theta\ c$, where x is a domain variable θ is comparison operator in the set $\{=, >, \geq, <, \leq, \neq\}$ and c is a constant in the domain of the attribute for which x is a domain variable.

As in tuple calculus, atoms evaluate to either true or false for a specific set of values, called the truth values of the atoms. In case 1, if the domain variables are assigned values corresponding to a tuple of the specified relation r, then the atom is true. In cases 2 and 3, if the domain variables are assigned values that satisfy the condition, then the atom is true.

We can build up formulae from atoms by using the following rules –

(i) An atom is a formula.

(ii) If P_1 is a formula, then so are $\neg P_1$ and (P_1).

(iii) If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$.

(iv) If P_1 is a formula in x, where x is a domain variable, then

$$\exists x\ (P_1\ (x)) \text{ and } \forall x\ (P_1\ (x))$$

are also formulae.

Q.69. Differentiate between the tuple and domain calculus.

(R.G.P.V., Dec. 200

Or

Differentiate between tuple oriented and domain oriented relation calculus.

(R.G.P.V., Dec. 200

Or

How does tuple-oriented relational calculus differ from domain-orient relational calculus ?

(R.G.P.V., May 201

Ans. Refer to Q.68.

NUMERICAL PROBLEMS

Prob.14. Specify the following query in relational algebra –

Supplier (sid, sname, address)

Part (pid, pname, color)

Catalog (sid, pid, cost)

(i) Find the name of suppliers who supply some red or green par

(ii) Find the sids of suppliers who supply every part.

(iii) Find the sids of suppliers who supply red and green part.

(R.G.P.V. Dec. 201

Sol. (i) (((Part WHERE COLOR = COLOR ('Red' ∨ 'Green'))
JOIN Catalog) {sid} JOIN Supplier) {sname}

(ii) ((Supplier {sid} DIVIDEBY Part {pid} PER
Catalog {sid, pid}) JOIN Supplier) {sid}

(iii) (((Part WHERE COLOR = COLOR ('Red' ∧ 'Green')) JOI
Catalog) {sid} JOIN Supplier) {sid}

Prob.15. Consider the following relational database –

Employee (EMP_Name, Street, City)

Works (EMP_Name, Company_Name, Salary)

Company (Company_Name, City)

Manages (EMP_Name, Manager_Name)

For each of the following queries, give an expression in the relationa algebra. Tuple relational calculus –

(i) Find the names of all employees who work for Satyam.

(ii) Find the names and cities of residence who work for Satyam

(iii) Find the names of all employees who live in the same city the company for which they work.

(iv) Find the names of all employees who do not work for Satyam.

(v) Find the names of all employees who live in the same city and on the same street as do their managers.

(R.G.P.V., June 2010)

Sol. (i) $\Pi_{\text{EMP\_Name}}(\sigma_{\text{Company\_Name = 'Satyam'}}(\text{Works}))$

(ii) $\Pi_{\text{EMP\_Name, City}}(\text{Employee} \bowtie \pi_{\text{City}}$
$(\sigma_{\text{Company\_Name='Satyam'}}(\text{Company}))$

(iii) $\Pi_{\text{EMP\_Name}}(\sigma_{\text{Employee.City=Company.City}}(\text{Employee} \bowtie$
$(\text{Works} \bowtie \text{Company}))$

(iv) $\Pi_{\text{EMP\_Name}}(\text{Works}) - \pi_{\text{EMP\_Name}}(\sigma_{\text{Company\_Name='Satyam'}}(\text{Works}))$

(v) $\Pi_{\text{Employee.EMP\_Name}}(\sigma_{\text{Employee.Street\textbackslash Managers.Street}}$
$_{\wedge\ \text{Employee.City=Managers.City}}(\text{Employee} \times \rho_{\text{Managers(Street, City)}}$
$(\Pi_{\text{Street, City}}(\sigma_{\text{Employee.EMP\_Name=Managers.Manager\_Name}}$
$(\text{Employee} \times \text{Managers})))))$

Prob.16. Consider the relations –

PROJECT(proj#, proj_name)

EMPLOYEE(emp#, emp_name)

ASSIGNED(proj#, emp#)

Use relational algebra to express the following queries –

(i) Find the employee number of employees who work on at least all of the projects that employee 107 works on.

(ii) Get the employee number of employees who work on all projects.

(R.G.P.V., Dec. 2010)

Sol. (i) $\text{temp1} \leftarrow \sigma_{\text{emp\# = 107}}$ (ASSIGNED)
$\text{temp2} \leftarrow \Pi_{\text{emp\#, proj\#}}$ (EMPLOYEE $\bowtie$ ASSIGNED)
Result $\leftarrow \Pi_{\text{emp\#}}$ (temp1 $\cup$ temp2)

(ii) $\Pi_{\text{emp\#}}$(EMPLOYEE $\bowtie$ ASSIGNED)

Prob.17. Let R(A, B) and S(A, C) be two relations. Give relational algebra expressions for the following domain calculus expressions –

(i) $\{< a > \mid \exists b(< a, b > \in r \wedge b = 17)\}$
(ii) $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$
(iii) $\{< a > \mid \exists b(\exists c(< a, b > \in r) \wedge < a, c > \in s)\}\}$

(R.G.P.V., Dec. 2008, 2010)

Sol. (i) The relational algebra expression corresponds to given domain calculus expression is as follows –

$$\Pi_a(\sigma_{b = 17}(r))$$

(ii) The relational algebra expression corresponds to given domain calculus expression is as follows –

$$\Pi_{a, b, c}(\sigma_{a.r = a.s}(r \times s))$$

(iii) The relational algebra expression corresponds to given domain calculus expression is as follows –

$$\Pi_a(r \bowtie s)$$

Prob.18. Consider the following tables –

Employee (Emp_no, Name, Emp_city)

Company (Emp_no, Company_name, Salary)

(i) Write a SQL query to display employee name and company name.

(ii) Write a SQL query to display employee name, employee city, company name and salary of all the employees whose salary > 10000.

(iii) Write a query to display all the employees working in "XYZ" company.

(R.G.P.V., Dec. 2017)

Sol. The SQL queries are as follows –

(i) Select E.name, C.Company_name from Employee E inner join Company C ON E.Emp_no = C.Emp_no.

(ii) Select E.name, E.Emp_city, C.Company_name, C.Salary from Employee E inner join Company C ON E. Emp_no = C. Emp_no where C.salary > 10000.

(iii) Select * from Employee where Emp_no. IN (select Emp_no from Company where Company_name = "XYZ")

Prob.19. Consider the following relational schema –

Employee (empno, name, office, age)

Books (isbn, title, authors, publisher)

Loan (empno, isbn, date)

Write the following queries in relational algebra.

(i) Find the names of employees who have borrowed a book published by McGraw-Hill.

(ii) Find the names of employees who have borrowed all books published by McGraw-Hill.

(iii) Find the names of employees who have borrowed more than five different books published by McGraw-Hill.

(R.G.P.V., Dec. 2017)

Sol. (i) Select E.name from Employee E inner join Loan L ON E.empno = L.empno inner join Book B ON L.isbn = B.isbn where B.publisher = "McGraw -Hill AND count (B.isbn) = = 1.

(ii) Select E.name from Employee E inner join Loan L ON E.empno = L.empno inner join Book B ON L.isbn = B.isbn where B.publisher = "McGraw-Hill".

(iii) Select E.name from Employee E inner join Loan L ON E.empno = L.empno inner join Book B ON L.isbn = B.isbn where B.publisher = "McGraw-Hill" AND count (B.isbn) > 5.

Prob.20. Consider the following relations –

Person (name, street, city)

Owns (name, reg_no, model, year)

Accident (data, reg_no)

Answer the following using tuple relational calculus –

(i) Find the names of persons who are not involved in any accident.

(ii) Find the names and street of persons who own a maruti car.

(iii) Find the registration numbers of the cars manufactured in the year 2004.

(R.G.P.V., June 2009)

Sol. (i) The query is as follows –

$\{t \mid \exists s \in \text{owns } (t[\text{name}] = s[\text{name}]) \wedge \neg \exists\, r \in \text{accident } (t[\text{reg\_no}] = r[\text{reg\_no}])\}$

(ii) The query is as follows –

$\{t \mid \exists s \in \text{owns } \exists\, r \in \text{person } (s[\text{name}] = r[\text{name}]) \wedge s[\text{model}] = \text{'maruti'} \wedge t[\text{name}] = r[\text{name}] \wedge t[\text{street}] = r[\text{street}])\}$

(iii) The query is expressed as follows –

$\{t \mid \exists s \in \text{owns } (t[\text{reg\_no}] = s[\text{reg\_no}] \wedge s[\text{model}] = 2004)\}$

Prob.21. Consider the relation schema. Find out the relational algebra of the following queries –

Employee (employee name, street, city),

Work (employee name, company_name, salary)

Company (company name, city),

Manages (employee name, manager_name)

(i) Find the names and cities of residence of employees working for TCS.

(ii) Find the names, street and cities of residence of employees working for infosys and earning more than 20,000.

(iii) Find the name of employees working in the same city where they live.

(iv) Find the name of employees who are not working for WIPRO.

(v) Find the total and average salary paid by each company.

(R.G.P.V., May 2018)

Ans. (i) $\Pi_{\text{E-NAME, CITY}}$ ($\sigma_{\text{COMPANY\_NAME = "TCS"}}$ (works_for*employee)

(ii) $\Pi_{\text{E-NAME, STREET, CITY}}$ ($\sigma_{\text{COMPANY\_NAME = "Infosys"} \wedge \text{SALARY > 200}}$ (works_for*employee))

(iii) $\Pi_{\text{E-NAME, CITY}}$ (employee* works-for * company)

(iv) $\Pi_{\text{E-NAME}}$ (employee) – $\Pi_{\text{E-NAME}}$ ($\sigma_{\text{COMPANY\_NAME = "WIPRO"}}$ (works-for))

(v) Company-Name $\mathcal{G}_{\text{MAX (SALARY), AVG (SALARY)}}$ (works-for)

UNIT 3

DATABASE DESIGN – INTRODUCTION TO NORMALIZATION, NORMAL FORMS, FUNCTIONAL DEPENDENCY

Q.1. What is normalization ? Justify the need for normalization with examples. ***(R.G.P.V., Dec. 2015)***

Or

What do you mean by normalization ? ***(R.G.P.V., Dec. 2016)***

Or

Write short note on normalization. ***(R.G.P.V., Nov. 2018)***

Ans. Normalization is a process of converting a relation into a standard form. The normalization process or just normalization is built around the concept of normal forms. A relation is said to be in a particular form if it satisfies a certain set of conditions. For example, let we have a relation suppliers with attributes name, address, item, price.

suppliers (name, address, item, price)

Now, the problems that we can face due to this relation are –

(i) Redundancy – The address of supplier is repeated once for each item supplied. So this causes repetition of same information many times.

(ii) Potential Inconsistency (Update Anomalies) – As we have redundancy in our relation i.e., same information is repeated many times. It is possible that when we are updating some information, some get updated while others remain fixed, which will result inconsistency.

As in above example, if we update the address of a supplier in one tuple, while leaving it fixed in another. Thus, we would not have a unique address for each supplier.

(iii) Insertion Anomalies – We cannot insert a tuple in a relation, if that tuple does not have value for any of the attribute in relation.

For example, we cannot record an address for a supplier, if that supplier does not currently supply at least one item. We might put null values in the item and price components of a tuple for that supplier but then, when enter an item for that supplier, we will remember to delete the tuple with the nulls.

(iv) Deletion Anomalies – On deleting some information, we lo some other information too. For example, if we delete all items supplied b one suppliers, we unintentionally lose track of its address.

So, due to the all of the above problems we normalize a relation. In th example, if we replace suppliers by two relation schemas

SA (name, address)
SIP (name, item, price)

Then in SA, for each supplier we have an address separately without a redundancy. At the same time, we can enter address for a supplier even if currently supplies no items.

Q.2. Show the different levels of normalization with the help of diagra

Ans. The first three (1NF, 2NF, 3NF) were defined by Codd. Fig. 3 shows, all normalized relations are in 1NF; some 1NF relations are also i 2NF, and some 2NF relations are also in 3NF. Codd's original definition o 3NF turned out to suffer from certain inadequacies. A stronger definition o 3NF, called Boyce-cods normal form (BCNF) was proposed later by Boyc and Codd. All these normal forms are based on the functional dependencie among the attributes of a relation. Fagin defined a fourth normal form (4NF and a fifth normal form (5NF) based on the concepts of multivalued dependencies and join dependencies respectively.

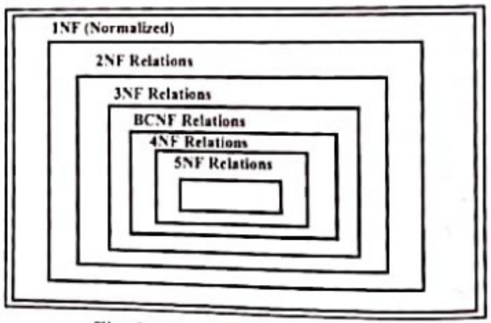


Fig. 3.1 Levels of Normalization

Q.3. What is first normal form (1NF) ? Explain with the help of a example. How do we achieve 1NF ?

Ans. A relation is in 1NF if and only if, for every legal value of the relation, every tuple has exactly one value for each attribute.

Or

A relation is said to be in first normal form (1NF) if the values in the domain of each attribute of the relation are atomic. (A domain is atomic if the elements of domain are considered to be indivisible units.) In other words only one value is associated with each attribute and the value is not a set of values or a list of values. A database is in first normal form if every relation included in database is in 1NF.

Example – First normal form deals only with the basic structures of the relation and does not resolve the problems of redundant information or any other anomalies.

Let us take an example of an organization, where we have information about employee_id, employee_name, phone_no and their salary.

Table 3.1 Non-domain Form

employee_id	*employee_name*	*phone_no*	*salary*
101	Priya	432049	10,000
102	Ashish	420420 323232	15,000
103	Kaushlendra	583119	12,000
104	Piyush	500698 434596	20,000

Here, we see that an employee can have more than one phone number. So, each tuple of above table does not have only one value for each attribute.

For converting it into 1NF we use the following steps –

(i) For each repeating field value, create a new tuple.

(ii) Insert the repeating field value in that field of the new tuple.

(iii) Duplicate the values of all other attributes of the original tuple and put them in the new tuple.

In our example, we have phone_no field of Ashish and Piyush have repeating value. So, we have to create new tuples only for these two employees. This can be done as follows –

Table 3.2 Normalized-form

employee_id	*employee_name*	*phone_no*	*salary*
101	Priya	432049	10,000
102	Ashish	420420	15,000
102	Ashish	323232	15,000
103	Kaushlendra	583119	12,000
104	Piyush	500698	20,000
104	Piyush	434596	20,000

Problems or anomalies to be faced in 1NF are as follows –

(i) INSERT – We cannot insert the information about an employee, who does not have any phone_no.

(ii) DELETE – If we delete the sole first tuple for a particular employee, we delete not only the phone_no of employee but also lose other information, such as salary and employee_id.

For example, if we delete a tuple of employee Priya whose phon disconnected, then we also lose other information of Priya such as sal (Insert and delete are two sides of same coin).

(iii) UPDATE – In this form of relation, we have same inform repeated many times. If we want to update some information, for this first we h to search all the tuples containing that information and then update all that tupl

For example, if salary of Piyush get incremented to 25,000 and we w to update this information in our relation, for this first of all we have to sea all the tuples having employee_name "Piyush" and then update the salary 25,000 of each tuple. But if we forget updation of any of tuple named Piyu then it will result inconsistency. To avoid this, we go for 2NF.

Q.4. What is second normal form (2NF) ? Explain it with the help an example. How is it achieved and what are its advantages over 1NF?

Ans. A relation is in second normal form (2NF), if it is in 1NF and if nonprime attributes are fully functionally dependent on the primary key.

A database schema is in 2NF if every relation included in the databas in 2NF. The steps that involved in transforming the relation from 1NF to 2N are as follows –

(i) Identify the set of attributes that makes up the primary key.

(ii) Create all subsets of the above set obtained in step 1.

(iii) Designate each of these subsets at the primary key of a relati that contains those attributes which are dependent on these primary keys.

Example – For example, we take a relation of marketing departme named sale with the attributes as shown in table 3.3.

Table 3.3 Relation Sale

S#	*Status*	*City*	*P#*	*Pname*	*Color*	*Qty*
S1	10	Paris	P1	Bulb	Pink	300
S1	10	Paris	P2	Tubes	Yellow	100
S2	20	London	P3	Fan	Green	500
S3	30	Italy	P4	Cooler	Black	200
S3	30	Italy	P1	Bulb	Pink	450
S4	40	India	P5	Lamp	White	400
S5	10	Paris	P2	Tubes	Yellow	250

The explanation of the attributes of table 3.3 is as follows –

(i) S# – Supplier number.

(ii) Status – This denotes status of supplier with constraint th CITY determines status.

(iii) City – Name of city in which supplier lives.

(iv) P# – Product number.

(v) Pname – Name of the product.

(vi) Color – Color of the product.

(vii) Qty – Denotes quantity of a product sold by a particular supplier.

This relation is already in 1NF and faces all the anomalies that 1NF relation faces i.e., insertion, deletion and updation. After 1NF, for simplicity we draw the functional dependency diagram (FD diagram) of relation, as shown in fig. 3.2.

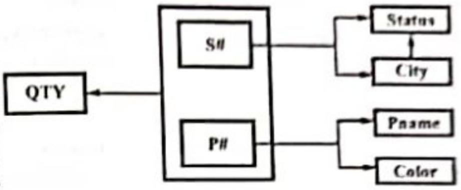


Fig. 3.2 FD for Relation Sale

To convert this relation sale in 2NF, the steps are as follows –

(i) Attributes that make primary key are S# and P# i.e., {S#,P#}.

(ii) Subsets of above set are {}, {S#}, {P#}, {S#, P#}.

(iii) We designate these subsets as the primary key of the relation with attributes that are dependent on these primary key.

So, we have one relation S with primary key S# and attributes Status and City. Another relation P with primary key P# and attributes Pname and Color. Another relation SP with primary key S# and P# with attribute Qty.

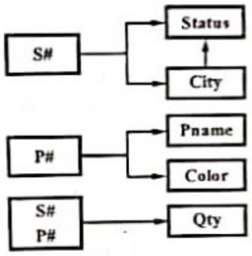


Fig. 3.3 FD Diagram in 2NF

In second normal form, above relation can be given in fig. 3.3.

These relations can be given in following way –

Table 3.4 Sample Values for Relation S, P and SP

(a) Relation S

S#	*Status*	*City*
S1	10	Paris
S2	20	London
S3	30	Italy
S4	40	India
S5	10	Paris

(b) Relation P

P#	*Frame*	*Color*
P1	Bulb	Pink
P2	Tubes	Yellow
P3	Fan	Green
P4	Cooler	Black
P5	Lamp	White

(c) Relation SP

S#	*P#*	*Qty*
S1	P1	300
S1	P2	100
S2	P3	500
S3	P4	200
S3	P1	450
S4	P5	400
S5	P2	250

The advantage of 2NF is that, the problems that we were facing in 1NF are solved in 2NF. These are solved as follows –

(i) Insert – We can insert the information that supplier S lives in city Paris, eventhough S1 does not currently supply any part, by simply inserting appropriate tuple into relation S.

(ii) Delete – We can delete the connection S1 and P2 by deleting a tuple from relation SP, without losing the information that S1 is located in Paris, as this information we have in relation S.

(iii) Update – The city for a given supplier is written once, its not repeated many times as shown in relation S. The primary key of this relation is S#. For one value of S#, there is only one value of city, which was repeated many times in 1NF.

So, if we want to update the city of a particular supplier, updation is required only once. Eventhough, 2NF faces some anomalies for removal of which we go for 3NF. These anomalies are as follows –

(i) Insert – We cannot insert the fact that a particular city has a particular status e.g., we cannot state that a supplier in Italy must have a status 30, until we have some supplier located in that city.

(ii) Delete – If we delete the sole S tuple for a particular city, we delete not only the information for the supplier concerned, but also the information that city has that particular status.

For example, if we delete second tuple from relation S. We lose the information that city London has status 20. At the same time, information that supplier S2 is concerned with city London.

(iii) Update – Status of a city in relation S appears repeatedly.

For example, in relation S status of city Paris appears repeatedly. For updating the status of city Paris, first we have to find all the tuples that have city Paris then we have to update them.

If we miss any tuple, it will result inconsistency.

Q.5. What is normalization ? Explain second normal form with the help of an example. ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.1 and Q.4.

Q.6. Define the term third normal form.

(R.G.P.V., June 2010, Dec. 2011, June 2016)

Ans. A relation is in third normal form (3NF) if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key. By the transitive functional dependency, we mean that –

If, x (primary key) → y and y → z then x → z.

So, z is transitively dependent on x. And according to definition of 3NF as such dependency should be there, that is relation.

Q.7. What is 3NF ? Explain with the help of an example. How is it achieved and what are its advantages over 2NF ?

Ans. **Third Normal Form (3NF)** – Refer to Q.6.

Example – Let us consider the fig. 3.4.

In our example from fig. 3.4 we can say that in relation S –

S# → City

City → Status

So, S# → Status.

This shows transitive dependency, which we have to remove other relationship P and SP does not show any such type of dependency.

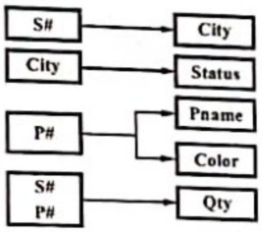


Fig. 3.4 FD Diagram in 3NF

So following the step (i), we determine the non-key attribute that determines the some other non-key attribute. City that is a non-key attribute determines status which is a non-key attribute i.e., for any two equal values of city, status definitely have equal values.

The relation can be given in the following way.

Table 3.5 Relations in 3NF

(a) Relation SC

S#	*City*
S1	Paris
S2	London
S3	Italy
S4	India
S5	Paris

(b) Relation CS

City	*Status*
Paris	10
London	20
Italy	30
India	40

(c) Relation SP

S#	*P#*	*Qty*
S1	P1	300
S1	P2	100
S2	P3	500
S3	P4	200
S3	P1	450
S4	P5	400
S5	P2	250

(d) Relation P

P#	*Pname*	*Color*
P1	Bulb	Pink
P2	Tubes	Yellow
P3	Fan	Green
P4	Cooler	Black
P5	Lamp	White

The advantage is that we can solve the problems that we were facing in 2NF. We can solve them as follows –

(i) INSERT – In relation CS, we can insert a tuple indicating that a particular city has a particular status.

So, we can say that supplier in Italy must have status 30.

(ii) DELETE – On deleting any tuple from relation SC, we do not lose information about the status of city in which that supplier lives.

For example, if we delete second tuple from relation SC, then we also know that city London has status 20.

(iii) UPDATE – Similarly to insertion and deletion, updation is also easy. If we want to change the status of any city then we have to just update one tuple in relation CS. For example, if we wish to change the status of London from 20 to 50 then we have to just update the second tuple in relation CS. Even now problem is not fully solved. For this, we go for further normalization.

Note – It is never possible to solve all the problems or finish the redundancy completely.

Q.8. Write the different steps involved in transforming the relation from 2NF to 3NF.

Ans. Steps involved in transforming the relation from 2NF to 3NF.

(i) Determine the non-key attributes that determine some other non-key attributes.

(ii) Make separate relation taking first one as primary key.

Or

(i) In other words, we can say that we try to find attributes that show transitive dependency between them. This we can search from FD diagram.

(ii) Try to avoid this transitivity by using steps given above.

Q.9. Explain Boyce-Codd normal form with example and also compare BCNF and 3NF. ***(R.G.P.V., Dec. 2010)***

Or

Define BCNF. How does it differ from 3NF ? Explain briefly. ***(R.G.P.V., Nov. 2018)***

Ans. Boyce-codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF, because every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily in BCNF.

The BCNF can be defined as follows –

A relation is in BCNF, if and only if every nontrivial, left irreducible FD has a candidate key as its determinant.

Or

A relation R is said to be in BCNF, if whenever X → A holds in R and A is not in X, then X is a candidate key for R.

In other words, the only arrows in the FD diagram are arrows out of candidate keys, and there are no others, meaning there are no arrows that can be eliminated by the normalization procedure.

Example – Relations Sale and S which are not in 3NF, are not in BCNF while the relations SP, SC, and CS which are in 3NF, are also in BCNF. Because in relation Sale, we have four determinants {S#}, {P#}, {CITY}, {S#, P#} of these only {S#, P#} is a candidate key. So Sale is not in BCNF.

Similarly, in relation S we have 2 determinants {S#}, {CITY}, and CITY is not a candidate key. So S is also not in BCNF.

While relation SP, SC and CS are each in BCNF, because in each case the Sale candidate key is the only determinant in the relation.

Mostly, relations that are in 3NF are also in BCNF. Infrequently, a 3NF relation is not in BCNF and this happens only if –

(i) The candidate keys in the relation are composite keys.

(ii) There is more than one candidate keys in the relation, and

(iii) The keys are not disjoint, that is, some attributes in the keys are common.

Now, we take another example that is in BCNF.

Table 3.6 Relation Player

Name	*Language*	*Hobby*
Nitin	Hindi	Football
Nitin	English	Swimming
Nitin	Hindi	Swimming
Nitin	English	Football
Jitin	French	Jogging
Jitin	Spanish	Jogging

Here, we are supposing name as a unique key. Here, a player can know more than one languages and may have more than one hobbies too. So, each of the attributes is a primary key. So it is in BCNF. But this also suffers from some anomalies, such as, if we want to add one more hobby of Jitin that is singing. Then we can do in following manner–

Table 3.7

Name	*Language*	*Hobby*
Nitin	Hindi	Football
Nitin	English	Football
Nitin	Hindi	Swimming
Nitin	English	Swimming
Jitin	French	Jogging
Jitin	Spanish	Jogging
Jitin	French	Singing

Table 3.8

Name	*Language*	*Hobby*
Nitin	Hindi	Football
Nitin	English	Football
Nitin	Hindi	Swimming
Nitin	English	Swimming
Jitin	French	Jogging
Jitin	Spanish	Jogging
Jitin	French	Singing
Jitin	Spanish	Singing

But this would mean that Jitin has hobby singing in French language but he does not know singing in Spanish language. So, adding a tuple with hobby singing and language Spanish, would result

BCNF faces some anomalies –

(i) ***INSERT*** – If we want to add a single tuple then several others have to be added as given above.

(ii) ***DELETE*** – Similar to insertion, if we delete a single tuple, several other tuples also have to be deleted.

For example, if we want to delete (Nitin, Hindi, Football) then (Nitin, English, Football) also has to be deleted.

Q.10. What is 1NF, 2NF, 3NF and BCNF (Boyce-Codd Normal Form)?
(R.G.P.V., Dec. 201)

Ans. Refer to Q.3, Q.4, Q.6 and Q.9.

Q.11. Differentiate between 3NF and BCNF with examples.
(R.G.P.V., Dec. 201)

Ans. Refer to Q.7 and Q.9.

Q.12. What do you mean by normalization ? Explain BCNF and 3NF with suitable example. ***(R.G.P.V., Dec. 201)***

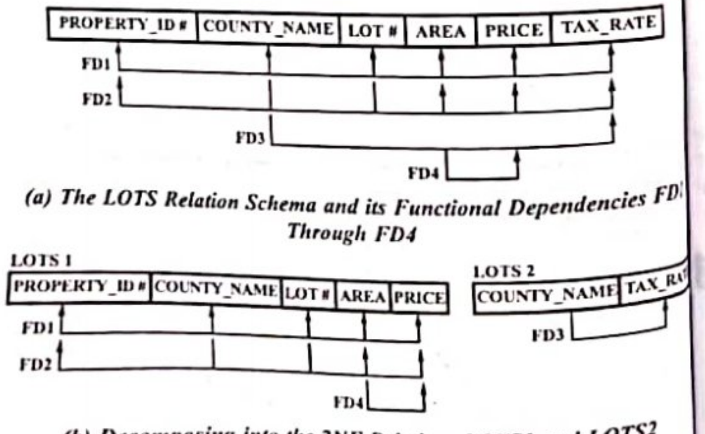
Ans. Normalization – Refer to Q.1.

BCNF – Refer to Q.9.

3NF – Refer to Q.7.

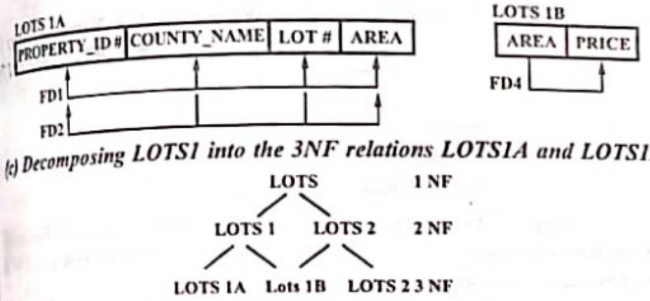
Q.13. Show that if relational schema is in BCNF it is in 3NF but the reverse is not true. ***(R.G.P.V., June 201)***

Ans. Fig. 3.5 shows lots relation schema. Fig. 3.5 (a) contains four functional dependencies, FD1 through FD4. Suppose that, there are lots from only two counties – Dekalb and Fulton. Suppose also that lot sizes for the Dekalb county are .5, .6, .7, .8, .9 and 1.0 acres, whereas lot sizes in fulton county are 1.1, 1.2,.., 1.9 and 2.0 acres. In this situation, we will have the additional functional dependency FD5 : area → county_name. If we add this to other dependencies, the relation LOTS1A still is in 3NF because county_name is a prime attribute. The area of a lot that determines the county can be represented by 16 tuples in a separate relation R(area, county_name), since there are only 16 possible area values. This representation reduces the redundancy of repeating the same information in the LOTS1A tuples.



(a) The LOTS Relation Schema and its Functional Dependencies FD1 Through FD4

(b) Decomposing into the 2NF Relations LOTS1 and LOTS2



(c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B

(d) Summary of Normalization of LOTS

Fig. 3.5 Normalization to 2NF and 3NF

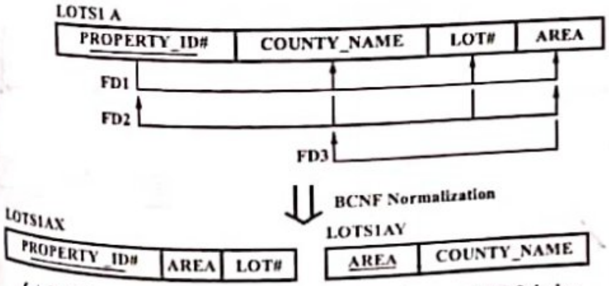
In this example, FD5 violates BCNF in LOTS1A because area is not a superkey of LOTS1A.FD5 satisfies 3NF in LOTS1A because county_name is a prime attribute, but this condition does not exist in the definition of BCNF. We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY as shown in fig. 3.6. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation.

In practice, most relation schemas that are in 3NF are also in BCNF. Only if X → A holds in a relation schema R with X not being a superkey and A being a prime attribute will R be in 3NF but not in BCNF. The relation schema R shown in fig. 3.6 (b) shows the general case of such a relation.

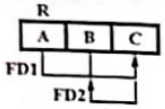
Fig. 3.7 shows a relation TEACH with the following dependencies –

FD1 : {STUDENT, COURSE} → INSTRUCTOR

FD2 : INSTRUCTOR → COURSE



(a) BCNF Normalization with the Dependency of FD2 being Lost in the Decomposition



(b) A Relation R in 3NF but not in BCNF

Fig. 3.6 Boyce-codd Normal Form

In this figure {STUDENT, COURSE} is a candidate key for this relation and the dependencies follow the pattern in fig. 3.6 (b). Hence, this relation is in 3NF but not in BCNF.

EACH

STUDENT	COURSE	INSTRUCTOR
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating System	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

Fig. 3.7 A Relation TEACH that is in 3NF but not in BCNF

Q.14. Explain 3NF and BCNF. Give an example when a relation is in 3NF but not in BCNF. ***(R.G.P.V., Dec. 2009)***

Ans. 3NF – Refer to Q.6.

BCNF – Refer to Q.9.

Example – Refer to Q.13.

Q.15. How 2NF is better than 1NF ? Is BCNF better than 3NF ? Give an example of a relation that is in 3NF but not in BCNF.(R.G.P.V., Dec. 2012)

Ans. **Advantages of 2NF over 1NF** – Refer to Q.4.

Is BCNF better than 3NF – It is observed that any schema that satisfies BCNF also satisfies 3NF. It is because each of its functional dependencies would satisfy one of the two alternatives – (i) α → β is a trivial functional dependency and (ii) α is a superkey for R. Therefore, BCNF is a more restrictive constraint than is 3NF.

S# → Status

Example – Refer to Q.13.

Q.16. Define and explain 4NF with an example.

Or

Prove that a relation which is in 4NF must be in BCNF. ***(R.G.P.V., June 2008, Dec. 2008, 2013)***

Ans. A relation R is in 4NF if and only if, whenever there exists subsets A and B of the attributes of R such that the nontrivial multivalued dependency A →→ B is satisfied, then all attributes of R also functionally dependent on A.

Or

A relation is in 4NF if it is in BCNF and it has no multivalued dependency.

The EMP relation of fig. 3.8 (a) is not in 4NF because in the nontrivial MVDs ENAME →→ PNAME and ENAME →→ DNAME, ENAME is not a superkey of EMP. We decompose EMP into EMP-PROJECTS and EMP-DEPENDENTS as shown in fig. 3.8 (b). Now, both EMP-PROJECTS and EMP-DEPENDENTS are in 4NF, because the MVDs ENAME →→ PNAME in EMP-PROJECTS and ENAME →→ DNAME in EMP-DEPENDENTS are trivial MVDs.

Every 4NF schema is in BCNF. To see this fact, if a schema R is not in BCNF then there is a nontrivial functional dependency α → β holding on R, where α is not a super key. Since α → β implies α → β, R cannot be in 4NF.

EMP

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(a) The EMP Relation with Two MVDs ENAME →→ PNAME and ENAME →→ DNAME

EMP-PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

EMP-DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

(b) Decomposing EMP into Two Relations in 4NF

Fig. 3.8

Q.17. Explain functional dependency. ***(R.G.P.V., Nov./Dec. 2007)***

Or

Define the term functional dependency. (R.G.P.V., June 2010, Dec. 2011)

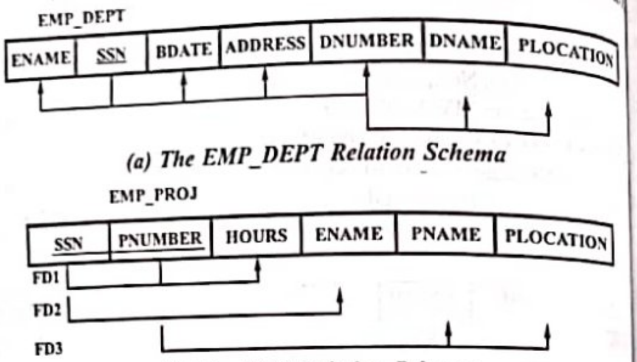
Ans. A functional dependency, (or FD) denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the tuples that can form a relation state of r of R. The constraint is that, for any two tuples t_1 and t_2 in r that have $t_1[X] = t_2[X]$, we must have $t_1[Y] = t_2[Y]$. It means that the values of the Y of a tuple in r depend on, or are determined by, the values of the X component. Alternatively, the values of X component of a tuple uniquely (or functionally) determine the values of the Y component. In other words, there is a functional dependency from X to Y or that Y is functionally dependent on X. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.

Thus, X functionally determines Y in a relation schema R if and only if, whenever two tuples of r(R) agree on their X-value, they must necessarily agree on their Y-value.

It should be noted that – .

(i) If a constraint on R states that there cannot be more than tuple with a given X-value in any relation i.e., X is a candidate key of R, implies that X → Y for any subset of attributes Y of R.

(ii) If X → Y in R, this does not say whether or not Y → X



(a) The EMP_DEPT Relation Schema

(b) The EMP_Relation Schema

Fig. 3.9 Two Relation Schemas and their Functional Dependencies; I Surface Update Anomalies

For example, in the relation schema EMP-PROJ in fig. 3.9, the follo functional dependencies should hold –

(i) SSN → ENAME

(ii) PNUMBER → {PNAME, PLOCATION}

(iii) {SSN, PNUMBER} → HOURS

These functional dependencies specify that (i) the value of an employ social security number (SSN) uniquely determines the employee (ENAME), (ii) the value of a project's number (PNUMBER) uniquely determ that project name (PNAME) and location (PLOCATION) and (iii) a combin of SSN and PNUMBER values uniquely determines the number of hours employee works on the project per week (HOURS). Alternatively, we can that ENAME is functionally determined by SSN, or "given a value of SSN know the value of ENAME", and so on.

Q.18. What do you mean by normalization ? Illustrate BCNF suitable example. How BCNF is better than 3NF ? Justify your answer. explain functional dependency. ***(R.G.P.V., June 2***

Ans. Normalization – Refer to Q.1.

BCNF – Refer to Q.9 and Q.15.

Functional Dependency – Refer to Q.17.

Q.19. Describe about trivial and non-trivial dependencies.

Or

Explain trivial dependency. ***(R.G.P.V., Nov./Dec. 2007)***

Or

Write a brief note on trivial and non-trivial dependencies. ***(R.G.P.V., Dec. 2015)***

Ans. Some functional dependencies are said to be ***trivial*** because they are satisfied by all relations.

For example, A → A is satisfied by all relations involving attribute A. By the definition of functional dependency, we see that, for all types t_1 and t_2 such that $t_1[A] = t_2[A]$, it is the case that $t_1[A] = t_2[A]$.

Similarly, AB → A is satisfied by all relations involving attribute A.

In fact, an FD is trivial if and only if the right-hand side is a subset of the left-hand side.

Non-trivial dependencies are one which is not trivial. That shows the real relations between the attributes of the relation schema.

In general, the functional dependency of the form $\alpha \rightarrow \beta$ is nontrivial if and only if $\beta \not\subseteq \alpha$.

For example, consider the relation –

Product (P#, Pname, Color, Weight).

In this relation, we have following dependencies –

P# → Pname

P# → Color.

These are non-trivial in nature, as they are not showing dependencies among itself.

Q.20. Explain functional dependency and trivial functional dependency with examples. ***(R.G.P.V., Dec. 2017)***

Ans. Refer to Q.17 and Q.19.

Q.21. What are the inference rules for functional dependencies ? ***(R.G.P.V., June 2006)***

Ans. The set of inference rules (also called Armstrong's axioms) by which new FD's can be inferred from given FD's are as follows –

(i) ***Reflexivity*** – If α is a set of attributes and , $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ holds.

(ii) ***Augmentation*** – If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.

(iii) ***Transitivity*** – If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds, then $\alpha \rightarrow \gamma$ holds.

Several other rules derived from the three given rules can be as follow,

(iv) Union – If $\alpha \to \beta$ holds and $\alpha \to \gamma$ holds, then $\alpha \to \beta\gamma$ hol

(v) Decomposition – If $\alpha \to \beta\gamma$ holds, then $\alpha \to \beta$ holds an $\alpha \to \gamma$ holds.

(vi) Pseudotransitivity – If $\alpha \to \beta$ holds and $\gamma\beta \to \delta$ holds, the $\alpha\gamma \to \delta$ holds.

Q.22. Prove or disprove the following inference rules for function dependencies –

(i) $\{W \to Y, X \to Z\} \models \{WX \to Y\}$
(ii) $\{X \to Y\}$ *and* $Y \supseteq Z \models \{X \to Z\}$
(iii) $\{X \to Y, Y \to Z\} \models \{X \to YZ\}$
(iv) $\{X \to Z, Y \to Z\} \models \{X \to Y\}$
(v) $\{XY \to Z, Z \to W\} \models \{X \to W\}$.

(R.G.P.V., Dec. 20

Ans. (i) (a) $W \to Y$ (given)
(b) $X \to Z$ (given)
(c) $WX \to XY$ (by augmenting in (a) by X)
(d) $WX \to Y$ (by decomposition in (c)).

(ii) Given $Y \supseteq Z$ and that two tuples t_1 and t_2 exist in some rel instance r of R such that $t_1[Y] = t_2[Y]$. Then $t_1[Z] = t_2[Z]$ because $Y \supseteq Z$; he $Y \to Z$ must hold. By transitivity $X \to Y$ (given) and $Y \to Z$ then $X \to Z$

(iii) (a) $X \to Y$ (given)
(b) $Y \to Z$ (given)
(c) $Y \to YZ$ (applying augmentation rule on (b) with Y)
(d) $X \to YZ$ (by transitivity rule on (a) and (c)).

(iv) Given $X \to Z$ i.e., $X \supseteq Z$ and that any two tuples t_1 and t_2 of relation R such that $t_1[X] = t_2[X]$ then $t_1[Z] = t_2[Z]$. Also given $Y \to Z$ $Y \supseteq Z$ and that any two tuples t_1 and t_2 of r in relation R such that $t_1[$ $t_2[Y]$ then $t_1[Z] = t_2[Z]$.

This implies that $Y \supseteq X$; i.e., $X \to Y$.

(v) (a) $XY \to Z$ (given)
(b) $Z \to W$ (given)
(c) $XY \to W$ (by transitivity rule on (a) and (b))
(d) $X \to W$ (by decomposition rule on (c)).

Q.23. What is a canonical cover? If
$F = \{A \to BC, CD \to E, E \to C, D \to AEH, ABH \to BD, DH \to$
show that the non-redundant cover for F is
$\{A \to BC, E \to C, D \to AEH, ABH \to BD\}$

(R.G.P.V., Dec.

Ans. A set of functional dependencies F_c is a ***canonical*** cover if every FD in F_c satisfies the following –

(i) Each FD in F_c is simple. Recall that in a simple FD the right-hand side has a single attribute i.e., each FD is of the form $X \to A$.

(ii) For no FD $X \to A$ with $Z \subset X$ is $\{(F_c - (X \to A)) \cup (Z \to A)\}$ F_c. In other words, the left-hand side of each FD does not have any extraneous attributes, or the FDs in F_c are left reduced.

(iii) No FD $X \to A$ is redundant i.e., $\{F_c - (X \to A)\}$ does not logically imply F_c.

A canonical cover is sometimes called ***minimal.***

If $F = \{A \to BC, CD \to E, E \to C, D \to AEH, ABH \to BD, DH \to BC\}$, then a non-redundant cover for F is $\{A \to BC, E \to C, D \to AEH, ABH \to BD\}$. The FD $ABH \to BD$ can be decomposed into the FDs $ABH \to B$ and $ABH \to D$. Now, since the FD $A \to B$ is in F, we can left reduce these decomposed FDs into $AH \to B$ and $AH \to D$. We also notice that $AH \to B$ is redundant since the FD $A \to B$ is already in F. This gives us the canonical cover as $\{A \to B, A \to C, E \to C, D \to A, D \to E, D \to H, AH \to D\}$.

Q.24. What is irreducible set of dependencies ? Relation R with attributes A, B, C, D and FDs.

$$A \to BC, B \to C, A \to B, AB \to C, AC \to D$$

Compute an irreducible set of FDs that is equivalent to this given set.

(R.G.P.V., Dec. 2010)

Ans. **Irreducible Set of Dependencies** – A set S of FDs is irreducible if and only if it satisfies the following three properties –

(i) The right-hand side (the dependent) of every FD in S involves just one attribute (i.e., it is a singleton set).

(ii) The left-hand side (the determinant) of every FD in S is irreducible in turn–meaning that no attribute can be discarded from the determinant without changing the closure S^+(i.e., without converting S into some set not equivalent to S). We will say that such an FD is left-irreducible.

(iii) No FD in S can be discarded from S without changing the closure S^+ (i.e., without converting S into some set not equivalent to S).

With regard to points (ii) and (iii), it is not necessary to know exactly what the closure S^+ is in order to tell whether it will be changed if something is discarded.

Problem – Computation of an irreducible set of FDs that is equivalent to given set is as follows –

(i) The first step is to rewrite the FDs such that each has a singleton right-hand side –

$A \to B, A \to C, B \to C, A \to B, AB \to C, AC \to D$

We observe that the FD A → B occurs twice, so one occurrence can b deleted.

(ii) Next, attribute C can be eliminated from the left-hand side o the FD AC → D, because we have A → C, so A → AC by augmentation an we are given AC → D, so A → D by transitivity. Thus the C on left-hand sid of AC → D is redundant.

(iii) Next, we observe that the FD AB → C can be eliminated, becau again we have A → C, so AB → CB by augmentation, so AB → C by decom position.

(iv) Finally, the FD A → C is implied by the FDs A → B and B → C so it can also be eliminated. We are left with –

A → B, B → C A → C

This set is irreducible.

NUMERICAL PROBLEMS

Prob.1. Compute the closure of the following set F of function dependencies for relation schema –

$$R = (A, B, C, D, E)$$
$$A \to BC,\ CD \to E$$
$$B \to D,\ E \to A$$

List candidate keys for R. Compute B^+ and the canonical cover F_c.
(R.G.P.V., Dec. 2005, June 200

Sol. The members of closure of F, F^+ are as follows –

(i) A → CD. Since by augmentation rule on B → D to inf BC → CD. Now, applying transitivity on A → BC and BC → CD.

(ii) BC → E. Since B → D and CD → E, the pseudotransitivity ru implies BC → E.

(iii) A → B and A → C. Since, A → BC holds, applying decompositi rule we get A → B and A → C.

The candidate keys are CD and BC.

The canonical cover F_c is {A → BC, B → D, E → A, C → E}.

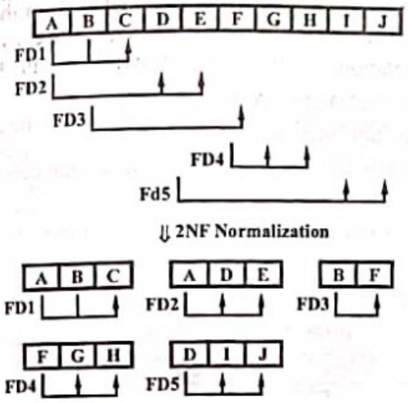
Prob.2. Consider the relation R(A, B, C, D, E, F, G, H, I, J) and set functional dependencies –

$$F = \{\{A, B\} \to \{C\}$$
$$\{A\} \to \{D, E\}$$
$$\{B\} \to \{F\}$$
$$\{F\} \to \{G, H\}$$
$$\{D\} \to \{I, J\}\}$$

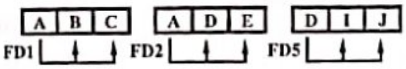
What is the key of R, decompose R in 2NF and 3NF ?
(R.G.P.V., Nov./Dec. 2007, Dec. 2013, June 201

Sol. The key of R is AB.

Normalization into 2NF –



Normalization into 3NF –



Prob.3. Consider a relation R with five attributes A, B, C, D, E having following dependencies : A → B, BC → E and ED → A

(i) List all keys for R

(ii) In which normal form table is, justify your answer.

(R.G.P.V., Dec. 2016)

Sol. (i) A relation R with five attributes A, B, C, D, E having given dependencies contains the following keys – CDE, ACD and BCD.

(ii) The relation R is in 3NF because B, E and A are all parts of keys.

Prob.4. Consider the relation r(A, B, C, D, E) and the set F = {AB → CE, E → AB, C → D}. What is the highest normal form of this relation ?
(R.G.P.V., June 2010)

Sol. Since r is a relation by definition the relation is already in 1NF. To determine the normal form of the given relation, we need to determine all possible keys of the relation. The two possible keys of this relation are AB and E. Let us see why this is so.

AB is a key

(i) AB → A (reflexivity axiom)

(ii) AB → B (reflexivity axiom)

(iii) From AB → CE (given) we have that AB → C (projectivity axiom).

(iv) From AB → CE (given) we have that AB → E (projectivity axiom).

(v) From AB → C (step (iii)) and C → D (given) we have that AB → D (Transitivity axiom).

Since AB determines all attributes of the relation AB is a key.

E is a key.

Since E → AB (given) and AB is a key then E is also a key because b application of the transitivity axiom E can functionally determine all oth attributes of the relation.

The prime attributes are – A, B and E

The nonprime attributes are – C and D

The relation is in 2NF because there are no partial dependencies on any the key.

The relation is not in 3NF because there is a transitivity dependency the keys. Notice that C → D and both attributes are nonprime.

DECOMPOSITION, DEPENDENCY PRESERVATION AND LOSSLESS JOIN, PROBLEMS WITH NULL VALUED AND DANGLING TUPLES, MULTIVALUED DEPENDENCIES

Q.25. Discuss the term decomposition and give its properties.

Ans. The decomposition of a relation schema R = {A_1, A_2, ...,A_n} is replacement by a collection

$$\rho = \{R_1, R_2, ...,R_k\} \text{ of subsets of R such th}$$
$$R = R_1 \cup R_2 \cup ,......, \cup R_k$$

There is no requirement that the R_iS be disjoint. One of the motivatio performing a decomposition is that it may eliminate some of problems. F example, consider relation Sale (S#, Status, City, P#, Pname, Color, City) this relation, we cannot insert the information that a supplier S1 lives in c Paris, if S1 does not currently supply any parts. It can be done by decompos the relation Sale into relations S, SP, and P, as shown in table 3.4. This ma possible to insert the information that supplier S1 is located in city Paris, ev though S1 does not currently supply any part, by simply inserting a tuple i relation S.

A decomposition may be lossy or lossless. A decomposition of a relat R into relations R_1, R_2, R_3, ..., R_n is called a lossless-join decomposit (with respect to FDs F) if the relation R is always the natural join of relations R_1, R_2, ..., R_n. It should be noted that natural join is the only w to recover the relation from the decomposed relations. There is no other se operators that can recover the relation if the join cannot.

If we are able to recover the original relation from natural join decomposed relations, then this type of relation is lossless, otherwise relat is lossy.

Properties – The properties of decomposition are –

(i) Non-loss or lossless decomposition.

(ii) Dependency preservation.

(iii) No need of repetition of information.

Q.26. Define the term dependency preservation.

(R.G.P.V., Nov./Dec. 2007, June 2010, Dec. 20

Or

Explain dependency preservation with example. (R.G.P.V., Dec. 2009)

Ans. It is desirable for a decomposition to have the lossless-join property, because it guarantees that any relation can be recovered from its projections.

Another important property of a decomposition of relation schema R into ρ = (R_1, ..., R_k) is that the set of dependencies F for R be implied by the projection of F onto the R_iS.

Also, if a relation schema has functional dependency, A → B, then these two attributes are closely related. It would be useful to have such type of attributes in same relation. So it is desirable that, decompositions be such that, each dependency in F may be checked by looking at only one relation and that no joins need to be computed for checking dependencies.

For example, let us consider the relation R (A, B, C) with functional dependencies –

$$A \rightarrow C \quad \text{and} \quad B \rightarrow C$$

If we decompose this relation into two relations AB and BC, then we can easily check FD, B → C by looking at relation BC, but we would not be able to check dependency A → C by looking only at one relation.

So this decomposition is not desirable.

Consider another example of following table –

Table 3.9 having relation S(S#, City, Status) with functional dependencies–

$$S\# \rightarrow City$$
$$City \rightarrow Status$$

So, transitively we have

$$S\# \rightarrow Status$$

Table 3.9 Relation S

S#	*Status*	*City*
S1	10	Paris
S5	10	Holand

In fig. 3.10, dotted line shows transitive dependency, while bold lines show direct dependencies. We have a problem in this relation i.e., we cannot insert status of a city if that city does not have any supplier.

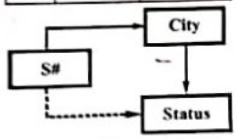


Fig. 3.10 FD's for Relation S

This problem could be solved by decomposing relation S into two relations.

$$SC\ (S\#,\ City) \quad \text{and} \quad CS\ (City,\ Status)$$

This decomposition can solve our problem, now we can easily insert the status of a city even if it does not have any supplier. Also, this decomposition is lossless.

At the same time, we can easily check dependency;

$$S\# \rightarrow City \text{ and } City \rightarrow Status$$

By looking only at one relation at a time. If these two FD's hold, the transitive dependency, S# → Status, holds automatically. So, this decomposition preserves dependency. Hence, this decomposition is desirable.

Consider another possible decomposition of relation S –

$$SC\ (S\#, City) \text{ and } SS\ (S\#, Status)$$

This decomposition is also lossless. But these two relations or projections are not independent i.e., here also we can not insert the status of a city, if the city does not have any supplier.

Also, problem arises, if a supplier move from one city to another, the second relation also has to be updated for changing the status of that supplier. So we can say that, these projections are not independent.

Now, come to the point of dependency preservation,we can check FD's

$$S\# \rightarrow City \text{ and } S\# \rightarrow Status,$$

But, we cannot check City → Status by looking only at one relation.

So, we can say that this decomposition does not preserve dependency and hence is not desirable.

Q.27. Explain lossless join decomposition with example.

(R.G.P.V., Dec. 20

Or

Define the term lossless decomposition.

(R.G.P.V., Nov./Dec. 2007, June 2010, Dec. 20

Or

What do you mean by the terms lossless decomposition ?

(R.G.P.V., Dec. 20

Or

What is meant by lossless join decomposition ? (R.G.P.V., May 20

Ans. Let R be a relation schema and let F be a set of FDs over R. decomposition of R into two schemas with attribute sets X and Y is said to lossless-join or nonadditive or nonloss join decomposition with respect to F if every instance r of R that satisfies the dependencies in F, $\Pi_X(r) \bowtie \Pi_Y(r) = r$ other words, we can recover the original relation from the decomposed relation.

When a relation is decomposed into a number of smaller relations, the is extremely important that the decomposition be lossless. The following simple test is very useful.

Let R be a relation schema, and let F be a set of functional dependencies on R. Let R_1 and R_2 form a decomposition of R. This decomposition is lossless (nonadditive) join decomposition of R if at least one of the following functional dependencies is in F^+ –

$$\text{(i) } R_1 \cap R_2 \rightarrow R_1 \quad \text{(ii) } R_1 \cap R_2 \rightarrow R_2$$

In other words, if $R_1 \cap R_2$ forms a superkey of either R_1 or R_2, the decomposition of R is lossless-join decomposition.

For example, the relation schema lending-schema is as follows –

lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)

Now, we have to show that the decomposition of this schema is lossless-join decomposition. Suppose, lending-schema is decomposed into two schemas –

branch-schema = (branch-name, branch-city, assets)

loan-info schema = (branch-name, customer-name, loan-number, amount)

Since branch-name → branch-city assets, the augmentation rule for functional dependency implies that

branch-name → branch-name branch-city assets

Since branch-schema ∩ loan-info-schema = {branch-name}, it follows that our initial decomposition is a lossless-join decomposition.

Next, we decompose loan-info-schema into

loan-schema = (loan-number, branch-name, amount)

borrower-schema = (customer-name, loan-number).

This step results in a lossless-join decomposition since loan-number is a common attribute and loan-number → amount branch-name.

Q.28. Explain non loss decomposition and functional dependencies with example. ***(R.G.P.V., June 2016)***

Ans. Refer to Q.27 and Q.17.

Q.29. Give an example of a relation schema R' and set F' of functional dependencies such that there are at least three distinct lossless join decomposition of R' into BCNF. ***(R.G.P.V., June 2008, 2009)***

Ans. Fig. 3.11 shows an algorithm to decompose a relation schema so as to satisfy BCNF. If R is not in BCNF, then it can be decomposed into a collection of BCNF schemas $R_1, R_2, \ldots, R_n$ by the algorithm. The algorithm uses dependencies that demonstrate violation of BCNF to perform the decomposition.

The decomposition that the algorithm generates is not only in BCNF, b is also a lossless-join decomposition

```
result= {R};
done : = false;
compute F⁺;
while (not done) do
    if (there is a schema Rᵢ in result that is not in BCNF)
        then begin
            let α → β be a nontrivial functional dependency that holds on Rᵢ
            such that α → Rᵢ is not in F⁺, and
            α ∩ β = ϕ;
            result : = (result – Rᵢ) ∪ (Rᵢ – β) ∪ (α, β);
    end
else done : = true;
```

Fig. 3.11 BCNF Decomposition Algorithm

For example, the BCNF decomposition algorithm is applied to Lending-schem Lending-schema = (branch_name, branch_city, assets, customer_nam loan_number, amou

The set of functional dependencies that are required to hold on Lendin schema are

branch_name → assets branch_city
loan_number → amount branch_name

A candidate key for this schema is {loan_number, customer_name}

The algorithm of fig. 3.11 is applied to the Lending-schema as follow

(i) The functional dependency
branch_name → assets branch_city

holds on *Lending_schema*, but *branch_name* is not a superkey. Th *Lending_schema* is not in BCNF. We replace *Lending_schema* by

Branch_schema = (*branch_name, branch_city, assets*)
Loan_info_schema = (*branch_name, customer_name, loan_number, amou*

(ii) The only nontrivial functional dependencies that hold on *bran schema* include *branch_name* on the left side of the arrow. Since *branch_n* is a key for *Branch_schema*, the relation *Branch_schema* is in BCNF.

(iii) The functional dependency
loan_number → amount branch_name

holds on *loan_info_schema*, but loan-number is not a key for *loan_info_sche* We replace *loan-info-schema* by

loan_schema = (*loan_number, branch_name, amount*)
borrower_schema = (*customer_name, loan_number*)

(iv) *loan_schema* and *Borrower_schema* are in BCNF.

Thus, the decomposition of *Lending_schema* results in the three relation sche *Branch_schema, loan_schema,* and *Borrower_schema*, each of which is in BC

Q.30. *What do you mean by decomposition of a relation ? Consider the relational scheme –*

R (A, B, C, D, E, F) and FD.

$A \rightarrow BC$, $C \rightarrow A$, $D \rightarrow E$, $F \rightarrow A$, $E \rightarrow D$

Is the decomposition of R into $R_1(A, C, D)$, R_2 (B, C, D) and R_3 (E, F, D) lossless ?

Explain the requirements for lossless decomposition and dependency preserving. **(R.G.P.V. Dec. 2011)**

Ans. **Decomposition** – Refer to Q.25.

Step (i) – Since the original relation has 6 attributes and the original relation has been decomposed into 3 relations, we need to create a table with 6 columns and 3 rows. The column of the table are named A, B, C, D, E and F respectively. The rows of the table are named R_1, R_2 and R_3. For explanation purposes we have renamed the attributes A_1,...,A_6. The table is shown below –

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$	$F(A_6)$
R_1						
R_2						
R_3						

Step (ii) – The attributes of the scheme of R_1 are – $A(A_1)$, $C(A_3)$ and $D(A_4)$. Therefore, we place a_1, a_3 and a_4 under these columns, respectively. The remaining entries of this row are filled with b_{12}, b_{15} and b_{16}.

The attributes of the scheme R_2 are – $B(A_2)$, $C(A_3)$ and $D(A_4)$. Therefore, we place a_2, a_3 and a_4 under these columns respectively. The remaining entries of this row are filled with b_{21}, b_{25} and b_{26}.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$	$F(A_6)$
R_1	a_1	b_{12}	a_3	a_4	b_{15}	b_{16}
R_2	b_{21}	a_2	a_3	a_4	b_{25}	b_{26}
R_3	b_{31}	b_{32}	b_{33}	a_4	a_5	a_6

The attributes of the scheme R_3 are – $E(A_5)$, $F(A_6)$ and $D(A_4)$. Therefore, we place a_5, a_6 and a_4 under these columns respectively. The remaining entries of this row are filled with b_{31}, b_{32} and b_{33}.

Step (iii) Ist Time – Considering $A \rightarrow BC$ we check if the rows of table have the same value under the columns that make up the determinant of the FD. Since the rows do not have identical values, we repeat step (iii) and consider another FD.

IInd Time – Considering $C \rightarrow A$ we check if the rows of the table have the same value under the columns that make up the determinant of the FD. Since the rows do not have identical value, we repeat step (iii) and consider another FD.

IIIrd Time – Considering D → E we check if the rows of the table have the same value under the columns that make up the determinant of the FD. Rows has value a_4 under column D. Therefore, we need to equate all the corresponding entries for these rows under column E. Since the entry under the column E is a_5 (for row R_3), we make all the remaining b_{ij}'s equal to a_5 in this column.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$	$F(A_6)$
R_1	a_1	b_{12}	a_3	a_4	a_5	b_{16}
R_2	b_{21}	a_2	a_3	a_4	a_5	b_{26}
R_3	b_{31}	b_{32}	b_{33}	a_4	a_5	a_6

IVth Time – Considering F → A we check if the rows of the table have the same value under the columns that make the determinant of the FD. Since the rows do not have identical value, we repeat step (iii) and consider another FD.

Vth Time – Considering E → D we check if the rows of the table have the same value under the columns that make the determinant of the FD. Rows has value a_5 under the column E. Therefore, we need to equate all the corresponding entries for these rows under the column D, which is already exist.

VIth Time – There are no more FDs to consider and the table cannot undergo any more changes. Therefore, we move to step (iv).

Step (iv) – We now look for a row that has all a_i's. Since there is no row in the table that has all a_i's in its entries the decomposition is lossy.

Whenever a relation is decomposed it is necessary to ensure that the data in the original relation is represented faithfully by the data in the relations that are the result of the decomposition, i.e., we need to make sure that we can recover the original relation from the new relations that have replaced it.

Dependency preserving property requires that the decomposition satisfy all the FDs that are satisfied by the original relation. The reason this is desirable is that the set of functional dependencies that are satisfied by a relation define integrity constraints that the relation needs to meet.

Q.31. Describe NULL value and dangling tuple problems. ***(R.G.P.V., Dec. 2006)***

Or

What is null value problem ? ***(R.G.P.V., Dec. 2016)***

Ans. The problems associated with nulls must be carefully considered while designing a relational database schema. There is no fully satisfactory relational design theory as yet that includes null values. One problem occurs when some tuples have null values for attributes that will be used to JOIN individual relations in the decomposition. To illustrate this, consider the database shown in fig. 3.12 (a), where two relations EMPLOYEE and DEPARTMENT are shown. The last two employee tuples – Berger and Benitez – represent newly hired employees who have not yet been assigned to a department. Now suppose that we want to retrieve a list of (ENAME, DNAME) values for all the employees. If NATURAL JOIN operation is applied on EMPLOYEE and DEPARTMENT (fig. 3.12 (b)), the two aforementioned tuples will not appear in the result. The OUTER JOIN operation can deal with this problem. If we take the LEFT OUTER JOIN of EMPLOYEE with DEPARTMENT, tuples in EMPLOYEE that have null for the join attribute will still appear in the result, joined with an "imaginary" tuple in DEPARTMENT that has nulls for all its attribute values. Fig. 3.12 (c) shows the result.

EMPLOYEE

ENAME	SSN	BDATE	ADDRESS	DNUM
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	null
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	null

DEPARTMENT

DNAME	DNUM	DMGRSSN
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

(a) A Database with Nulls for Some Join Attributes

ENAME	SSN	BDATE	ADDRESS	DNUM	DNAME	DMGRSSN
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

(b) Result of Applying NATURAL JOIN to the EMPLOYEE and DEPARTMENT Relations

ENAME	SSN	BDATE	ADDRESS	DNUM	DNAME	DMGRSSN
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	null	null	null
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	null	null	null

(c) Result of Applying OUTER JOIN to EMPLOYEE and DEPARTMENT

Fig. 3.12 Illustrating Null Value Join Issues

In general, whenever a relational database schema is designed where two or more relations are interrelated via foreign keys, particular care must be devoted to watching for potential null values in foreign keys. This can cause unexpected loss of information in queries that involve joins on that foreign key.

EMPLOYEE_1

ENAME	SSN	BDATE	ADDRESS
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX

(a) The Relation EMPLOYEE_1

(Includes All Attributes of EMPLOYEE Except DNUMBER)

EMPLOYEE_2

SSN	DNUM
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	null
888664444	null

(b) The Relation EMPLOYEE_2 (Includes DNUMBER Attribute with Null Values)

EMPLOYEE_3

SSN	DNUM
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

(c) The Relation EMPLOYEE_3 (Includes DNUMBER Attribute but does not include Tuples for which DNUMBER has Null Values)

Fig. 3.13 The "Dangling Tuple" Problem

A related problem is that of ***dangling tuples,*** which may occur if we carry a decomposition too far. Suppose that we decompose the EMPLOYEE relation of fig. 3.12 (a) further into EMPLOYEE_1 and EMPLOYEE_2, shown in fig. 3.13 (a) and (b). If we apply the NATURAL JOIN operation to EMPLOYEE_ and EMPLOYEE_2 we obtain the original EMPLOYEE relation. However, we may use the alternative representation shown in fig. 3.13 (c), where we do not include a tuple in EMPLOYEE_3 if the employee has not been assigned a department. If we use EMPLOYEE_3 instead of EMPLOYEE_2 and apply a NATURAL JOIN on EMPLOYEE_1 and EMPLOYEE_3, the tuples for Berger and Benitez will not appear in the result. These are called *dangling tuples* because they are represented in only one of the two relation that represent employees and hence are lost if we apply an (inner) join operation.

Q.32. Explain dangling tuple. Is BCNF is a stronger normal form than 3NF ? Mention reason. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.31 and Q.15.

Q.33. Discuss how dangling tuple may arise ? ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.31.

Q.34. Explain multivalued dependency. ***(R.G.P.V., Nov./Dec. 2007)***

Or

What is multivalued dependencies ? ***(R.G.P.V., Dec. 2015)***

Ans. Functional dependencies rule out certain tuples from being in a relation i.e., if A → B then we cannot have two tuples with the same A value but different B values. On the other hand, multivalued dependencies do not rule out the existence of these tuples. That is why, functional dependencies sometimes referred to as ***equality-generating dependencies*** and multivalued dependencies are referred to as ***tuple-generating dependencies.***

Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency $\alpha \rightarrow\rightarrow \beta$ holds on R if, in any legal relation r(R), for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$
$$t_3[\beta] = t_1[\beta]$$
$$t_3[R - \beta] = t_2[R - \beta]$$
$$t_4[\beta] = t_2[\beta]$$
$$t_4[R - \beta] = t_1\ [R - \beta]$$

Whenever $\alpha \rightarrow\rightarrow \beta$ holds we say that α multidetermines y.

Fig. 3.14 shows a tabular representation of t_1, t_2, t_3 and t_4. Intuitively, the multivalued dependency $\alpha \rightarrow\rightarrow \beta$ say that the relationship between α and β is independent of the relationship between α and R – β. If the multivalued dependency $\alpha \rightarrow\rightarrow \beta$ is satisfied by all relations on schema R, then $\alpha \rightarrow\rightarrow \beta$ is a trivial multivalued dependency on schema R. Thus, $\alpha \rightarrow\rightarrow \beta$ is trivial if $\beta \subseteq \alpha$ or $\beta \cup \alpha = R$.

	α	β	$R-\alpha-\beta$
t_1	$a_1 \cdots a_i$	$a_{i+1} \cdots a_j$	$a_{j+1} \cdots a_n$
t_2	$a_1 \cdots a_i$	$b_{i+1} \cdots b_j$	$b_{j+1} \cdots b_n$
t_3	$a_1 \cdots a_i$	$a_{i+1} \cdots a_j$	$b_{j+1} \cdots b_n$
t_4	$a_1 \cdots a_i$	$b_{i+1} \cdots b_j$	$a_{j+1} \cdots a_n$

Fig. 3.14 Tabular Representation of $\alpha \rightarrow\rightarrow \beta$

Q.35. Differentiate between functional dependency and multivalued dependency.

Ans. Functional dependencies rule out certain types from being in a relation. If A → B, then there cannot be two tuples with the same A value but different

B values. Multivalued dependencies, on the other hand, do not rule out the existence of certain tuples. Instead, they require that other tuples of a certain form be present in the relation.

Q.36. Explain fifth normal form with example. *(R.G.P.V., Dec. 2009)*

Ans. There may be no functional dependency in a relation schema R that violates any normal form up to BCNF and there may be no nontrivial MVD in R that violates 4NF. We then use another dependency called the join dependency. It carries out a multiway decomposition into fifth normal form (5NF).

A join dependency (JD), denoted by JD·(R_1, R_2, ..., R_n), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legal state r of R should have a lossless join decomposition into R_1, R_2, ..., R_n, i.e. for every such r we have

$$* (\Pi_{R1}(r), \Pi_{R2}(r), \ldots, \Pi_{Rn}(r)) = r$$

An MVD is a special case of a JD where n = 2. That is, a JD denoted as JD (R_1, R_2) implies an MVD ($R_1 \cap R_2$) $\twoheadrightarrow$ ($R_1 - R_2$). A join dependency JD (R_1, R_2, ... R_n), specified on relation schema R, is a ***trivial*** JD if one of the relation schemas R_i in JD (R_1, R_2, ..., R_n) is equal to R. Such a dependency is called trivial because it has the lossless join property for any relation state r of R and hence does not specify any constraint on R.

A relational schema R is in fifth normal form (5NF) or projection join normal form (PJNF) with respect to a set F of functional multivalued, and join dependencies if, for every non-trivial join dependency JD (R_1, R_2, ... R_n) in F^+ (that is implied by F), every R_i is a superkey of R.

For an example of JD, consider the SUPPLY all-key relation of fig. 3.15 (a). Suppose that the following constraint always holds – Whenever a supplier s supplies part p, and a project j uses part p, and the supplier s supplies at least one part to project j, then supplier s will also be supplying part p to project j. This constraint can be restated as specifying a join dependency JD (R_1, R_2, R_3) among the three projections R1 (SNAME, PARTNAME), R_2 (SNAME, PROJNAME), and R_3 (PARTNAME, PROJNAME) of SUPPLY. If this constraint holds, the tuples below the dotted line in fig. 3.15(a) must exist in any legal state of the SUPPLY relation that also contains the tuples above the dotted line. Fig. 3.15(b) shows how the SUPPLY relation with the join dependency is decomposed into three relations R_1, R_2 and R_3 that are each in 5NF.

SUPPLY

SNAME	PARTNAME	PROJNAME
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nall	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(a) The Relation SUPPLY with no MVDs Satisfies 4NF but does not Satisfy 5NF if the JD(R_1, R_2, R_3) Holds

R_1

SNAME	PARTNAME
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nall

R_2

SNAME	PROJNAME
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

PARTNAME	PROJNAME
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nall	ProjX

(b) Decomposing SUPPLY into three 5NF Relations

Fig. 3.15

Q.37. What is join dependency ? How is it different to that of multivalued dependency and functional dependency ? *(R.G.P.V., Dec. 2014)*

Ans. Refer to Q.36.

Q.38. What is normalization ? Discuss various normal forms with the help of examples. *(R.G.P.V., June 2010)*

Ans. **Normalization** – Refer to Q.1.

Types of Normalization – Refer to Q.3, Q.4, Q.7, Q.9, Q.16 and Q.36.

NUMERICAL PROBLEMS

Prob.5. Given the relation r(X, Y, W, Z, P, Q) and the set F = {XY → W, XW → P, PQ → Z, XY → Q}, consider the decomposition R_1(Z, P, Q), R_2(X, Y, Z, P, Q). Is this decomposition lossless or lossy ?

Use the lossless-join algorithm. *(R.G.P.V., June 2010)*

Sol. **Step I** – Construct a table with six columns (one column per attribute) and two rows (one row per relation of the decomposition). Name each column with the name of one attribute and each row with the name of a relation.

For explanation purpose we have renamed the attributes A_1, A_2,....,A_6.

	X(A_1)	Y(A_2)	W(A_3)	Z(A_4)	P(A_5)	Q(A_6)
R_1						
R_2						

Step II – Fill in the entries as follows –

The attributes of the scheme of R_1 are – $Z(A_4)$, $P(A_5)$, $Q(A_6)$. Therefore, we place a_4, a_5 and a_6 under these columns respectively. The remaining entries of this row are filled with b_{11}, b_{12} and b_{13} respectively.

The attributes of the scheme of R_2 are – $X(A_1)$, $Y(A_2)$, $Z(A_4)$, $P(A_5)$, $Q(A_6)$. Therefore, we place a_1, a_2, a_4, a_5 and a_6 under these columns respectively. The remaining entry of this row is filled with b_{23}.

	$X(A_1)$	$Y(A_2)$	$W(A_3)$	$Z(A_4)$	$P(A_5)$	$Q(A_6)$
R_1	b_{11}	b_{12}	b_{13}	a_4	a_5	a_6
R_2	a_1	a_2	b_{23}	a_4	a_5	a_6

Step III – (i) Considering XY → W we check if the two rows of the table have the same value under the columns XY that make up the determinant of the FD. Since the rows do not have identical values, the table will remain unchanged and we repeat step III by considering another FD.

(ii) Considering XW → P we check if the two rows of the table have the same value under the columns XW that make up the determinant of the FD. Since the rows do not have identical values, the table will remain unchanged and we repeat step III by considering another FD.

(iii) Considering PQ → Z we check if the two rows of the table have the same value under the columns PQ that make up the determinant of the FD. Since the rows R_1 and R_2 have values of a_5 and a_6 under the columns PQ. Therefore, we need to equate all the corresponding entries for the rows under column $Z(A_4)$. Since all the values under column Z are already equal no changes are necessary. Repeat step III again.

(iv) Considering XY → Q we check if the two rows of the table have the same value under the columns XY that make up the determinant of the FD. Since the rows do not have identical values, the table will remain unchanged. Now, there are no more FDs to consider and the table cannot undergo any more. Therefore, we move to step 7IV.

Step IV – Since there is no row in the table that has all a';s in its entires the *decomposition is lossy*. That is, the original table cannot be recovered from the natural join of relations R_1 and R_2.

Prob.6. Suppose that we decompose the scheme R = (A, B, C, D, E) into (A, B, C) and (A, D, E). Show that this decomposition is a lossless join decomposition if the following functional dependencies hold –

A → BC, CD → E, B → D

Show that this decomposition is dependency preserving decomposition.

(R.G.P.V., June 2011)

Sol. **Step (i)** – Since the original relation has 5 attributes and the original relation has been decomposed into 2 relations, we need to create a table with

columns and 2 rows. The columns of the table are named A, B, C, D and E respectively. The rows of the table are named R_1 and R_2. For explanation purposes we have renamed the attributes A_1,...., A_5. The table is shown in table.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1					
R_2					

Step (ii) – The attributes of the scheme of R_1 are – $A(A_1)$, $B(A_2)$, $C(A_3)$. Therefore, we place a_1, a_2 and a_3 under these columns, respectively. The remaining entries of this row are filled with b_{14} and b_{15}.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	a_1	b_{22}	b_{23}	a_4	a_5

The attributes of the scheme of R_2 are – $A(A_1)$, $D(A_4)$, $E(A_5)$. Therefore, we place a_1, a_4 and a_5 under these columns respectively. The remaining entries of this row are filled with b_{22} and b_{23}.

Step (iii) Ist Time – Considering A → BC we check if the two rows of table have the same value under the columns that make up the determinat of the FD. Rows R_1 and R_2 has values a_1 under column A. Therefore,,we need to equate all the corresponding entries for these rows under columns B and C. Since entry under column B is a_2 (for row R_1), we make all the remaining b_{ij}'s equal to a_2 in this column. The entry under C column is a_3, we make the corresponding entry a_3 for this column.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1	a_1	a_2	a_3	b_{14}	b_{15}
R_2	a_1	a_2	a_3	a_4	a_5

IInd Time – Considering CD → E, we check for rows that have the same value in the columns C and D. In this case, rows do not have identical values we repeat step (iii) and consider another FD.

IIIrd Time – Considering B → D, we check if the two rows of the table have the same value under the columns that makeup the determinant of the FD. Since the rows have identical values under column A_2, we equate the corresponding entries under A_4 column. The entries under A_4 are b_{14} and a_4, so we take entry a_4 under A_4 column arbitrarily.

	$A(A_1)$	$B(A_2)$	$C(A_3)$	$D(A_4)$	$E(A_5)$
R_1	a_1	a_2	a_3	a_4	b_{15}
R_2	a_1	a_2	a_3	a_4	a_5

IVth Time – There are no more FDs to consider and the table cannot undergo any more changes. Therefore we move to step (iv).

Step (iv) – We now look for a row that has all a_i's. Since row R_2 has become $a_1a_2a_3a_4a_5$, the decomposition is lossless.

Prob.7. Consider the relation Student (stid, name, course, year). Given that a student may take more than one course but has unique name and the year of joining.

(i) Identify the functional and multivalued dependencies for student.

(ii) Identify a candidate key using the functional and multivalued dependencies arrived at in step (i).

(iii) Normalize the relation so that every decomposed relation is in 4NF.

(R.G.P.V., Dec. 2010)

Sol. Suppose the student relation is as follows –

stid	*name*	*course*	*year*
s1	n1	c1	y1
s1	n1	c2	y1
s2	n2	c1	y2
s3	n3	c2	y3

(i) Functional dependencies are –

stid → name

stid → year

name → year

Multivalued dependencies are –

stid, name, year →→ course

(ii) The candidate key is *name.*

(iii) We find that stid, name, year →→ course is a nontrivial multivalued dependency. We replace the student schema by two schemas -

(stid, course) and (stid, name, year)

QUERY OPTIMIZATION – INTRODUCTION, STEPS OF OPTIMIZATION, VARIOUS ALGORITHMS TO IMPLEMENT SELECT, PROJECT AND JOIN OPERATIONS OF RELATIONAL ALGEBRA, OPTIMIZATION METHODS – HEURISTIC BASED, COST ESTIMATION BASED

Q.39. State the purpose of query optimization. (R.G.P.V., June 2016)

Ans. The purpose of query optimization is to make a database program and the machine it controls – work smart, not hard. In order to retrieve information efficiently, a database needs to analyze each query it receives and do some planning – hopefully figuring out the shortest route to the goal before it starts shifting through data. The plan must take into account how much data there is, how it is grouped and sorted, whether or not it is been indexed, and what must be done to ensure that the answer is technically correct.

Q.40. What is query optimization ? Discuss various steps of optimization.
(R.G.P.V., Dec. 2009)

Or

Explain various steps of query optimization. Also discuss optimization methods.
(R.G.P.V., Dec. 2013)

Or

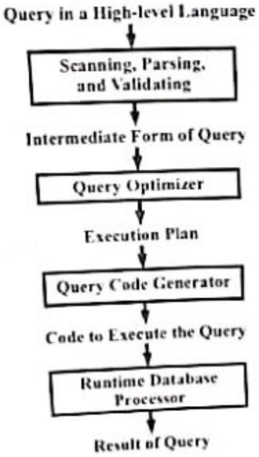
What is query optimization ? (R.G.P.V., Dec. 2014)

Or

How query optimization is performed ? (R.G.P.V., Dec. 2016)

Ans. A query has many possible execution strategies, and the process of choosing a suitable one for processing a query is known as query optimization. The aspect of optimization takes place at the relational algebra level, where the system tries to find an expression that is equivalent to the given expression, but more efficient to execute. Second aspect is to choose a detailed strategy for processing the query, like selecting the algorithm to use for executing an operation, selecting the specific indices to use, and so on. A query expressed in a high-level query language such as SQL must first be scanned, parsed, and validated. The scanner identifies the language tokens, such as SQL keywords, attribute names, and relation names, in the text of the query, whereas the parser checks the query syntax to determine whether it is formulated according to the syntax rules (i.e., rules of grammar) of the query language. The query must also be validated, by checking that all attribute and relation names are valid and semantically meaningful names in the schema of the particular database being queried. An internal representation of the query is then created as a tree data structure called a ***query tree.*** It is also possible to represent the query using a graph data structure called a ***query graph.*** The DBMS must then device an execution strategy for retrieving the result of the query from the database files.

Fig. 3.16 shows the various steps of optimization. The ***query optimizer*** module has the task of producing an execution plan, and the ***code generator*** generates the code to execute that plan. The ***runtime database processor*** has the task of running the query code, whether in compiled or interpreted mode, to produce the query result. If a runtime error results, an error message is generated by the runtime database processor.


Query in a High-level Language
Scanning, Parsing, and Validating
Intermediate Form of Query
Query Optimizer
Execution Plan
Query Code Generator
Code to Execute the Query
Runtime Database Processor
Result of Query

Code can be –
- Executed Directly (Interpreted Mode)
- Stored and Executed Later whenever Needed (Compiled Mode)

Fig. 3.16 Steps of Optimization

Q.41. Discuss the different algorithms for each of the following relational algebra operations –

(i) Select (ii) Join (iii) Project.

Ans. (i) Select Operation – There are many options for executing a SELECT operation. Some depend on the file having specific access paths and may apply only to certain types of selection conditions. Some of the algorithms for implementing SELECT are discussed using the following example operations –

(op1) – $\sigma_{SSN = '12345'}$ (EMPLOYEE)

(op2) – $\sigma_{DNUMBER > 5}$ (DEPARTMENT)

(op3) – $\sigma_{DNO = 5}$ (EMPLOYEE)

(op4) – $\sigma_{DNO = 5 \text{ AND } SALARY > 30000 \text{ AND } SEX = 'F'}$ (EMPLOYEE)

(op5) – $\sigma_{ESSN = '12345' \text{ AND } PNO = 10}$ (WORKS_ON)

Search Methods for Simple Selection – The following search methods are examples of some of the search algorithms that can be used to implement a select operation –

(a) Linear Search (brute force) – Retrieve every record in the file, and test whether its attribute values satisfy the selection condition.

(b) Binary Search – If the selection condition involves an equality comparison on a key attribute on which the file is ordered, binary search can be used. An example is op1 if SSN is the ordering attribute for the EMPLOYEE file.

(c) Using a Primary Index (or hash key) – If the selection condition involves an equality comparison on a key attribute with a primary index (or hash key), for instance SSN = '12345' in op1, use the primary index (or hash key) to retrieve the record. This condition retrieves at most a single record.

(d) Using a Primary Index to Retrieve Multiple Records – If the comparison condition is >, >=, <, or <= on a key field with a primary index, for instance DNUMBER > 5 is op2, use the index to find the record satisfying the corresponding equality condition (DNUMBER = 5), then retrieve all subsequent records in the (ordered) file. For the condition DNUMBER < 5 retrieve all the preceding records.

(e) Using a Clustering Index to Retrieve Multiple Records – If the selection condition involves an equality comparison on a non-key attribute with a clustering index, for instance DNO = 5 in op3, use the index to retrieve all the records satisfying the condition.

(f) Using a Secondary (B^+ tree) Index on an Equality Comparison – This search method retrieves a single record if the indexing field is a key or retrieves multiple records if the indexing field is not a key. This can also be used for comparisons involving >, >=, <, or <=.

Search Methods for Complex Selection – If a condition of a SELECT operation is conjunctive condition, i.e., if it is made up of several simple conditions connected with the AND logical connective as in op4, the DBMS uses the following additional methods to implement the operation –

(g) Conjunctive Selection Using an Individual Index – If an attribute involved in any single simple condition is the conjunctive condition has an access path that permits the use of one of the methods (b) to (f), use that condition to retrieve the records and then check whether each retrieved record satisfies the remaining simple conditions in the conjunctive condition.

(h) Conjunctive Selection Using a Composite Index – If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined fields, for instance, if an index has been created on the composite key (ESSN, PNO) of the WORKS_ON file for op5, we can use the index directly.

(i) Conjunctive Selection by Intersection of Record Pointers – If secondary indexes are available on more than one of the fields involved in simple conditions in the conjunctive condition, and if the indexes include record pointers rather than block pointers, then each index can be used to retrieve the set of record pointers that satisfy the individual condition. The intersection of these sets of record pointers gives the record pointers that satisfy the conjunctive condition, which are then used to retrieve those records directly. If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.

(ii) Join Operation – The join operation is the most time consuming operations in query processing. Here, the term join refers to an EQUIJOIN (or NATURAL JOIN). There are many ways to implement a two-way join, which is a join on two files. The algorithms we consider are for join operations of the form

$$R \bowtie_{A=B} S$$

where A and B are domain-compatible attributes of R and S, respectively. The most common techniques for performing such a join, using the following example operations –

(op1) – EMPLOYEE $\bowtie_{DNO=DNUMBER}$ DEPARTMENT

(op2) – DEPARTMENT $\bowtie_{MGRSSN=SSN}$ EMPLOYEE

are as follows –

(a) Nested-loop Join (brute force) – For each record t in R (outer loop), retrieve every record s from S (inner loop) and test whether the two records satisfy the join condition t[A] = s[B].

(b) **Single-loop Join (using an access structure to retrieve the matching records)** – If an index (or hash key) exists for one of the two join attributes, say B of S, retrieve each record t in R, one at a time (single loop), and then use the access structure to retrieve directly all matching records s from S that satisfy s[B] = t[A].

(c) **Sort-merge Join** – If the records of R and S are physically sorted by value of the join attributes A and B, respectively, we can implement the join in the most efficient way possible. Both files are scanned concurrently in order of the join attributes, matching the records that have the same values for A and B. If the file are not sorted, they may be sorted first by using external sorting. In this method, pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file. Fig. 3.17 shows the sort-merge join algorithm. We use R(i) to refer to the ith record in R. A variation of the sort-merge join can be used when secondary indexes exist on both join attributes. The indexes provide the ability to access the records in order of the join attributes, but the records themselves are physically scattered all over the file blocks, so this method may be quite inefficient, as every record access may involve accessing a different disk block.

```
sort the tuples in R on attribute A; (*assume R has n tuples (records) *)
sort the tuples in S on attribute B; (*assume S has m tuples (records) *)
set i ← 1, j ← 1;
while (i ≤ n) and (j ≤ m)
do{   if R(i)[A] > S(j)[B]
          then     set j ← j+1
      elseif R(i)[A] < S(j)[B]
          then     set i ← i + 1
          else  {  (* R(i)[A] = S(j)[B], so we output a matched tuple*)
                   output the combined tuple <R(i), S(j)> to T;
                        (*output other tuples that match R(i), if any*)
                        set l ← j+1;
                        while (l ≤ m) and (R(i)[A] = S(l)[B])
                        do     {             output the combined tuple
                                             <R(i), S(l)> to T; set l ← l+1
                               }
                        (*output other tuples that match S(j), if any*)
                        set k ← i+1;
                        while(k ≤ n) and (R(k)[A] = S(j)[B])
                        do     {             output the combined tuple
                                             <R(k), S(j)>toT; set k ← k+1
                               }
                        set i ← k, j ← l
                }
}
```

Fig. 3.17 Implementing the Operation $T \leftarrow R \bowtie_{A=B} S$

(d) **Hash-join** – The records of files R and S are both hashed to the same hash file, using the same hashing function on the join attributes A of R and B of S as hash keys. First, a single pass through the file with fewer records (say, R) hashes its records to the hash file buckets, this is called the ***partitioning phase***, since the records of R are partitioned into the hash buckets. In the second phase, called the ***probing phase***, a single pass through the other file (S) then hashes each of its records to probe the appropriate bucket, and that record is combined with all matching records from R in that bucket. This simplified description of hash-join assumes that the smaller of the two files fits entirely into memory buckets after the first phase.

(iii) Project Operation – A project operation $\pi_{<attribute\ list>}(R)$ is simple to implement if <attribute list> includes a key of relation R, because in this case the result of the operation will have the same number of tuples as R, but with only the values for the attributes in <attribute list> in each tuple. If <attribute list> does not include a key of R, duplicate tuples must be eliminated. This is done by sorting the result of the operation and then eliminating duplicate tuples, which appear consecutively after sorting. A sketch of the algorithm is shown in fig. 3.18. Hashing can also be used to eliminate duplicates as each record is hashed and inserted into a bucket of the hash file in memory, it is checked against those already in the bucket. It is a duplicate, it is not inserted.

```
create a tuple t[<attribute list>] in T' for each tuple t in R;
    (*T' contains the projection result before duplicate elimination*)
If <attribute list> includes a key of R
    then T←T'
    else  {     sort the tuples in T';
                set i←1, j←2;
                while i≤n
                      do  {    output the tuple T'[i] to T;
                               while T[i] = T'[j] and j≤n do j←j+1; (*eliminate
                               duplicates*) i←j; j←i+1
                          }
          }
(* T contains the projection result after duplicate elimination *)
```

Fig. 3.18 Implementing the Operation $T \leftarrow \pi_{<attribute\ list>}(R)$

Q.42. How does a DBMS represent a relational query evaluation plan ?
(R.G.P.V., June 2010, Dec. 2011)

Ans. An execution (evaluation) plan for a relational algebra expression represented as a query tree includes information about the access methods available for each relation as well as the algorithms to be used in computing the relational operators represented in the tree. As a simple example, consider query Q_1 as given below –

$$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{DNAME='RESEARCH'}}(\text{DEPARTMENT})$$

$$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{DNUMBER=DNO}} \text{EMPLOYEE})$$

$$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, ADDRESS}}(\text{RESEARCH\_EMPS})$$

whose corresponding relation algebra expression is

$$\pi_{\text{FNAME, LNAME, ADDRESS}}(\sigma_{\text{DNAME = 'RESEARCH'}}(\text{DEPARTMENT}) \bowtie_{\text{DNUMBER=DNO}} \text{EMPLOYEE})$$

The query tree is shown in fig. 3.19. To convert this into an execution plan, the optimizer might choose an index search for the SELECT operation (assuming one exists), a table scan as access method for EMPLOYEE, a nested-loop join algorithm for the join, and a scan of the JOIN result for the PROJECT operator. In addition, the approach taken for executing the query may specify a materialized or a pipelined evaluation.

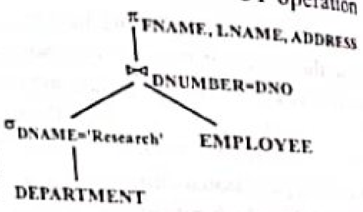


Fig. 3.19 A Query Tree for Query Q_1

With **materialized evaluation,** the result of an operation is stored as a temporary relation (that is, the result is *physically materialized*). For example, the join operation can be computed and the entire result stored as a temporary relation, which is then read as input by the algorithm that computes the PROJECT operation, which would produce the query result table. On the other hand, with **pipelined evaluation,** as the resulting tuples of an operation are produced, they are forwarded directly to the next operation in the query sequence. For example, as the selected tuples from DEPARTMENT are produced by the SELECT operation, they are placed in a buffer; the JOIN operation algorithm would then consume the tuples from the buffer, and those tuples that result from the JOIN operation are pipelined to the projection operation algorithm. The advantage of pipelining is the cost savings in not having to write the intermediate results to disk and not having to read them back for the next operation.

Q.43. What is meant by term heuristic optimization ? Discuss the main heuristic that is applied during query optimization. What is cost based optimization ?

(R.G.P.V., Nov. 2018)

Ans. The heuristic optimization is a technique that apply heuristic rules to modify the internal representation of a query, which is usually in the form of a query tree or a query graph data structure, to improve its expected performance. The parser of a high-level query first generates an initial internal representation, which is then optimized according to heuristic rules. Following that, a query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.

The main heuristic is to apply first the operations that reduce the size of intermediate results. This includes performing as early as possible SELECT operations to reduce the number of tuples and PROJECT operations to reduce the number of attributes. This is done by moving SELECT and PROJECT operations as far down the tree as possible. In addition, the SELECT and JOIN operations that are most restrictive, i.e. result in relations with the fewest tuples or with the smallest absolute size, should be executed before other similar operations. This is done by reordering the leaf nodes of the tree among themselves while avoiding cartesian products, and adjusting the rest of the tree appropriately.

A query optimizer should also estimate and compare the costs of executing a query using different execution strategies and should choose the strategy with the lowest cost estimate. For this approach to work, accurate cost estimates are required so that different strategies are compared fairly and realistically. In addition, we must limit the number of execution strategies to be considered, otherwise too much time will be spent making cost estimates for the many possible execution strategies. This approach is called as cost-based query optimization.

Q.44. Discuss the cost components for a cost function that is used to estimate query execution cost. Which cost components are used most after as the basis for cost functions ?

Ans. The cost of executing a query involves the following components –

(i) ***Access Cost to Secondary Storage*** – This is the cost of searching for reading, and writing data blocks that reside on secondary storage, mainly on disk. The cost of searching for records in a file depends on 4he type of access structures on that file such as ordering, hashing, and primary or secondary indexes. In addition, the factors such as whether the file blocks, are allocated contiguously on the same disk cylinder or scattered on the disk affect the access cost.

(ii) ***Storage Cost*** – This is the cost of storing any intermediate files that are generated by an execution strategy for the query.

(iii) ***Memory Usage Cost*** – This is the cost pertaining to the number of memory buffers required during query execution.

(iv) ***Computation Cost*** – This is the cost of performing in memory operations on the data buffers during query execution. These operations are searching for and sorting records, merging records for a join, and performing computations on field values.

Q.3. Explain durability in transaction. ***(R.G.P.V., Dec. 2016)***

Ans. Refer to Q.2 (iv).

Q.4. What do you mean by consistency in transaction ? ***(R.G.P.V., Dec. 2016)***

Ans. Refer to Q.2 (ii).

Q.5. Explain various transaction states with their description. Also discuss its state diagram. ***(R.G.P.V., Dec. 2013)***

Ans. A transaction must be in one of the following states –

(i) Active – It is the initial state of the transaction. The transaction stays in this state while it is executing.

(ii) Partially Committed – The transaction enters in this state after the final statement has been executed.

(iii) Failed – The transaction enters in this state if the normal execution can no longer proceed.

(iv) Aborted – The transaction enters in this state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

(v) Committed – The transaction enters in this state after successful completion.

Fig. 4.1 shows a state transition diagram that describes how a transaction moves through its execution states. A transaction initially goes into an active state and starts execution, where it can issue read and write operation. When the transaction ends, it moves to the partially committed state. At this point, some recovery protocols need to ensure that a system failure will not result in an ability to record the changes of the transaction permanently. Once the check is successful, the transaction is said to have reached its commit point and enters the commit state. Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.

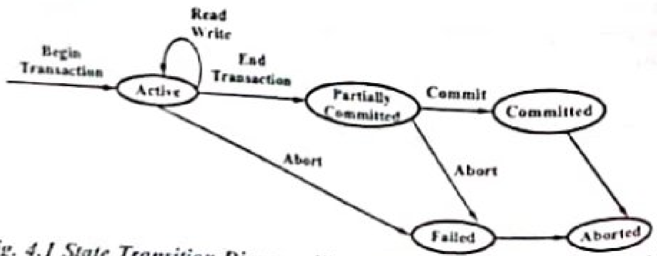


Fig. 4.1 State Transition Diagram Illustrating the States for Transaction Execution

However, a transaction can go to the failed state if one of the checks fails or the transaction is aborted during its active state. The transaction may have to be rolled back to undo the effect of its write operations on the database. The terminated state responds to the transaction leaving the system. Failed or aborted transactions may be restarted later either automatically or after being resubmitted by the user as brand new transactions.

Q.6. Write short note on transaction processing. ***(R.G.P.V., June 2004, 2005, Dec. 2005, June 2007)***

Ans. A transaction is a logical unit of database processing that includes one or more database access operations such as insertion, deletion, modification, or retrieval. The database operations that form a transaction can either be embedded within an application program or they can be specified interactively through a high-level query language such as SQL, where it is delimited by statements of the form ***begin transaction*** and ***end transaction.*** In this case, all database access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries.

Transactions access data using two operations –

(i) Read (X) – It transfers the data item X from the database to a local buffer belonging to the transaction that executed the read operation.

(ii) Write (X) – It transfers the data item X from the local buffer of the transaction that executed the write back to the database.

Fig. 4.2 shows examples of two very simple transactions. The read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that the transaction writes. For example, the read-set of T_1 in fig. 4.2 is {X, Y} and its write-set is also {X, Y}.

```
      T1                T2
 ------------      ------------
  read (X);         read (X);
  X := X – N;       X := X + M
  write (X);        write (X);
  read (Y);
  Y := Y + N;
  write (Y);
```

(a) Transaction T_1 ***(b) Transaction T_2***

Fig. 4.2 Two Sample Transactions

Q.7. Write short note on transaction control statements (TCS). ***(R.G.P.V., June 2010)***

Ans. A transaction is an atomic unit of work that is either completed in its entirety or not done at all. Transaction control statements are as follows –

(i) BEGIN_TRANSACTION – This marks the beginning of transaction execution.

(ii) READ or WRITE – These specify read or write operations on the database items that are executed as part of a transaction.

(iii) *END_TRANSACTION* – This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability or for some other reasons.

(iv) *COMMIT-TRANSACTION* – This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.

(v) *ROLLBACK (or ABORT)* – This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.

Q.8. Transaction usually cannot be nested inside one another. Why not? ***(R.G.P.V., Dec. 2013)***

Ans. Such a feature of transaction will create confliction with the objective of transaction atomicity. Suppose what would happen if transaction B were nested inside transaction A and the following sequence of events occurred -

```
BEGIN TRANSACTION (A);
   ..............
     BEGIN TRANSACTION (B);
     transaction B updates tuple t;
     COMMIT (B);
     ..............
     ..............
     ..............
ROLLBACK (A);
```

If tuple t is restored to its pre-A value at this point, then B's COMMIT was not in fact a COMMIT at all. However, if B's COMMIT was genuine then tuple t cannot be restored to its pre-A value, and hence A's ROLLBACK cannot be performed.

By observing this, we can say that transactions cannot be nested and a program can execute a BEGIN TRANSACTION operation only when it has no transaction currently executing.

Q.9. Explain serializability. Also, discuss the kinds of serializability. ***(R.G.P.V., Dec. 2009)***

Or

Define serializability and differentiate conflict and view serializability. ***(R.G.P.V., June 2010)***

Ans. ***A serializable schedule*** over a set S of committed transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over S. That is, the database instance that results from executing the transactions in some serial order.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
R(B)	
W(B)	
	R(B)
	W(B)
	commit
commit	

Fig. 4.3 A Serializable Schedule

T1	T2
	R(A)
	W(A)
R(A)	
	R(B)
	W(B)
W(A)	
R(B)	
W(B)	
	commit
commit	

Fig. 4.4 Another Serializable Schedule

For example, the schedule shown in fig. 4.3 is serializable. Even though, the actions of T1 and T2 are interleaved, the result of this schedule is equivalent to running T1 (in its entirety) and then running T2. Intuitively, T1's read and write of B is not influenced by T2's actions on A, and the net effect is the same if these actions are swapped to obtain the serial schedule T1, T2.

Executing transactions serially in different orders may produce different results, but all are presumed to be acceptable. The DBMS makes no guarantees about which of them will be the outcome of an interleaved execution. To see this, the two example transactions from fig. 4.3 can be interleaved as shown in fig. 4.4. This schedule, also serializable, is equivalent to the serial schedule T2, T1.

Kinds of Serializability – Serializability can be of following two types –

(i) ***Conflict Serializability*** – Consider a schedule S in which there are two consecutive instructions I_i and I_j, of transactions T_i and T_j respectively (i ≠ j). If I_i and I_j refer to different data items, then I_i and I_j can be swapped without affecting the results of any instruction in the schedule. However, if I_i and I_j refer to the same data item Q, then the order of steps may matter. There are four cases to be considered while dealing with only read and write instructions –

(a) I_i = read (Q), I_j = read(Q). The order of I_i and I_j does not matter, since the same value of Q is read by T_i and T_j, regardless of the order.

(b) I_i = read (Q), I_j = write (Q). If I_i comes before I_j, then T_i does not read the value of Q that is written by T_j in instruction. If I_j comes before I_i, then T_i reads the value of Q that is written by T_j. Thus, the order of I_i and I_j matters.

(c) I_i = write (Q), I_j = read (Q). The order of I_i and I_j matters as in case (b).

(d) I_i = write (Q), I_j = write (Q). The order of these instructions does not affect either T_i or T_j, since both are write operations. However , the value obtained by the next read (Q) instruction of S is affected, since the result of only the latter of the two write instructions is preserved in the database. If there is no other write (Q) instruction after I_i and I_j in S, then the order of I_i and I_j directly affects the final value of Q in the database state that results from schedule S.

We say that I_i and I_j conflict if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation.

For example, consider the schedule 1 in fig. 4.5. The write (A) instruction of T_1 conflicts with the read (A) instruction of T_2. However, the write (A) instruction of T_2 does not conflict with the read (B) instruction of T_1, because the two instructions access different data items.

Let I_i and I_j be consecutive instructions of a schedule S. If I_i and I_j are instructions of different transactions and I_i and I_j do not conflict, the order of I_i and I_j can be swapped to produce a new schedule S'. It is expected that S is equivalent to S', since all instructions appear in the same order in both schedules except for I_i and I_j, whose order does not matter.

Since the write (A) instruction of T_2 in schedule 1 does not conflict with the read (B) instruction of T_1, we can swap these instructions to generate an equivalent schedule, schedule 2 in fig. 4.6. Schedules 1 and 2 both produce the same final system state.

T_1	T_2
Read (A) Write (A)	
	Read (A) Write (A)
Read (B) Write (B)	
	Read (B) Write (B)

Fig. 4.5 Schedule 1 – Showing Only the Read and Write Instructions

T_1	T_2
Read (A) Write (A)	
	Read (A)
Read (B) Write (B)	Write (A)
	Read (B) Write (B)

Fig. 4.6 Schedule 2 – Schedule 1 after Swapping of a Pair of Instruction

Now, continue to swap nonconflicting instructions –

(a) Swap the read (B) instruction of T_1 with the read (A) instruction of T_2.

(b) Swap the write (B) instruction of T_1 with the write (A) instruction of T_2.

(c) Swap the write (B) instruction of T_1 with the read (A) instruction of T_2.

The final result of these swaps is schedule 3 as shown in fig. 4.7. It is a serial schedule. Thus, schedule 1 is equivalent to a serial schedule. This equivalence implies that, regardless of the initial system state, schedule 1 will produce the same final state as will some serial schedule.

T_1	T_2
Read (A) Write (A) Read (B)	
	Read (A) Write (A) Read (B) Write (B)

Fig. 4.7 Schedule 3 – A Serial Schedule Equivalent to Schedule 1

T_3	T_4
Read (Q)	
	Write (Q)
Write (Q)	

Fig. 4.8 Schedule 4

If a schedule S can be transformed into a schedule S' by a series of swaps of nonconflicting instructions, the S and S' are ***conflict equivalent.*** The concept of conflict equivalence leads to the concept of conflict serializability. A schedule S is conflict serializable if it is conflict equivalent to a serial schedule.

Fig. 4.8 shows schedule 4, which consists of read and write operations of transactions T_3 and T_4. This schedule is not conflict serializable, since it is not equivalent to either the serial schedule $<T_3, T_4>$ or the serial schedule $<T_4, T_3>$.

It is possible that two schedules produce the same result, but are not conflict equivalent.

(ii) ***View Serializability*** – Consider two schedules S and S' with the same set of transactions participating in both schedules. The schedules S and S' are said to be view equivalent if following three conditions are met –

(a) For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i in schedule S' must read the initial value of Q.

(b) For each data item Q, if transaction T_i executes read (Q) in schedule S, and if that value was produced by a write (Q) operation executed by transaction T_j, then the read (Q) operation of transaction T_i in schedule S' must also read the value of Q that was produced by the same write (Q) operation of transaction T_j.

(c) For each data item Q, the transaction (if any) that performs the final write (Q). Operation in schedule S must perform the final write (Q) operation in schedule S'.

Conditions (a) and (b) ensure that each transaction reads the same values in both schedules and, therefore, performs the same computation. Condition (c), coupled with conditions (a) and (b), ensures that both schedules result in the same final system state.

The concept of view equivalence leads to the concept of view serializability. A schedule S is view serializable if it is view equivalent to a serial schedule.

For example, fig. 4.9 shows schedule 5, which is view serializable. It is view equivalent to the serial schedule <T_3, T_4, T_6>, since the one read (Q) instruction reads the initial value of Q, and T_6 performs the final write of Q.

Every conflict-serializable schedule is view serializable, but its vice-versa is not true.

T_3	T_4	T_6
Read (Q)		
	Write (Q)	
Write (Q)		
		Write (Q)

Fig. 4.9 Schedule 5 – A View Serializable Schedule

Q.10. Write short note on serializability.

(R.G.P.V., June 2006, Nov. 2018)

Or

What is serializability ? ***(R.G.P.V., June 2016)***

Ans. Refer to Q.9.

Q.11. What is serializability ? Write an algorithm for testing conflict serializability of a schedule. ***(R.G.P.V., Dec. 2006)***

Ans. Serializability – Refer to Q.9.

Algorithm for Testing Conflict Serializability of a Schedule –

Testing conflict serializability of a schedule S.

(i) For each transaction T_i participating in schedule S, create a node labeled T_i in the precedence graph.

(ii) For each case in S where T_j executes a read_item (X) after T_i executes a write_item (X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.

(iii) For each case in S where T_j executes a write_item(X) after T_i executes a read_item (X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.

(iv) For each case in S where T_j executes a write_item (X) after T_i executes a write_item (X), create an edge ($T_i \rightarrow T_j$) in the precedence graph.

(v) The schedule S is serializable if and only if the precedence graph has no cycle.

Q.12. Explain ACID properties. Explain serializability of schedule. ***(R.G.P.V., Dec. 2016)***

Ans. Refer to Q.2 and Q.9.

Q.13. What is the utility of normalization and various normal forms ? Explain the concept of transaction atomicity and serializability. ***(R.G.P.V., Dec. 2012)***

Ans. **Utility of Normalization and Various Normal Forms** – Refer to Q.1 (Unit-III) and Q.2 (Unit-III).

Transaction Atomicity – Refer to Q.2 (i).

Serializability – Refer to Q.9.

Q.14. Discuss the various types of serializability with a suitable example. ***(R.G.P.V. Dec. 2011)***

Ans. Refer to Q.9.

Q.15. Explain recoverable schedule. ***(R.G.P.V., Dec. 2010)***

Ans. A ***recoverable schedule*** is one where, for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j.

If a transaction T_i fails due to any reason, we need to undo the effect of this transaction to ensure the atomicity property of the transaction. In a system that allows concurrent execution, it is also necessary to ensure that any transaction T_j that is dependent on T_i (i.e. T_j has read data written by T_i) is also aborted. Consider schedule 6 in fig. 4.10, in which transaction T_9 performs only one transaction read (A). Suppose that the system allows T_9 to commit immediately after executing the read (A) instruction. Thus, T_9 commits before T_8 does. Now, suppose that T_8 fails before it commits. Since T_9 has read the value of data item A written by T_8, we must abort T_9 to ensure transaction atomicity. However, T_9 has already committed and cannot be aborted. Thus, there is situation from where it is impossible to recover correctly from the failure of T_8. Thus, schedule 6 is a nonrecoverable schedule, which should not be allowed. Most database systems require that all schedules be recoverable.

T_8	T_9
Read (A) Write (A)	
	Read (A)
Read (B)	

Fig. 4.10 Schedule 6

Q.16. Explain cascadeless schedule. ***(R.G.P.V., Dec. 2010)***

Ans. A cascadeless schedule is one where, for each pair of transaction T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j.

Even if a schedule is recoverable, to recover correctly from the failure of a transaction T_i, several transactions are to be rolled back. Such situations occur if transactions have read data written by T_i. For example, consider the partial schedule 7 of fig. 4.11. Transaction T_{10} writes a value of A that is read by transaction T_{11}. Transaction 11 writes a value of A that is read by transaction T_{12}. Suppose that, at this point, T_{10} fails. T_{10} must be rolled back. Since T_{11} is dependent on T_{10}, T_{11} must be rolled back. Since T_{12} is dependent on T_{11}, T_{12} must be rolled back. This phenomenon, in which a single transaction failure leads to a series of transaction rollback, is called cascading rollback.

T_{10}	T_{11}	T_{12}
Read (A) Read (B) Write (A)		

	Read(A)
	Write(A)	

		Read A

Fig. 4.11 Schedule 7

Cascading rollback is undesirable, since it leads to the undoing of a significant amount of work. It is desirable to restrict the schedules to those where cascading rollbacks cannot occur.

Q.17. What do you mean by schedule in the context of concurrent execution of transactions in RDBMS ? What is serializable schedule ? ***(R.G.P.V., Dec. 2011)***

Ans. A schedule (or history) S of n transactions $T_1, T_2, \ldots, T_n$ is an ordering of the operations of the transactions subject to the constraint that for each transaction T_i that participates in S, the operations of T_i in S must appear in the same order in which they occur in T_i. However, that operations from other transactions T_j can be interleaved with the operations of T_i in S.

Two operations in a schedule are said to ***conflict*** if they satisfy all three of the following conditions –

(i) They belong to different transactions.

(ii) They access the same item X.

(iii) At least one of the operations is a write_item (X).

A schedule S of n transactions $T_1, T_2, \ldots, T_n$ is said to be a ***complete*** schedule if the following conditions hold –

(i) The operations in S are exactly those operations in $T_1, T_2, \ldots, T_n$, including a commit or abort operation as the last operation for each transaction in the schedule.

(ii) For any pair of operations from the same transaction T_i, their order of appearance in S is the same as their order of appearance in T_i.

(iii) For any two conflicting operations, one of the two must occur before the other in the schedule.

A schedule is called ***serial*** if the operations of each transaction are executed consecutively, without any interleaved operations from other transaction.

A schedule is called ***interleaved nonserial*** if each sequence interleaves operations from two transactions.

Fig. 4.12 shows four schedules. Schedules A and B are serial because entire transactions are performed in serial order – T_1 and then T_2 in fig. 4.12 (a), and T_2 and then T_1 in fig. 4.12 (b). Schedules C and D in fig. 4.12 (c) are nonserial.

Formally, a schedule S is serial if for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule. Otherwise, the schedule is called non-serial. Hence, in a serial schedule, only one transaction at a time is active. No interleaving occurs in a serial schedule.

The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules. A schedule S of n transactions is ***serializable*** if it is equivalent to some serial schedule of the same n transactions. Each serial schedule consists of a sequence of instructions from various transactions, where the instructions belonging to one single transaction appear together in that schedule. Thus, there are n ! possible serial schedules of n transactions and many more possible nonserial schedules. Saying that a nonserial schedule is serializable is equivalent to saying that it is correct, because it is equivalent to a serial schedule, which is considered correct.

T_1	T_2

read_item(X);
X:=X-N;
write_item(X);
read_item(Y);
Y:=Y+N;
write_item(Y);	

	read_item(X)
	X:=X+M;
	write_item(X);

Schedule A

(a) Serial Schedule
A – T_1 followed by T_2

Time

T_1	T_2

	read_item(X)
	X:=X+M;
	write_item(X);

read_item(X);
X:=X-N;
write_item(X);
read_item(Y);
Y:=Y+N;
write_item(Y);	

Schedule B

(b) Serial Schedule
B – T_2 Followed by T_1

T_1	T_2

read_item(X);
X:=X-N;	

	read_item(X)
	X:=X+M;

write_item(X);
read_item(Y);	

	write_item(X);

Y:=Y+N;
write_item(Y);	

Schedule C

T_1	T_2

read_item(X);
X:=X-N;
write_item(X);	

	read_item(X)
	X:=X+M;
	write_item(X);

read_item(Y);
Y:=Y+N;
write_item(Y);	

Schedule D

(c) Two Nonserial Schedules C and D with Interleaving of Operations

Fig. 4.12 Examples of Serial and Nonserial Schedules Involving Transactions T_1 and T_2

Q.18. What is a schedule ? What is an interleaved schedule ? How is schedule related to the term serializability ? ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.17.

Q.19. Describe different types of transaction failures. What is meant by catastrophic failure ? ***(R.G.P.V., Dec. 2006)***

Ans. Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution –

(i) A Computer Failure (System Crash) – A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures. For example, main memory failure.

(ii) A Transaction or System Error – Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition the user may interrupt the transaction during its execution.

(iii) Local Errors or Exception Conditions Detected by the Transaction – During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. An exception condition such as insufficient account balance in a blanking database, may cause a transaction, such as a fund withdrawal, to be cancelled. This exception should be programmed in the transaction itself, and hence would not be considered a failure.

(iv) Concurrency Control Enforcement – The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

(v) Disk Failure – Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

(vi) Physical Problems and Catastrophes – This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting a wrong tape by the operator.

Failures of types 1, 2, 3 and 4 are more common than those of types 5 or 6. Whenever a failure of type 1 through 4 occurs, the system must keep sufficient information to recover from the failure. Disk failure or other catastrophic failures of type 5 or 6 do not happen frequently; if they do occur, recovery is a major task.

Q.20. What are the recovery implications of physical writing database buffers at COMMIT ? ***(R.G.P.V., Dec. 2013)***

Ans. The recovery implications of physical writing database buffers at COMMIT express that Redo is never necessary following system failure.

Q.21. Write short note on recovery system in DBMS. ***(R.G.P.V., Dec. 2004)***

Or

Why we need to do recovery in DBMS ? Explain various method use for recovery. ***(R.G.P.V., May 2018)***

Ans. Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database or the transaction has no effect whatsoever on the database or on any other transactions. The DBMS must not permit some operations of a transaction T to be applied to the database while other operations of T are not. This may happen if a transaction fails after executing some of its operations but before executing all of them. Recovery from failures means that the database is restored to the most consistent state just before the time of failure. To do this, the system must keep information about the changes that were applied to data items by the various transactions. This information is kept in *system log.*

Informally, a typical strategy for recovery may be summarized as follows –

(i) If there is extensive damage to a wide portion of the database due to catastrophic failure, such a disk crash, the recovery method stores a past copy of the database that was backed up to archival storage and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed up log, up to the time of failure.

(ii) When the database is not physically damaged but has become inconsistent due to noncatastrophic failures. The strategy is to reverse any changes that caused the inconsistency by undoing some operations.

There are two main techniques for recovery from noncatastrophic transaction failures (i) deferred update and (ii) immediate update.

Q.22. State with examples desirable properties of a transaction. What is the system log used for ? ***(R.G.P.V., June 2010)***

Ans. **Properties of a Transaction** – Refer to Q.2.

System Log – The log is a structure used for recording database modifications. It is a sequence of log records, recording all the update activities in the database. We use this log to recover form failures and to bring back the database to the consistent state. An update log record contains following fields –

(i) ***Transaction Identifier*** – It uniquely identifies the transaction.

(ii) ***Data Item Identifier*** – It uniquely identifies the data to be used.

(iii) ***Old Value*** – It is the value of the data item before to the write.

(iv) ***New Value*** – It is the value of the data item, that will be held by data item after the write.

There are many types of log records, we denote the various types of log records as follows –

(i) < T_i **start** > Transaction T_i has started.

(ii) < T_i, X_j, V_1, V_2 > transaction T_i has performed a write on data item X_j. X_j has value V_1 before the write and will have value V_2 after the write.

(iii) < T_i **commit** > Transaction T_i has committed.

(iv) < T_i **abort** > Transaction T_i has aborted.

Whenever a transaction performs write operation, database is modified only when log record for that write is created. And logs may or may not contain the record of reading of data-item. It is so because, reading the data item does not affect the consistency of the database and is nowhere helpful in recovery mechanism.

Once a log record exists, we can output the modifications of the database, if that is desirable. Also we have the ability to undo a modification that has already been output to the database. We undo it by using the old-value field in the log records.

Two techniques for using the log to ensure transaction atomicity despite failures are as follows –

(i) Deferred database modification

(ii) Immediate database modification.

Q.23. What is a transaction in database ? Discuss the property of transaction along with various states of transaction. Briefly discuss purpose of log file in database recovery. ***(R.G.P.V., June 2011)***

Ans. Transaction – Refer to Q.1.

Property of Transaction – Refer to Q.2.

States of Transaction – Refer to Q.5.

Log File – Refer to Q.22.

Q.24. State the write-ahead log rule. Why is the rule necessary ? ***(R.G.P.V., Dec. 2013)***

Ans. Generally, transactions are considered as the unit of work, however. they are also specified as the unit of recovery. Thus, when a transaction successfully commits then the system will ensure that its updates will be permanently installed in the database, even if the system crashes the next moment. Generally, it is existed, suppose that the system may crash after the commit has been performed but before the updates have been physically written to the database. They may still be looking for their chance in a main memory buffer and so be disappeared at the time of the crash. Even if that occurs, the system's restart procedure will still install those updates in the database; it is able to find out the values to be written by inspecting the relevant entries in the log. It follows the write-ahead log rule which expresses that the log must be physically written before commit processing can complete. Thus, any transactions that completed successfully will be recovered by the restart procedure. These transactions did not manage to achieve their updates physically written prior to the crash. Hence, transactions are indeed the unit of recovery.

Q.25. What are checkpoints, and why are they important ?

Or

Explain checkpoint record. ***(R.G.P.V., Dec. 2013)***

Ans. A checkpoint is a type of entry in the log. A [checkpoint] record is written into the log periodically at that point when the system writes out to the database on disk all DBMS buffers that have been modified. As a consequence of this, all transactions that have their [commit, T] entries in the log before a [checkpoint] entry do not need do have their WRITE operations redone in case of a system crash, since all their updates will be recorded in the database on disk during checkpointing.

The recovery manager of a DBMS must decide at what intervals to take a checkpoint. The interval may be measured in time-say, every m minutes or in the number t of committed transactions since the last checkpoint, where the values of m or t are system parameters. Taking a checkpoint consists of the following actions –

(i) Suspend execution of transactions temporarily.

(ii) Force-write all main memory buffers that have been modified to disk.

(iii) Write a [checkpoint] record to the log, and force-write the log to disk.

(iv) Resume executing transactions.

Q.26. Explain the recovery process after system failure using checkpoint. ***(R.G.P.V., June 2008, Dec. 2008, 2010)***

Ans. Recovery using the log records contains two major problems –

(i) The search process is time consuming, as one might conclude that recovery requires just the scanning of log as a whole for recent transactions.

(ii) Most of the transactions that according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.

To reduce these types of overheads, we introduce ***checkpoints.***

During execution, the DBMS maintains the log, but periodically performs checkpoints consisting of the following actions –

(i) Temporarily halting the initiation of new transaction until all the active ones are committed or aborted.

(ii) Making a backup copy of database.

(iii) Writing all log records currently residing in primary memory to stable storage.

(iv) Appending to the end to the log a record indicating that a checkpoint has occurred, then writing it to disk storage.

In this, after a failure has occurred, the recovery scheme examines the log to determine the most recent transaction T_i that started executing before the most recent checkpoint took place.

For this, we search log backward from the end of log until it finds the first <checkpoint> record, then it continues the search backward until it finds the next <T_i start> record.

This record identifies a transaction T_i. For example, consider the set of transactions $\{T_0, T_1, ..., T_{100}\}$ executed in the order of the subscripts. Suppose that the most recent checkpoint took place during the execution of transaction T_{80}. Thus, only transactions $T_{80}, T_{81}, ..., T_{100}$ need to be considered during the recovery scheme.

Each of them need to be redone if it has committed otherwise, it needs to be undone.

Q.27. Compare and contrast the features of log based recovery mechanism versus check pointing based recovery. Suggest applications where you prefer log based recovery scheme over check pointing. Give an example of check pointing based recovery scheme. Discuss the recoverable schedule also.

(R.G.P.V., Dec. 2011)

Ans. **Log-based and Checkpoint Recovery Mechanism** – Refer to Q.22 and Q.26.

Recoverable Schedule – Refer to Q.15.

Q.28. Explain the purpose of checkpoint mechanism. How often should checkpoints be performed.

(R.G.P.V., Dec. 2017)

Ans. Refer to Q.25 and Q.26.

Q.29. What is deadlock ? Discuss deadlock prevention and deadlock detection.

Ans. A system is in a deadlock state if each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set. Hence, each transaction in the set is on a waiting queue, waiting for one of the other transactions in the set to release the lock on an item.

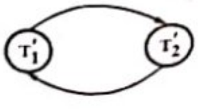
	T'_1	T'_2

Time ↓	Read_lock(Y);
	Read_item(Y);	

		Read_lock(X);
		Read_item(X);

	Write_lock(X);	
		Write_lock(Y);

(a) A Partial Schedule of T'_1 and T'_2 in Deadlock State



(b) A Wait-for Graph for the Partial Schedule in (a)

Fig. 4.13 Illustration of Deadlock Problem

Fig. 4.13 shows that two transactions T'_1 and T'_2 are deadlocked in a partial schedule T_1 is on the waiting queue for X , which is locked by T'_2 and T'_2 is on the waiting queue for Y which is locked by T'_1.

Deadlock Prevention Protocols – The way to prevent deadlock is to use a deadlock prevention protocol. One deadlock prevention protocol requires that every transaction locks all the items it needs in advance i.e., before execution. If any of the items cannot be locked, none of the items are locked. Rather, the transaction waits and then tries again to lock all the items it needs. This solution has two main disadvantages –

(a) It is very hard to predict, before the transaction begins, what data items need to be locked.

(b) Data-item utilization may be very low, since many of the data items may be locked but unused for a long time.

Another approach is to impose an ordering of all data-items, and to require that a transaction lock data item in a sequence consistent with the ordering.

The second approach for preventing deadlocks is to use preemption and transaction rollbacks. In preemption, when a transaction T_2 requests a lock that transaction T_1 holds, the lock granted to T_1 may be preempted by rolling back of T_1, and granting of the lock to T_2. To control the preemption, we assign a unique timestamp to each transaction. The system uses these timestamps to decide whether a transaction should wait or roll back. Locking is still used for concurrency control. If a transaction is rolled back, it retains its old timestamp when restarted. Two different deadlock prevention schemes using timestamps are as follows –

(a) **Wait-die** – It is a nonpreemptive technique. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than of T_j (i.e., T_i is older then T_j). Otherwise, T_i is rolled back (dies).

(b) **Wound-wait** – This is a preemptive technique. It is a counterpart to the wait-die scheme. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j (i.e., T_i is younger than T_j). Otherwise, T_j is rolled back (T_j is wounded by T_i).

Deadlock Detection and Timeouts – A more practical approach to dealing with deadlock is deadlock detection, where the system checks if a state of deadlock actually exists. A simple way to detect a state of deadlock is for the system to construct and maintain a ***wait-for-graph***. One node is created in the wait-for-graph for each transaction that is currently executing. Whenever a transaction T_i is waiting to lock an item X that is currently locked by a transaction T_j, a directed edge $T_i \rightarrow T_j$ is created in the wait-for-graph. When T_j releases the lock (s) on the items that T_i was waiting for, the directed edge is dropped from the wait-for-graph.

A deadlock exists in the system if and only if the wait-for-graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks the system needs to maintain the wait-for-graph, and periodically to invoke an algorithm that searches for a cycle.

Another simple scheme to deal with deadlock is the use of ***timeouts.*** This method is practical because of its low overhead and simplicity. In this method, if a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it regardless of whether a deadlock actually exits or not.

Q.30. Explain the term deadlock detection and recovery.

(R.G.P.V., Dec. 2011)

Ans. **Deadlock Detection** – Refer to Q.29.

Deadlock Recovery – When a detection algorithm determines that a deadlock exist, the system must recover from the deadlock. To recover from deadlock, the cycles in the wait-for graph must be broken. The common method of doing this is to roll back one or more transaction in the cycles until the system exhibits no further deadlock situation. For the selection of the transaction to be rolled back, three actions need to be taken –

(i) ***Selection of a Victim*** – Given a set of deadlocked transactions, we must determine which transaction to roll back to break the deadlock. It is preferable to roll back a transaction that has just started or has not modified any data-item, rather than one that has run for a considerable time and has modified many data-items.

(ii) ***Rollback*** – Once we have decided which particular transaction must be rolled back, we must determine how far this transaction should be rolled back. The simplest solution is a total rollback–abort the transaction and then restart it. Although it is more effective to roll back the transaction only as far as necessary to break the deadlock. Such partial rollback needs the system to maintain additional information about the state of all the running transactions. The recovery mechanism must be capable of performing such partial rollbacks.

(iii) ***Starvation*** – In a system where the selection of victims depends on cost factors, it may happen that the same transaction is always picked as a victim. As a result, this transaction never completes its designated task, thus there is starvation. We must ensure that transaction can be picked as a victim only a finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

CONCURRENCY CONTROL TECHNIQUES – CONCURRENCY CONTROL, LOCKING TECHNIQUES FOR CONCURRENCY CONTROL, TIME STAMPING PROTOCOLS FOR CONCURRENCY CONTROL, VALIDATION BASED PROTOCOL, MULTIPLE GRANULARITY, MULTIVERSION SCHEMES, RECOVERY WITH CONCURRENT TRANSACTION

Q.31. State and explain the three concurrency problems.

(R.G.P.V., June 2016)

Ans. Transactions submitted by the various users may execute concurrently and may access and update the same database items. If this concurrent execution is uncontrolled, it may lead to problems. The three problems are as follows –

(i) ***The Lost Update Problem*** – Consider the situation shown in fig. 4.14. This figure is meant to be read as follows – transaction A retrieves some tuple t at time t_1; transaction B retrieves that same tuple t at time t_2; transaction A updates the tuple at time t_3, and transaction B updates the same tuple at time t_4. Transaction A's update is lost at time t_4, because transaction B overwrites it without even looking at it.

Transaction A	Time	Transaction B
—	|	—
—	t_1	—
Retrieve t	|	—
—	t_2	—
—	|	Retrieve t
—	t_3	—
Update t	|	
—	t_4	—
—		Update t
—	↓	—

Fig. 4.14 Transaction A Loses an Update at Time t_4

(ii) ***The Uncommitted Dependency Problem*** – The uncommitted dependency problem arises if one transaction is allowed to retrieve – or, worse, update – a tuple that has been updated by another transaction but not yet committed by that other transaction. For if it has not yet been committed, there is always a possibility that it never will be committed but will be rolled back instead – in which case the first transaction will have seen some data that now no longer exists.

Consider figs. 4.15 and 4.16.

Transaction A	Time	Transaction B
—	|	—
—	t_1	—
—	|	Update t
—	t_2	—
Retrieve t	|	—
—	t_3	Rollback
—	↓	
—		

Fig. 4.15 Transaction A becomes Dependent on an Uncommitted Change at Tim

In the first example (fig. 4.15), transaction A sees an uncommitted update at time t_2. That update is then undone at time t_3. Transaction A is therefore operating on a false assumption – namely, the assumption that tuple t has the value seen at time t_2, whereas infact it has whatever it had prior to time t_1. As a result transaction A may produce incorrect output.

The second example (fig. 4.16) is even worse. Not only does transaction A become dependent on an uncommitted change at time t_2, but it actually loses an update at time t_3 because the rollback at time t_3 causes tuple t to be restored to its value prior to time t_1.

Transaction A	Time	Transaction B
–	|	–
–	t_1	–
–	|	Update t
–	t_2	–
Update t	|	–
–	t_3	–
–	↓	Rollback
–		

Fig. 4.16 Transaction A Updates an Uncommitted Change at Time t_2, and Loses that Update at Time t_3

(iii) The Inconsistent Analysis Problem – Fig. 4.17 shows two transactions A and B operating on account (ACC) tuples – Transaction A is summing account balances, transaction B is transferring an amount 10 from account 3 to account 1. The result produced by A, 110, is obviously incorrect; if A were to go on to write that result back into the database, it will actually leave the database in an inconsistent problem.

Acc 1	Acc 2	Acc 3
40	50	30

Transaction A	Time	Transaction B
–	|	–
–	|	–
Retrieve Acc 1 : sum = 40	t_1	–
Retrieve ACC 2 : sum = 90	t_2	–
–	|	–
–	t_3	–
–	|	–
–	t_4	Retrieve Acc 3
–	|	–
–	t_5	Update Acc 3 : 30 → 20
–	|	–
–	t_6	Retrieve Acc 1
–	|	–
–	t_7	Update Acc 1 : 40 → 50
–	|	–
–	t_8	Commit
Retrieve Acc 3 : sum = 110, not 120	↓	

Fig. 4.17 Transaction A Performs an Inconsistent Analysis

Q.32. Why is concurrency control important in any database ?

Ans. Suppose Ram and Shyam both read the same row from the employee table, and both are able to alter the data for a particular employee salary. Now the question arises whose changes should be saved ? Ram ? Shyam ? Neither ? Or a combination of both ? So, concurrency control takes care of the issues

d with allowing multiple users simultaneous updating a particular record. database without having a proper concurrency control in place will fail in nting the database consistency. Hence concurrency control is one of y feature of a relational database to keep the database in a stable state.

Q.33. What is meant by concurrent execution of database transaction in a multiuser system ? Discuss why concurrency control is needed and give example. ***(R.G.P.V., Nov. 2018)***

Ans. A database system is multiuser when many users can use the system, hence access the database concurrently. Most DBMSs are multiuser. For example, an airline reservations system is used by hundreds of travel agents reservation clerks concurrently. Systems in banks, insurance agencies, stock exchanges, supermarkets, and the like are also operated on by various users who submit transactions concurrently to the system.

Because of the concept of multiprogramming, which permits the computer to execute multiple programs or processes at the same time, multiple users to access databases and use computer systems simultaneously. Although, multiprogramming operating systems execute some commands from one process then suspend that process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence, concurrent execution of processes is actually interleaved, that represents two processes P and Q executing concurrently in an interleaved manner. Interleaving keeps the CPU busy if a process needs an input or output operation, like reading a block from disk. The CPU is switched to execute another process rather than remaining idle during I/O time. Interleaving also prevents a long process from delaying other processes.

Parallel processing of multiple processes is possible, if the computer system includes multiple hardware processors. In a multiuser DBMS, the primary resources are the stored data items which can be accessed concurrently by interactive users or application programs, that are constantly retrieving information from and modifying the database.

Also refer to Q.32.

Q.34. What is the advantage of locking mechanism ?

Ans. The purpose of database locking is to prevent a record from being processed simultaneously by two different users. In addition to this, it prevents other users from changing any column while the record is being processed.

Q.35. What are the problems of lock based protocols ? **(R.G.P.V., June 2016)**

Ans. Lock based protocols can lead to two major problems –

(i) The resulting transaction schedule may not be serializable.

(ii) The schedule may create deadlocks. A database deadlock, which is equivalent to traffic gridlock in a big city is caused when two transactions wait for each other to unlock data.

Q.36. What are database locks and how do they lead to deadlock ?

Ans. One way to ensure serializability is to require that data items be accessed in a mutually exclusive manner. That is, while one transaction is accessing a data item, no other transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item. A *lock* is a variable associated with a data item that describes the status of the data item with respect to possible operations that can be applied to it. Manipulating the value of a lock is called ***locking***. Generally, there is a one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database items.

There are two types of lock –

(i) ***Exclusive Lock*** – The exclusive lock is also called an update or a write lock. The intention of this mode of locking is to provide exclusive use of the data item to one transaction. If a transaction T_i has obtained an exclusive-mode lock (denoted by X) on item Q, then T_i can both read and write Q.

(ii) ***Shared Lock*** – The shared lock is also called a read lock. The intention of this mode of locking is to ensure that the data-item does not undergo any modifications. Any number of transactions can concurrently lock and access a data-item in the shared mode, but none of these transactions can modify the data-item. If a transaction T_i has obtained a shared-mode lock (denoted by S) on item Q, then T_i can read, but cannot write Q.

We require that every transaction request a lock in an appropriate mode on data item Q, depending on the types of operations that it will perform on Q. *The* transaction makes the request to the concurrency-control manager. The *transaction* can proceed with the operation only after the concurrency-control *manager* grants the lock to the transaction.

A compatibility function on a set of lock modes can be defined as follows –

Let A and B represent arbitrary lock modes. Suppose that a transaction *T_i requests* a lock of mode A on item Q on which transaction T_j ($T_i \neq T_j$) *currently holds* a lock of mode B. If transaction T_i can be granted a lock on *Q immediately; inspite* of the presence of the mode B lock, then mode A is *compatible with mode* B. This function can be represented by a matrix. The

…bility relation between the two modes of locking is shown in the …COMP of fig. 4.18. An element comp (A, B) of the matrix has the …e if and only if mode A is compatible with mode B.

…shared mode is compatible with shared …but not with exclusive mode. At any time, …al shared-mode locks can be held …taneously by different transactions on a …ular data item. A subsequent exclusive …lock request has to wait until the currently …shared-mode locks are released.

	S	X
S	True	False
X	False	False

Fig. 4.18 Lock-compatibility Matrix COMP

…transaction requests a shared lock on data item Q by executing the …S(Q) instruction. Similarly, a transaction requests an exclusive lock …gh the lock-X(Q) instruction. A transaction can unlock a data item Q by …unlock(Q) instruction.

To access a data item, transaction T_i must first lock that item. If the data …is already locked by another transaction in an incompatible mode, the …rrency control manager will not grant the lock until all incompatible …s held by other transactions have been released. Thus, T_i is made to wait …all incompatible locks held by other transactions have been released.

For example, consider a banking system, here we have two accounts A …B that are accessed by transactions T_1 and T_2. Transaction T_1 transfers …0 from account B to account A.

```
T1 : lock-X(B);
     read(B);
     B:= B – 50;
     write(B);
     unlock(B);
     lock-X(A);
     read(A);
     A: = A + 50;
     write(A);
     unlock(A);
```

Transaction T_2 displays total amount of money in accounts A and B, i.e., A+B.

```
T2 : lock-S(A);
     read(A);
     unlock(A);
     lock-S(B);
     read(B);
     unlock(B);
     display(A + B);
```

If two accounts have amounts $1000 and $2000 in account of A and B respectively. Then after executing transactions T_1 and T_2 serially in any order, i.e., T_1T_2 or T_2T_1, transaction T_2 result amount $3000.

But if these two transactions execute concurrently then some inconsistency may result as shown in fig. 4.19.

If unlocking is delayed to the end of the transaction as shown in transaction T_3 and T_4 then this unconsistency problem can be avoided.

T_1	T_2	Concurrency-control Manager
lock-X(B)		

read(B)
B := B − 50
write(B)
unlock(B)
		grant-X(B,T_1)

	lock-S(A)	

	read(A)
	unlock(A)
	lock-S(B)
		grant-S(A,T_2)

	read(B)
	unlock(B)
	display(A + B)
		grant-S(B,T_2)

lock-X(A)		

read(A)
A := A + 50
write(A)
unlock(A)
		grant-X(A,T_2)

Fig. 4.19 Schedule 1

```
T3 : lock-X(B);
     read(B);
     B := B – 50;
     write(B);
     lock-X(A);
     read(A);
     A := A + 50;
     write(A);
     unlock(B);
     unlock(A);
```

Transaction T_4 corresponds to T_2 with unlocking delayed, and is defined as –

```
T4 : lock-S(A);
     read(A).
     lock-S(B);
     read(B).
     display(A + B);
     unlock(A);
     unlock(B);
```

T_3	T_4
lock-X(B)	
read(B)	
B:= B – 50	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	

Fig. 4.20 Schedule 2

Unfortunately, locking can lead to an undesirable state. Consider the partial schedule of fig. 4.20 for T_3 and T_4. Since T_3 is holding an exclusive-mode lock on B and T_4 is waiting for T_3 to unlock B. Similarly, since T_4 is holding a shared-mode lock on A and T_3 is requesting an exclusive-mode lock on A. T_3 is waiting for T_4 to unlock A. Thus, we have arrived at a state where neither of these transactions can ever proceed with its normal execution. This is called ***deadlock***.

Q.37. What are database locks and how do they lead to deadlock ? Briefly strategies for preventing deadlocks. How do you detect deadlocks ?

(R.G.P.V., Dec. 2012)

Ans. Database Locks and Deadlock – Refer to Q.36.

Deadlock Prevention Strategies – Refer to Q.29.

Deadlock Detection – Refer to Q.29.

Q.38. Explain the concept of two-phase locking and show that it guarantees serializability. ***(R.G.P.V., Dec. 2010)***

Or

What is two-phase locking and how does it guarantee serializability ?

(R.G.P.V., Dec. 2017)

Ans. One protocol that ensures serializability is the two-phase locking protocol. A transaction is said to follow the two-phase locking protocol if all locking operations (*read_lock*, *write_lock*) precede the first unlock operation in the transaction. This transaction can be divided into two phases -

(i) Expanding/Growing (first) phase – During this phase, a transaction can obtain locks but cannot release any lock.

(ii) Shrinking (second) Phase – During this phase, a transaction can release locks, but cannot obtain any new locks.

If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) must be done during the expanding phase, and downgrading of locks (from write_locked to read_locked) must be done in the shrinking phase. Hence, a read-lock(X) operation that downgrades an already held write_lock on X can appear only in the shrinking phase

Transactions T_1 and T_2 in fig. 4.21 follow two-phase locking protocol

It can be proved that, if every transaction in a schedule follows the two-phase locking protocol, the schedule is guaranteed to be serializable, obviating the need to test for serializability of schedules any more. The locking mechanism, by enforcing two-phase locking rules, also enforces serializability.

Two-phase locking may limit the amount of concurrency that can occur in a schedule. This is because a transaction T may not be able to release an item X after it is through using it if T must lock an additional item Y later on; or conversely. Hence, X must remain

T_1	T_2

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y);
read_item(X);
X := X + Y;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	write_lock(Y);
	unlock(X);
	read_item(Y);
	Y := X + Y;
	write_item(Y);
	unlock(Y);

Fig. 4.21 Schedule Under Two-phase Locking

locked by T until all items that the transaction needs to read or write have been locked; only then can X be released by T. Meanwhile another transaction seeking to access X may be forced to wait, even though T is done with X; conversely, if Y is locked earlier than it is needed, another transaction seeking to access Y is forced to wait even though T is not using Y yet. This is the price for guaranteeing serializability of all schedules without having to check the schedules themselves. It is easy to recover.

Q.39. Explain two-phase locking protocol and also list advantages of this protocol. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.38.

Q.40. Describe strict two-phase locking protocol. (R.G.P.V., June 2010)

Or

Explain how strict 2-phase locking is implemented. Show with the example. ***(R.G.P.V., Nov. 2018)***

Ans. The strict two-phase locking or strict 2PL protocol has two rules –

(i) First Rule – If a transaction T wants to *read* (respectively, *modify*) an object, it first requests a **shared (respectively, exclusive)** lock on the object.

Even, a transaction that has an exclusive lock can also read the object; an additional shared lock is not required. A transaction that requests a lock is suspended until the DBMS is able to grant it the requested lock. The DBMS keeps track of the locks it has granted and ensures that if a transaction holds an exclusive lock on an object, no other transaction holds a shared or exclusive lock on the same object.

(ii) Second Rule – All locks held by a transaction are released when the transaction is completed.

Requests of acquire and release locks can be automatically inserted into transactions by the DBMS; users need not worry about these details.

In effect, the locking protocol allows only safe interleavings of transactions. If two transactions access completely independent parts of the database, they concurrently obtain the locks they need and proceed merrily on their ways. On the other hand, if two transactions access the same object, and one wants to modify it, their actions are effectively ordered serially – all actions of one of these transactions (the one that gets the lock on the common object first) are completed before (this lock is released and) the other transaction can proceed.

We denote the action of a transaction T requesting a shared (respectively, exclusive) lock on object O as $S_T(O)$ (respectively, $X_T(O)$) and omit the subscript denoting the transaction when it is clear from the context. As an

example, consider the schedule shown in fig. 4.22. This interleaving could result in a state that cannot result from any serial execution of the three transactions. For example, T_1 could change A from 10 to 20, then T_2 (which reads the value 20 for A) could change B from 100 to 200, and then T_1 would read the value 200 for B. If run serially either T_1 or T_2 would execute first, and read the values 10 for A and 100 for B.

T_1	T_2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
R(B)	
W(B)	
Commit	

Fig. 4.22 Reading Uncommitted Data

T_1	T_2
X(A)	
R(A)	
W(A)	

Fig. 4.23 Schedule Illustrating Strict 2PL

T_1	T_2
X(A)	
R(A)	
W(A)	
X(B)	
R(B)	
W(B)	
Commit	
	X(A)
	R(A)
	W(A)
	X(B)
	R(B)
	W(B)
	Commit

Fig. 4.24 Schedule Illustrating Strict 2PL with Serial Execution

If the strict 2-phase locking protocol is used, such interleaving is disallowed. Because assuming that the transactions proceed at the same relative speed as before, T_1 would obtain an exclusive lock on A first and then read and write A (see fig. 4.23). Then, T_2 would request a lock on A. However, this request cannot be granted until T_1releases its exclusive lock on A, and the DBMS therefore suspends T_2. T_1 now proceeds to obtain an exclusive lock on B, reads and writes B, then finally commits, at which time its locks are released. T_2's lock request is now granted, and it proceeds. In this example the locking protocol results in a serial execution of the two transactions, as shown in fig. 4.24.

However, the actions of different transactions could be interleaved. For example, consider the interleaving of two transactions shown in fig. 4.25, which is permitted by the strict two-phase locking protocol.

T_1	T_2
S(A)	
R(A)	
	S(A)
	R(A)
	X(B)
	R(B)
	W(B)
	Commit
X(C)	
R(C)	
W(C)	
Commit	

Fig. 4.25 Schedule Following Strict 2PL with Interleaved Actions

Q.41. Consider the following two transactions –

```
T1 : read (A);      T2 : write(A)
     read (B);           read(B)
B = A + B;
write (B)
```

Add lock and unlock instructions so that the transactions T1 and T2 observe two-phase locking protocol. Is it deadlock free ?

(R.G.P.V., Dec. 2010)

Ans. Implementation of Two-phase Locking – Transactions T1 and T2 of fig. 4.26 follow the two-phase locking protocol.

Determination of Deadlock – The schedule shown in fig. 4.26 will be deadlocked if transaction T2 will wait for A which is locked by T2. Also, T2 will wait for B which is locked by T1.

T1	T2
lock-S(A)	lock-X(A)
read(A)	write(A)
lock-X(B)	lock-S(B)
unlock(A)	unlock(A)
read(B)	read(B)
B = A+B	unlock(B)
write (B)	
unlock(B)	

Fig. 4.26 Schedule Under Two-phase Locking

Q.42. What is a timestamp ? How does the system generate timestamps ?

Ans. A ***timestamp*** is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the ***transaction start time.*** A timestamp of transaction T is denoted by TS(T).

Timestamps can be generated in several ways, one possibility is to use a counter that is incremented each time its value is assigned to a transaction. The transaction timestamps are numbered 1, 2, 3, ... in this scheme. A computer counter has a finite maximum value, so the system must periodically reset the counter to zero when no transactions are executing for some short period of time. Another way to implement timestamps is to use the current date/time value of the system clock and ensure that no two timestamp value are generated during the same tick of the clock.

Q.43. Discuss the timestamp ordering protocol for concurrency control. How does strict timestamp ordering differ from basic timestamp ordering ?

(R.G.P.V., Dec. 2010)

Ans. The idea for this scheme is to order the transactions based on their timestamps. A schedule in which the transactions participate is then serializable, and the equivalent serial schedule has the transactions in order of their timestamp values. This is called ***timestamp ordering (TO).*** Notice how this differs from 2 PL, where a schedule is serializable by being equivalent to some serial schedule allowed by the locking protocols. In timestamp ordering, however, the schedule is equivalent to the particular serial order corresponding to the order of the

action timestamps. The algorithm must ensure that, for each item accessed flicting operations in the schedule, the order in which the item is accessed not violate the serializability order. To do this, the algorithm associates with database item X two timestamp (TS) values –

(i) *Read_TS(X)* – The ***read timestamp*** of item X; this is the largest stamp among all the timestamps of transactions that have successfully of item X – that is, read_TS(X) = TS(T), where T is the youngest transaction that has read X successfully.

(ii) *Write_TS(X)* – The ***write timestamp*** of item X; this is the largest of all the timestamps of transactions that have successfully written item X – that is, write_TS(X) = TS(T), where T is the youngest transaction that has written X successfully.

Basic Timestamp Ordering – Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with the read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp. If T is aborted and rolled back, any transaction T_1 that may have used a value written by T must also be rolled back. Similarly, any transaction T_2 that may have used a value written by T_1 must also be rolled back, and so on. This effect is known as ***cascading rollback*** and is one of the problems associated with basic TO, since the schedules produced are not recoverable. An additional protocol must be enforced to ensure that the schedules are recoverable, cascadeless or strict. The concurrency control algorithm must check whether conflicting operations violate the timestamp ordering in the following two cases –

(i) Transaction T issues a write_item(X) operation –

(a) If read_TS(X) > TS(T) or if write_TS(X) >TS(T), then abort and roll back T and reject the operation. This should be done because some younger, transaction with a timestamp greater than TS(T), and hence after T in the timestamp ordening, has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.

(b) If the condition in part (a) does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

(ii) Transaction T issues a read_item(X) operation –

(a) If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with timestamp greater than TS(T), and hence after T in the timestamp order.ing, has already written the value of item X before T had a chance to read X.

(b) If write_TS(X) < = TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

Hence, whenever the basic TO algorithm detects two conflicting operations that occur in the incorrect order, it rejects the later of the two operations by aborting the transaction that issued it. The schedules produced by basic TO are hence guaranteed to be conflict serializable.

Strict Timestamp Ordering – A variation of basic TO called ***strict TO*** ensures that the schedules are both strict (for easy recoverability) and (conflict) serializable. In this variation, a transaction T that issues a read_item (X) or write_item(X) such that TS(T) > write_TS(X) has its read or write operation delayed until the transaction T' that wrote the value of X (hence TS(T') = write_TS(X)) has committed or aborted. To implement this algorithm, it is necessary to simulate the locking of an item X that has been written by transaction T until T' is either committed or aborted. This algorithm does not cause deadlock, since T waits for T' only if TS(T) > TS(T').

Q.44. Write short note on timestamp ordering protocol for concurrency control. ***(R.G.P.V., Nov. 2018)***

Ans. Refer to Q.43.

Q.45. What is time stamping ? Explain a mechanism of concurrency control that uses timestamping with the help of an examples. ***(R.G.P.V., Dec. 2014)***

Ans. Refer to Q.42 and Q.43.

Q.46. What is concurrent schedule ? What is conflict serializability in this context ? What is time-stamp based protocol in context of concurrent schedules ? ***(R.G.P.V., June 2011)***

Ans. **Concurrent Schedule –** Refer to Q.17.

Conflict Serializability – Refer to Q.9 (i).

Timestamp Based Protocol – Refer to Q.43.

Q.47. Explain multiple granularity level locking protocol.

Ans. Since the best granularity size depends on the given transaction, it seems appropriate that a database system support multiple levels of granularity, where the granularity level can be different for various mixes of transactions. Fig. 4.27 shows a simple granularity hierarchy with a database containing two files, each file containing several pages, and each page containing several records. This can be used to illustrate a ***multiple granularity level*** 2PL protocol, where a lock can be requested at any level.

Consider the following scenario, with only shared and exclusive lock types, that refers to the example in fig. 4.27. Suppose transaction T_1 wants to update all the records in file f_1, and T_1 requests and is granted an exclusive lock for f_1. Then all f_1's pages (p_{11} through p_{1n}) – and the records contained on those pages-are locked in exclusive mode. This is beneficial for T_1 because setting a single file – level lock is more efficient than setting n page-level locks or having to lock each individual record. Now suppose another transaction T_2 only wants to read record r_{1nj} from page p_{1n} of file f_1; then T_2 would request a shared record-level lock on r_{1nj}. However, the database system (that is, the transaction manager or more specifically the lock manager) must verify the compatibility of the requested lock with already held locks. One way to verify this is to traverse the tree from the leaf r_{1nj} to p_{1n} to f_1 db. If at any time a conflicting lock is held on any of those items, then the lock request for r_{1nj} is denied and T_2 is blocked and must wait. This traversal would be fairly efficient.

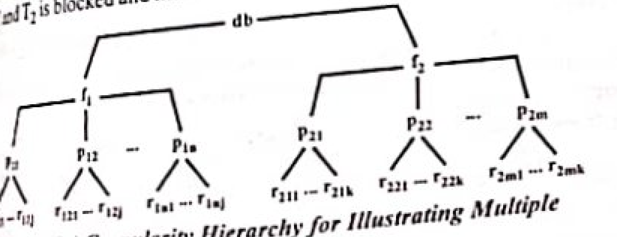


Fig. 4.27 A Granularity Hierarchy for Illustrating Multiple Granularity Level Locking

However, what if transaction T_2's request came before transaction T_1's request ? In this case, the shared record lock is granted to T_2 for r_{1nj}, but when T_1's file-level lock is requested, it is quite difficult for the lock manager to check all nodes (pages and records) that are descendants of node f_1 for a lock conflict. This would be very inefficient and would defeat the purpose of having multiple granularity level locks.

To make multiple granularity level locking practical, additional types of locks, called ***intention locks***, are needed. The idea behind intention locks is for a transaction to indicate, along the path from the root to the desired node, what type of lock (shared or exclusive) it will require from one of the node's descendants. There are three types of intention locks –

(i) Intention-shared (IS) indicates that a shared lock(s) will be requested on some descendant node(s).

(ii) Intention-exclusive (IX) indicates that an exclusive lock(s) will be requested on some descendant node(s).

(iii) Shared-intention-exclusive (SIX) indicates that the current node is locked in shared mode but an exclusive lock(s) will be requested on some descendant node(s).

The compatibility table of the three intention locks, and the shared and exclusive locks, is shown in fig. 4.28. Besides the introduction of the three types of intention locks, an appropriate locking protocol must be used. The ***multiple granularity locking*** (MGL) protocol consists of the following rules –

(i) The lock compatibility (based on fig. 4.28) must be adhered to.

(ii) The root of the tree must be locked first, in any mode.

(iii) A node N can be locked by a transaction T in S or IS mode only if the parent node N is already locked by transaction T in either IS or IX mode.

	IS	IX	S	SIX	X
IS	yes	yes	yes	yes	no
IX	yes	yes	no	no	no
S	yes	no	yes	no	no
SIX	yes	no	no	no	no
X	no	no	no	no	no

Fig. 4.28 Lock Compatibility Matrix for Multiple Granularity Locking

(iv) A node N can be locked by a transaction T in X, LX or SIX mode only if the parent of node N is already locked by transaction T in either IX or SIX mode.

T_1	T_2	T_3
IX(db)		
IX(f_1)		
	IX(db)	
		IS(db)
		IS(f_1)
IX(p_{11})		IS(p_{11})
X(r_{111})		
	IX(f_1)	
	X(p_{12})	
IX(f_2)		S(r_{11j})
IX(p_{21})		
X(r_{211})		
unlock(r_{211})		
unlock(p_{21})		
unlock(f_2)		
		S(f_2)
	unlock(p_{12})	
	unlock(f_1)	
unlock(r_{111})	unlock(db)	
unlock(p_{11})		
unlock(f_1)		
unlock(db)		
		unlock(r_{11j})
		unlock(p_{11})
		unlock(f_1)
		unlock(f_2)
		unlock(db)

Fig. 4.29 Lock Operations to Illustrate a Serializable Schedule

(v) A transaction T can lock a node only if it has not unlocked any node (to enforce the 2PL protocol).

(vi) A transaction T can unlock a node, N, only if none of the children of node N are currently locked by T.

Rule (i) simply stages that conflicting locks cannot be granted. Rules (ii), (iii) and (iv) state the conditions when a transaction may lock a given node in any of the lock modes. Rules (v) and (vi) of the MGL protocol enforce 2PL rules to produce serializable schedules. To illustrate the MGL protocol with the database hierarchy in fig. 4.27, consider the following three transactions –

(i) T_1 wants to update record r_{111} and record r_{211}.

(ii) T_2 wants to update all records on page p_{12}.

(iii) T_3 wants to read record r_{11j} and the entire f_2 file.

Fig. 4.29 shows a possible serializable schedule for these three transactions. Only the lock operations are shown. The notation <lock_type>(<item>) is used to display the locking operations in the schedule.

The multiple granularity level protocol is especially suited when processing a mix of transactions that include – (i) short transactions that access only a few items (records or fields), and (ii) long transaction that access entire files. In this environment, less transaction blocking and less locking overhead is incurred by such a protocol when compared to a single level granularity locking approach.

Q.48. *Describe multiversion timestamp ordering scheme.*

Or

Explain the term multiversion techniques. ***(R.G.P.V., Dec. 2011)***

Ans. In this method, several version Q_1, Q_2,....,Q_k of each data item Q are maintained. For each version, the value of version Q_i and the following two timestamps are kept –

(i) ***read_TS(Q_i)*** – The read timestamp of Q_i is the largest of all the timestamps of transactions that have successfully read version Q_i.

(ii) ***write_TS(Q_i)*** – The write timestamp of Q_i is the timestamp of the transaction that wrote the value of version Q_i.

Whenever a transaction T is allowed to execute a write_item (Q) operation, a new version Q_{k+1} of item Q is created, with both the write_TS (Q_{k+1}) and the read_TS(Q_{k+1}) set to TS(T). Correspondingly, when a transaction T is allowed to read the value of version Q_i, the value of read_TS(Q_i) is set to the larger of the current read_TS(Q_i) and TS(T).

To ensure serializability, the following two rules are used –

(i) If transaction T issues a write_item(Q) operation, and version i of Q has the highest write_TS(Q_i) of all versions of Q that is also less than or equal to TS(T), and read_TS(Q_i) > TS(T), then abort and roll back transaction

T. Otherwise, create a new version Q_j of Q with read_TS(Q_j) = write_TS(Q_j) = TS(T).

(ii) If transaction T issues a read_item(Q) operation, find the version i of Q that has the highest write_TS(Q_i) of all versions of Q that is also less than or equal to TS(T). Then, return the value of Q_i to transaction T, and set the value of read_TS(Q_i) to the larger of TS(T) and the current read_TS(Q_i).

In case (ii), a read_item(Q) is always successful, since it finds the appropriate version Q_i to read based on the write_TS of the various existing versions of Q. In case (i), however, transaction T may be aborted and rolled back. This happens if T is trying to write a version of Q that should have been read by another transaction T' whose timestamp is read_TS(Q_i). However, T' has already read version Q_i, which was written by the transaction with timestamp equal to write_TS(Q_i). If this conflict occurs, T is rolled back. Otherwise, a new version of Q, written by transaction T, is created. If T is rolled back, cascading rollback may occur. Hence, to ensure recoverability, a transaction T should not be allowed to commit until after all the transactions that have written some version that T has read have committed.

Q.49. What do you mean by time stamping protocol for concurrency control ? Discuss multi-version scheme of concurrency control also. ***(R.G.P.V., June 2010)***

Ans. **Time Stamping Protocol** – Refer to Q.43.

Multi-version Scheme of Concurrency Control – Refer to Q.48.

Q.50. Describe multiversion two-phase locking using certify locks.

Ans. In this multiple-mode locking scheme, there are locking modes for an item – read, write and certify. Hence, the state of LOCK(Q) for an item Q can be one of read-locked, write-locked, certify-locked, or unlocked. In the standard locking scheme with only read and write locks, a write lock is an exclusive lock. The relationship between read and write locks in the standard scheme by means of the ***lock compatibility table*** shown in fig. 4.30 (a). An entry of ***yes*** means that, if a transaction T holds the type of lock specified in the column header on item Q and if transaction T' requests the type of lock specified in the row header on the same item Q, then T' can obtain the lock because the locking modes are compatible. In contrast, an entry of no in the table indicates that the locks are not compatible, so T' must wait until T releases the lock.

	Read	Write
Read	yes	no
Write	no	no

(a) A Compatibility Table for Read/Write

	Read	Write	Certify
Read	yes	yes	no
Write	yes	no	no
Certify	no	no	no

(b) A Compatibility Table for Read/Write/Certify Locking Scheme

Fig. 4.30 Lock Compatibility Tables

In the standard locking scheme, once a transaction obtains a write lock on an item, no other transactions can access that item. The idea behind multiversion 2PL is to allow other transactions T' to read an item Q while a single transaction T holds a write lock on Q. This is achieved by allowing two versions for each item Q, one version must always have been written by some committed transaction. The second version Q' is created when a transaction T acquires a write lock on the item. Other transaction can continue to read the committed version of Q while T holds the write lock. Transaction T can write the value of Q' as required, without affecting the value of the committed version Q. However, once T is ready to commit, it must obtain a certify lock on all items that it currently holds write locks on before it can commit. The certify lock is not compatible with read locks, so the transaction may have to delay its commit until all its write-locked items are released by any reading transactions in order to obtain the certify locks. Once the certify locks, which are exclusive locks, are acquired, the committed version Q of the data item is set to the value of version Q', version Q' is discarded, and the certify locks are then released. Fig. 4.30 (b) shows the lock compatibility table for this scheme.

Q.51. Explain 2-phase commitment protocol and the behaviour of this protocol during lost messages and site failures. ***(R.G.P.V., June 2010)***

Ans. To maintain the atomicity of a multidatabase transaction, it is necessary to have a two-level recovery mechanism. A global recovery manager, or coordinator, is needed to maintain information needed for recovery, in addition to the local recovery managers and the information they maintain (log, tables). The coordinator usually follows a protocol called the ***two-phase commit protocol***, whose two phases can be stated as follows –

(i) ***Phase I*** – When all participating databases signal the coordinator that the part of the multidatabase transaction involving each has concluded, the coordinator sends a message "prepare for commit" to each participant to get ready for committing the transaction. Each participating database receiving that message will force-write all log records and needed information for local recovery to disk and then send a "ready to commit" or "OK" signal to the coordinator. If the force-writing to disk fails or the local transaction cannot commit for some reason, the participating database sends a "cannot commit" or "not OK" signal to the coordinator. If the coordinator does not receive a reply from a database within a certain time out interval, it assumes a "not OK" response.

(ii) Phase 2 – If all participating databases reply "OK", and the coordinator's vote is also "OK", the transaction is successful, and the coordinator sends a "commit" signal for the transaction to the participating databases. Because all the local effects of the transaction and information needed for local recovery have been recorded in the logs of the participating databases, recovery from failure is now possible. Each participating database completes transaction commit by writing a [commit] entry for the transaction in the log and permanently updating the database if needed. On the other hand, if one or more of the participating databases or the coordinator have a "not OK" response, the transaction has failed and the coordinator sends a message to "roll back" or UNDO the local effect of the transaction to each participating database. This is done by undoing the transaction operations, using the log.

The net effect of the two-phase commit protocol is that either all participating databases commit the effect of the transaction or none of them do. In case any of the participants – or the coordinator–fails, it is always possible to recover to a state where either the transaction is committed or it is rolled back. A failure during or before Phase1 usually requires the transaction to be rolled back, whereas a failure during Phase 2 means that a successful transaction can recover and commit.

Q.52. What is transaction in database ? Discuss properties of transaction. Explain transaction logging in databases and its use in two phase commit policy. ***(R.G.P.V., Dec. 2012)***

Ans. **Transaction** – Refer to Q.1.

Properties of Transaction – Refer to Q.2.

Transaction Logging – Refer to Q.22.

Use of Transaction Logging in Two Phase Commit Policy – Refer to Q.51.

Q.53. How do optimistic concurrency control techniques differ from the other concurrency control techniques ? Why they also called validation or certification techniques ? Discuss the various phases of an optimistic concurrency control method.

Ans. In all the concurrency control techniques, a certain degree of checking is done before a database operation can be executed. For instance, in locking, a check is done to determine whether the item being accessed is locked. In timestamp ordering, the transaction timestamp is checked against the read and write timestamps of the item. This checking represents overhead during transaction execution, with the effect of slowing down the transactions.

In optimization concurrency control techniques, also known as validation or certification techniques, no checking is done while the transaction is executing.

The idea behind optimistic concurrency control is to do all the checks at once. Hence, transaction execution proceeds with a minimum of overhead until the validation phase is reached. If there is little interference among transactions, most will be validated successfully. However, if there is much interference, many transactions that execute to completion will have their results discarded and must be restarted later. Under these circumstances, optimistic techniques do not work well. The techniques are called optimistic because they assume that little interference will occur and hence that there is no need to do checking during transaction execution.

Phases of Concurrency Control Protocol – There are three phases for concurrency control protocol –

(i) Read Phase – A transaction can read values of committed data items from the database. However, updates are applied only to local copies of the data items kept in the transaction workspace.

(ii) Validation Phase – Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.

(iii) Write Phase – If the validation phase is successful, the transaction updates are applied to the database; otherwise, the updates are discarded and the transaction is restarted.

Q.54. Discuss the concurrency control mechanism in detail using suitable example. ***(R.G.P.V., Dec. 2017)***

Or

Explain concurrency control techniques. Also discuss about locking technique. ***(R.G.P.V., Dec. 2016)***

Or

Explain techniques for concurrency control. ***(R.G.P.V., May 2018)***

Ans. When executing several transaction concurrently, they may result in interleaved operations. Thus, the isolation property of transaction not remains intact and results in an inconsistent state. Hence, it is required to control the interaction among concurrent transactions which is known as concurrency control. The concurrency control is implemented with the help of concurrency control techniques which are locking, timestamp, optimistic, multiple granularity level and multiversion technique.

Also, refer to Q.36, Q.43, Q.53, Q.47 and Q.48.

Q.55. Explain locking techniques for concurrency control. ***(R.G.P.V., Dec. 2009)***

Ans. Refer to Q.38, Q.41, Q.47, Q.50 and Q.53.

Q.56. What do you mean by locking techniques of concurrency control ? Discuss the various locking techniques and recovery with concurrent transaction also in detail. ***(R.G.P.V. Dec. 2011)***

Ans. **Locking** – Refer to Q.36.

Locking Techniques – Refer to Q.38, Q.41, Q.47, Q.50 and Q.53.

Recovery with Concurrent Transaction – The recovery techniques with multiple concurrent transactions are as follows –

(i) ***Interaction with Concurrency Control*** – The recovery scheme depends heavily on the concurrency control scheme that is used. To rollback a failed transaction, we must undo the updates performed by the transaction. Suppose that a transaction T_1 has to be rolled back, and a data item X that was updated by T_1 has to be restored to its old value. Using the log-based schemes for recovery, we restore the value by using the undo information in a log record. Now, suppose that a second transaction T_2 has performed yet another update on X before T_1 is rolled back. Then the update performed by T_2 will be lost if T_1 is rolled back.

Therefore, if a transaction T has updated a data item X, no other transaction may update the same data item until T has committed or been rolled back. We can ensure this requirement by using strict 2PL. This means that two-phase locking with exclusive locks held until the end of the transaction.

(ii) ***Transaction Rollback*** – A failed transaction T_i is rolled back by using the log. The system scans the log backward. For every log record of the form $<T_r, X_j, V_1, V_2>$ found in the log, the system restores the data item X_j to its old value V_1. Scanning of the log terminates when the log record $<T_i$, start> is found.

Scanning the log backward is important, since a transaction may have updated a data item more than once. For instance, consider the pair of log records

$<T_i, A, 10, 20>$

$<T_i, A, 20, 30>$

The log records represent a modification of data item A by T_i, followed by another modification of A by T_i. Scanning the log backward sets A correctly to 10. If the log were scanned in the forward direction, A would be set to 20, which is incorrect.

(iii) ***Checkpoints*** – In a concurrent transaction-processing system, we require that the checkpoint log record be of the form <checkpoint L>, where L is a list of transactions active at the time of the checkpoint. Again, it is assumed that transactions do not perform updates either on the buffer blocks or on the log while the checkpoint is in progress.

The requirement that transactions must not perform any updates to buffer blocks or to the log during checkpointing can be bothersome, because transaction processing will have to halt while a checkpoint is in progress. A ***fuzzy checkpoint*** is a checkpoint where transactions are allowed to perform updates even while buffer blocks are being written-out.

(iv) ***Restart Recovery*** – When the system recovers from a crash, it constructs two lists. The undo-list consists of transactions to be undone, and the redo-list consists of transactions to be redone.

The system constructs the two lists as follows – initially, they are both empty. The system scans the log backward, examining each record, until it finds the first <checkpoint> record –

(a) For each record found of the form $<T_i$ commit>, it adds T_i to redo-list.

(b) For each record found of the form $<T_i$ start>, if T_i is not in redo-list, then it adds T_i to undo-list.

When the system has examined all the appropriate log records, it checks the list L in the checkpoint record. For each transaction T_i in L, if T_i is not in redo-list then it adds T_i to the undo list.

Once both lists have been constructed, the recovery proceeds as follows –

(a) The system rescans the log from the most recent record backward, and performs an undo each log record that belongs transaction T_i on the undo-list. Log records of transactions on the redo-list are ignored in this phase. The scan stops when the $<T_i$ start> records have been found for every transaction T_i in the undo-list.

(b) The system locates the most recent <checkpoint L> record on the log.

(c) The system scans the log forward from the most recent <checkpoint L> record, and performs redo for each log record that belongs to a transaction T_i that is on the redo-list. It ignores log records of transactions on the undo-list in this phase.

After the system has undone all transactions on the undo-list, it redoes those transactions on the redo-list. In this case, it is important to process the log forward. When the recovery process has completed, transaction processing resumes.

It is important to undo the transaction in the undo-list before redoing transactions in the redo-list, using the algorithm in steps (a) to (c). Otherwise, a problem may occur. Suppose that data item A initially has the value 5. Suppose that a transaction T_i updated data item A to 20 and aborted. Transaction rollback would restore A to the value 5. Suppose that another transaction T_j then updated data item A to 30 and committed, after that the system crashed. The state of the log at the time of the crash is

$<T_i, A, 5, 20>$

$<T_j, A, 5, 30>$

$<T_j$ commit>

If the redo pass is performed first A will be set to 30; then, in the undo pass, A will be set to 5, which is wrong. The final value of Q should be 30, which can be ensure by performing undo before performing redo.

Q.57. Explain immediate update and deferred update of recovery techniques.
(R.G.P.V., June 2016)

Ans. The immediate-update technique allows database modifications to be output the database while the transaction is still in the active state. Data modifications written by active transactions are called ***uncommitted modifications.***

When a system crashes or a transaction fails, old value of the data item should be used for bringing the database into the consistent state. This can be done by undo operation. Before a transaction T_i starts its execution, the record <T_i **start**> is written to the log. During its execution, any write(X) operation by T_i is preceded by the writing of the appropriate new update record to the log. When T_i partially commits, the record <T_i **commit**> is written to the log.

We consider example of banking system taken earlier for transactions T_0 and T_1, such that T_0 is followed by T_1. For this, log record can be given as in fig. 4.31

<T_0 **start**>
<T_0, A, 1000, 950>
<T_0, B, 2000, 2050>
<T_0 **commit**>
<T_1 **start**>
<T_1, C, 700, 600>
<T_1 **commit**>

Fig. 4.31 Log Record Corresponding to T_0 and T_1

In this method for recovery, we use following two operations –

(i) Undo (T_i) – Restores the value of all data items updated by transaction T_i to the old values.

(ii) Redo (T_i) – Sets the values of all data items updated by transaction T_i to the new values. We need to undo a transaction T only when log contains the record <T **start**> but does not contain the record <T **commit**>. We need to redo a transaction T only when log contains the record <T **start**> and <T **commit**> both.

For example, we consider here three situations of log record when system crash occurs at different steps in fig. 4.32.

<T_0 **start**> <T_0, A, 1000, 950> <T_0, B, 2000, 2050>	<T_0 **start**> <T_0, A, 1000, 950> <T_0, B, 2000, 2050> <T_0 **commit**> <T_1 **start**> <T_1, C, 700, 800>	<T_0 **start**> <T_0, A, 1000, 950> <T_0, B, 2000, 2050> <T_0 **commit**> <T_1 **start**> <T_1 , C, 700, 800> <T_1 **commit**>
(a)	*(b)*	*(c)*

Fig. 4.32 Log Record at 3 Different Situations

Considering the fig. 4.32(a), if system crash occurs just after the log record for the step **write**(B) of transaction T_0 has been written to the stable storage. During the recovery, we do undo (T_0) as we have only <T_0 **start**> in log record but not <T_0 **commit**>.

If system crash occurs just after the log record for the step **write**(C) of transaction T_1 has been written to stable storage as in fig. 4.32(b).

During recovery, we do redo(T_0) and undo(T_1) as we have both <T_0 start> and <T_0 **commit**> in log record. But we do not have <T_1 **commit**> and <T_1 start> in log record. Undo(T_1) should be done first than redo(T_0) should be done. If system crash occurs just after the log record <T_1 **commit**> has been written to the stable storage space as in fig. 4.32 (c).

During recovery both redo(T_0) and redo(T_1) are performed.

Deferred database modification technique ensures transaction atomicity by recording all database modifications in the log. In this technique, all the write statements of the transaction are applied on the database only when the transaction is partially committed. A transaction is said to be partially commited when the final action of the transaction has been executed.

When a transaction partially commits, then the information on the log associated with the transaction is used in executing the deferred writes. If the system crashes before the transaction completes its execution or if the transaction aborts, then the information on the log is ignored.

The execution of transaction T_i proceeds as follows –

Before T_i starts its execution, a record <T_i **start**> is written in the log. A write(X) operation by T_i results in the writing of a new record to the log. Finally, when T_i partially commits, a record <T_i **commit**> is written to the log.

For example, a banking system have accounts of A, B and C with initial balances $1000, $2000 and $700 respectively. We transfer $50 from account A to account B through transaction T_0 -

For this, we write

```
T0 : read(A);
     A := A – 50;
     write(A);
     read(B);
     B := B + 50;
     write(B);
```

Let transaction T_1 withdraws $100 from account C. Then transaction T_1 can be defined as

```
T1 : read(C);
     C := C – 100;
     write(C);
```

Also, assume that these transactions execute serially in the order T_0 followed by T_1.

Log record for these transactions will have values.

<T_0 **start**>
<T_0, A, 950>
<T_0, B, 2050>
<T_0 **commit**>
<T_1 **start**>
<T_1, C, 600>
<T_1 **commit**>

Fig. 4.33 Log Record Corresponding to T_0 and T_1

Using the log, the system can handle any failure that results in the loss of information on volatile storage. The recovery scheme uses the following recovery procedure –

***redo** (T_i)* sets the value of all data items updated by transaction T_i to the new values. The set of data items updated by T_i and their respective new values can be found in the log. The redo operation must be idempotent. That is, executing it several times must be equivalent to executing it once.

After a failure, the recovery subsystem consults the log to determine which transactions need to be redone. Transaction T_i is redone if and only if the log record contains both <T_i **start**> **and** <T_i **commit**> statements.

For example, we again consider our banking example with transactions T_0 and T_1 executed one after the other in the order T_0 followed by T_1.

If system fails just after the log record for the step write(B) of transaction T_0 as shown in fig. 4.34(a). Then during recovery no redo operation will be done as we have only <T_0 **start**> in log record but not <T_0 **commit**>.

<T_0 **start**> <T_0, A, 950> <T_0, B, 2050>	<T_0 **start**> <T_0, A, 950> <T_0, B, 2050> <T_0 **commit**> <T_1 **start**> <T_1, C, 600>	<T_0 **start**> <T_0, A, 950> <T_0, B, 2050> <T_0 **commit**> <T_1 **start**> <T_1, C, 600> <T_0 **commit**>
(a)	*(b)*	*(c)*

Fig. 4.34 Log at Three Different Situations

If system crash occurs just after the log record write C as shown in fig. 4.34(b). Then during recovery only redo (T_0) is done, as we have only <T_0 start> and <T_0 **commit**> in log disk. At the same time, we have <T_1 start> in log disk but not <T_1 **commit**> so redo (T_1) will not be done.

Similarly, if crash occurs just after the log record <T_1 commit> as shown in fig. 4.34(c).

Then during recovery we will perform both redo (T_0) and redo (T_1) as we have both <T_0 **start**>, <T_0 **commit**> and <T_1 **start**>, <T_1 **commit**> in log disk.

INTRODUCTION TO DISTRIBUTED DATABASES, DATA MINING, DATA WAREHOUSING, OBJECT TECHNOLOGY AND DBMS, COMPARATIVE STUDY OF OODBMS VS DBMS, TEMPORAL, DEDUCTIVE, MULTIMEDIA, WEB AND MOBILE DATABASE

Q.58. Write short note on data mining.

Ans. The term ***data mining*** refers to the process of semiautomatically analyzing large databases to find useful patterns to guide decisions about future activities. Like knowledge discovery in artificial intelligence (also called machine learning), or statistical analysis, data mining attempts to discover rules and patterns from data. However, data mining differs from machine learning and statistics is that it deals with large volumes of data stored on disk. That is, data mining deals with knowledge discovery in databases. There is a general expectation that data mining tools should be able to identify these patterns in the data with minimal user input. The patterns identified by such tools can give a data analyst useful and unexpected insight that can be more carefully investigated subsequently perhaps using other decision support tools.

Some types of knowledge discovered from a database can be represented by a set of rules. For example, a rule stated informally as – "Young women with annual incomes greater than $ 50, 000 are the most likely people to buy small sports cars." Such rules are not universally true. Other types of knowledge are represented by equations relating different variables to each other, or by other mechanisms for predicting outcomes when the values of some variables are known. There are a variety of possible types of patterns that may be useful, and different techniques are used to find different types of patterns.

Q.59. Write explanatory note on data warehousing. Draw its architecture.

Or

Define the concept of data warehousing. *(R.G.P.V., Dec. 2014)*

Ans. A data warehouse is a repository (or archive) of information gathered from multiple sources, stored under a unified schema at a single site. Once gathered, the data are stored for a long time, permitting access to historical data. Thus, data warehouses provide the user a single consolidated interface to data, making decision support queries easier to write. Moreover, by accessing information for decision support from a data warehouse, a decision maker ensures that online transaction-processing systems are not affected by the decision-support workload.

W.H. Inmon characterized a data warehouse as "a subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management's decisions." Data warehouses provide access to data for complex analysis, knowledge discovery, and decision making.

Data warehouses are designed precisely to support efficient extraction, processing, and presentation for analytic and decision-making purposes. In comparison to traditional databases, data warehouses contain very large amounts of data from multiple sources that may include databases from different data models and sometimes files acquired from independent systems and platforms.

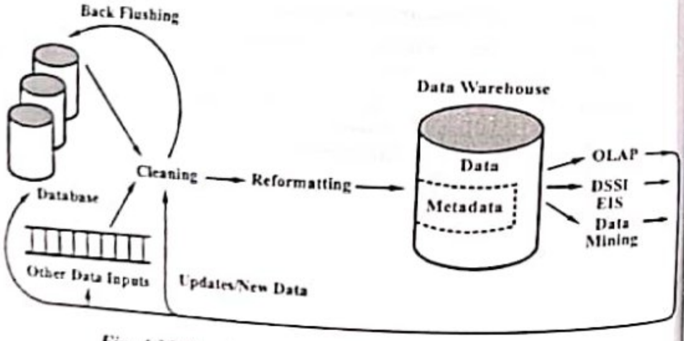


Fig. 4.35 The Overall Process of Data Warehousing

In general, a data warehouse is a collection of decision support technologies, aimed at enabling the knowledge worker to make better and faster decisions. Fig. 4.35 shows the conceptual structure of a data warehouse and its process. This process includes cleaning and reformatting of data before its warehousing. At the back end of process, OLAP, data mining, and DSS may generate new relevant information such as rules. The fig. 4.35 also shows that data sources may include files.

Q.60. Write short note on object-oriented database system.

(R.G.P.V., June 2005, Dec. 2006)

Ans. Objects are mainly focused in the object-oriented systems. So, the question arises that what is object ? Then, we can say "everything is an object".

...objects are immutable (examples might be integers i.e., 3, 42 and ...r strings i.e., "Rahul", ...) Other objects are mutable (...s might be the department ...ployee objects). Every ... has a type (the object term ...s)

...jects include properties and ...s that operate on it. Object ...logy can be shown as in fig.

Object Term	Traditional Term
immutable object	value
mutable object	variable
object class	type
method	operator
message	operator invocation

Fig. 4.36 Object Terminology

...jects are encapsulated i.e., the internal structure of such an object, ...DEPT ("department") object is not visible to users of that object, ...d users know only that the object is capable of executing certain ...ions (methods). For example, the methods that apply to DEPT objects ...ht be

HIRE_EMP, FIRE_EMP etc.

Object Identity – Objects in an object-oriented database usually ...spond to an entity in the enterprise being modeled by the database. An ...t retains its identity even if some of its properties change over time.

Likewise, an object retains its identity even if some or all of the values of ...ables or definitions of methods change over time.

Object Properties – Object-oriented system has some properties, few ...hem are as follows –

(i) Abstraction (data : = hiding), (ii) Inheritance

(iii) Encapsulation, (iv) Security, (v) Reusability etc.

Q.61. Write short note on object-orient database design.

(R.G.P.V., Nov./Dec. 2007)

Ans. Object-oriented design lies in the life cycle of software system. ...ject-oriented design methods have evolved from object-oriented ...gramming. The essence of OOP lies in the way that data and functions that ...nipulate them are brought together into a single entity called an ***object***. An ...ject is a data structure together with descriptions of the way it can be ...nipulated. Data are manipulated by one object by sending a message to ...other object which contain them. The message includes identification of ...nipulation required and then object determines how to manipulate itself ...pending on the content of the message.

Scanned with CamScanner

The term object in object-oriented design terminology is reserved to mean occurrences of a class and as such is a program run time, rather than a design consideration. Thus, object-oriented design deals with classes of things rather than occurrences of them. Classes in object-oriented design can be represented by abstract data types. An abstract data type describes a class in terms of operations that can be carried out on the underlying data it represents together with the properties of these operations which a user of them needs to know. It exploits the principle of information hiding. This means that in a well designed system, a module is seen by other modules only through the interface provided by that module.

The manner in which the underlying data is structured and actual mechanisms to manipulate it, are not visible in abstract data type. All that is 'on view' is the name of the class it represents and information about the interfaces it provides to other modules. Thus, an object-oriented design can be thought as a collection of abstract data types structured in some way in relationship to each other.

Q.62. Compare OODBMS and DBMS. ***(R.G.P.V., Dec. 2015)***

Or

Write short note on comparison between OODBMS and DBMS.

(R.G.P.V., Nov. 2018)

Ans. OODBMS is a result of blending object oriented programming and database technology to meet the application needs of system designed in OOT. It scores over pure relational database as it gives benefit of object orientation and its ability to have equivalence to object oriented programs. There are many benefits.

A database management system is a set of prewritten programs that are used to store, update and retrieve a database. The most important change brought about by DBMS is that the programs no longer interact with the data files directly. Instead they communicate with the DBMS, which acts as a middle agency. It controls the flow of information from and to the database.

Q.63. Explain the following –

(i) Time stamping protocol for concurrency control

(ii) OODBMS and compare it with RDBMS.

(R.G.P.V., Dec. 2009)

Ans. ***(i) Time Stamping Protocol for Concurrency Control*** – Refer to Q.43.

(ii) OODBMS and Compare it with RDBMS – Although RDBMS is the preferred choice of the computing industry worldwide, a recent development has taken place in the area of **Object Oriented Database Management Systems** technology. An OODBMS provides a persistent or permanent storage.

OODBMS is generally used in a multi-user client/server environment. It controls the concurrent access to objects, provides locking mechanisms and transactional features, offers security features at the object level and also ensures object backup and restoration.

The biggest difference between a RDBMS and OODBMS is that whereas the former stores the data related to an object in a table, the latter stores the state of an object. Each object in an OODBMS has a unique Object Identifier (OID), which is used to identify and link it with other objects when needed (e.g. in referential integrity relationships). An OODBMS supports encapsulation and inheritance.

We know that SQL is used in conjunction with RDBMS. OODBMS generally uses class definitions and traditional OOP languages, such as C++ and Java, to define manipulate and retrieve data. That is, an OODBMS is an extension of the in-memory data structures such as objects. An OODBMS integrates database-handling operations directly into the base OOP language. The idea is shown in fig. 4.37.

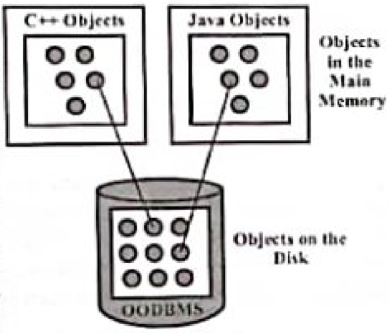


Fig. 4.37 OODBMS Concept

From what we have discussed about OODBMS, it appears that a programmer must be familiar with OODBMS with C++ or OODBMS with Java syntaxes, etc. In other words, are we saying that the syntax for using OODBMS in C++ is different from using OODBMS in Java? Does it mean that we must *relearn* OODBMS syntax the moment we change from one OOP language to another ?

Q.64. Write short note on temporal database.

Ans. Most database systems model the current state of the world. When the state of the real world changes, the database gets updated, and information about the old state gets lost. However, in many applications, it is important to store and retrieve information about past states. Databases that store information about states of the real world across time are called ***temporal databases.*** In broad sense, temporal databases encompass all database applications that require some aspect of time when organizing their information. There are many applications where some aspect of time is needed to maintain the information in a database. These include healthcare, where patient histories need to be maintained; insurance, where claims and accident histories are required as well as information on the times when insurance policies are in effect, and so on.

Q.65. Write short note on deductive databases.

Ans. A deductive database uses two main types of specifications_facts and rules. Facts are specified in a manner similar to the way relations are specified, except that it is not necessary to include the attribute names. A tuple in a relation describes some real-world fact whose meaning is partly determined by the attribute names. In a deductive database, the meaning of an attribute value in a tuple is determined solely by its position within the tuple. Rules are somewhat similar to relational views. They specify virtual relations that are not actually stored but that can be formed from the facts by applying inference mechanisms based on the rule specifications. The main difference between rules and views is that rules may involve recursion and hence may yield virtual relations that cannot be defined in terms of standard relational views.

The evaluation of Prolog programs is based on a technique called backward chaining, which involves a top-down evaluation of goals. In the deductive databases that use Datalog, attention has been devoted to handling large volumes of data stored in a relational database. Hence, evaluation techniques have been devised that resemble those for a bottom-up evaluation. Prolog suffers from the limitation that the order of specification of facts and rules is significant in evaluation; moreover, the order of literals within a rule is significant. The execution techniques of Datalog programs attempt to circumvent these problems.

Q.66. Write explanatory note on multimedia databases.

Ans. Multimedia databases are special types of database that stores particular multimedia data such as audio, video and images. Now a days multimedia data typically are stored outside the database, in file systems. This kind of storage is not a problem when the number of multimedia objects is relatively small, since features provided by databases are usually not important.

However, database features become important when the number of multimedia objects stored is large. Issues such as transactional updates, querying facilities, and indexing then become important. Multimedia objects often have descriptive attributes, such as those indicating when they were created, who created them, and to what category then belong one approach to building a database for such multimedia objects is to use databases for storing the descriptive attributes and for keeping track of the files in which multimedia objects are stored.

However, storing multimedia outside the database makes it harder to provide database functionality, such as indexing on the basis of actual multimedia content. It can lead to inconsistencies. Therefore, it is desirable to store the data themselves in the database.

Q.67. Write short note on databases on WWW.

Or

What is web databases ? ***(R.G.P.V., Dec. 2014, 2015)***

Ans. The database on the *WWW* is a collection of data that can be accessed and retrieved from anywhere in the world. The world wide Web, sometimes referred as 'the Web', originally developed in Switzerland in early 1990 as large scale hypermedia information service system for biological scientists to share information. Today this technology allows universal access to this shared information to anyone having access to the Internet.

The Web technology is based on client-server technology and Web pages formatted with HTML (hyper text markup language). A page has many hyperlinks, means, a link that enables a user to browse or move from one page to another across the Internet. This ability provides a tremendous power to the user for searching and navigating related information, even across different continents.

Information on the Web is organized according to a uniform resource locator (URL), similar to an address that provides completes pathname of a file, and the path contains name of machine, directory and filename separated by slashes like following –

http://www.shivanipublications.com/DBMS107.html

The URL always begins with ***hypertext transport protocol (http)***, that is the protocol used by Web browsers. Generally, a collection of HTML documents and other files accessible via the URL on Web server is a ***Web site*** in the given example, www.shivanipublications.com.

Accessing Database on the WWW – The Web server uses a standard interface ***common gateway interface (CGI)*** to act as additional layer between the user interface front-end and the DBMS back-end that facilitates access to the database. The CGI executes the user programs or scripts to obtain the information and returns the information to the server in HTML, which is back to the browser.

The existing approaches are divided into two categories –

(i) Access Using CGI Scripts – In this method client request with server for data, the server uses CGI scripts for data to database. All

works are done by using query, and given back the data in same manner to clients. This process and structure of this method is given in fig. 4.38.

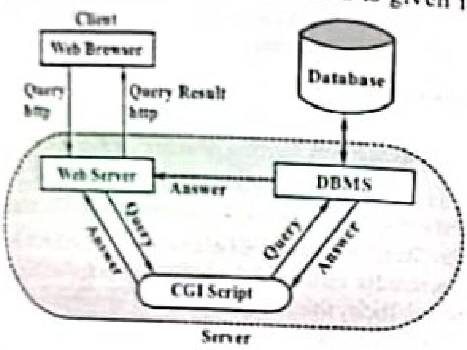


Fig. 4.38 Database Access on the WWW by Using CGI Scripts

(ii) *Access Using JDBC – Java Database Connectively (JDBC)* is a set of Java classes to allow access to relational databases through the execution of SQL statements. It is a way of connecting databases without any additional process to reach client process.

Q.68. Explain web and mobile database. ***(R.G.P.V., May 2018)***

Ans. Web Database – Refer to Q.67.

Mobile Database – Recent advances in wireless technology have led to mobile computing, a new dimension in data communication and processing. The mobile computing environment will provide database applications with useful aspects of wireless technology The mobile computing platform allows users to establish communication with other users and to manage their work while they are mobile. This feature is especially useful to geographically dispersed organizations. However, there are a number of hardware as well as software problems that must be resolved before the capabilities of mobile computing can be fully utilized. Some of the software problems – which may involve data management, transaction management, and database recovery – have their origin in distributed database systems. In mobile computing, however, these problems become more difficult to solve, mainly because of the narrow bandwidth of the wireless communication channels, the relatively short active life of the power supply (battery) of mobile units, and the changing locations of required information (sometimes in cache, sometimes in air, sometimes at the server). In addition mobile computing has its own unique architectural challenges.

UNIT 5

STUDY OF RELATIONAL DATABASE MANAGEMENT SYSTEMS THROUGH ORACLE/MYSQL – ARCHITECTURE, PHYSICAL FILES, MEMORY STRUCTURES, BACKGROUND PROCESS

Q.1. What do you understand by RDBMS ?

Ans. A database management system based on relational model is called elational database management system or RDBMS. A relational model consists f two-dimensional data structures, to store data, called the relations. Each elation has a set of attributes to describe the properties of data contained in it, nd for each attribute, a relation has related data in the form of tuples (or rows).

Let us consider table 5.1.

Table 5.1 A Student Relation

Roll_No	*Name*	*City*
1001	Rahul	Gwalior
1002	Suman	Jabalpur
1003	Ajay	Jabalpur
1004	Anil	Gwalior

Table 5.2 Marks Table

Roll_No	*Marks*
1001	92
1002	81
1003	78
1004	85

Table 5.1 shows a student relation as a structure which has attributes Roll_No, Name, and City with four rows. A relation is also called a table and the attributes are called columns.

In RDBMS, tables can be related to each other. The relation is established by using common values for column/s that belonging to the separate tables. Let us consider table 5.2. In table 5.2, the column Roll_No should contain those roll numbers that exist in student table.

Q.2. Explain RDBMS. ***(R.G.P.V., June 2011)***

***Ans.* RDBMS – Re** [illegible].

RDBMS Architec[illegible] BMS provides a total environment for the development and runnin[illegible] nal database systems. Fig. 5.1 shows typical RDBMS 'onion skin' a[illegible]

In fig. 5.1, outer layer consists of various interfaces available to system developers and users. Next layer consists of RDBMS software that must run in the environment given by the operating system. Databases exist under control of this layer.

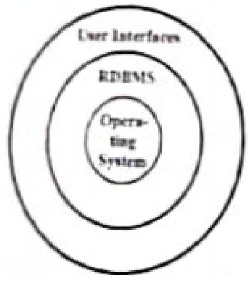


Fig. 5.1

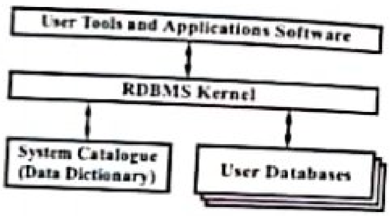

User Tools and Applications Software
RDBMS Kernel
System Catalogue (Data Dictionary)
User Databases

Fig. 5.2 RDBMS Architecture

Any RDBMS consists of a kernel part and a set of user interfaces, or tools, that interact with that kernel (see fig. 5.2).

All access to the database by the user tools or by any applications software system developed with their use, is through the kernel. Access to the kernel from the tools and/or applications is by means of SQL commands.

A single copy of RDBMS can support many distinct user databases. The interaction between kernel and database is transparent to both users and tools they use. Conceptually RDBMS stores databases in two parts. The storage schema that consists of table, view and index definitions, together with associated access permissions, user defined specifications affecting physical placement of data on disks and so forth, are stored in some form of system catalogue, sometimes referred to as the RDBMS's data dictionary. Actual data in databases are stored by the kernel using information that held in catalogue.

The system catalogue is used to store other administrative information about system and its databases. Some of their data content is directly accessible by users (in particular by DBA). The user tools and kernel of RDBMS can be organized in a client-server architecture.

RDBMS supporting the client-server architecture, can take advantage of this equipment arrangement by allowing user tools and/or applications to run in client host processor, whilst the kernel runs in the server that also hosts the databases. The effect is to partition the processing work load across the network, thus maximizing the use of very significant computing power now available with modern, relatively inexpensive, workstations and PCs, whilst at same time reducing the need for powerful central processors.

Q.3. Explain basic component of RDBMS. ***(R.G.P.V., Dec. 2016)***

Ans. Refer to Q.2.

Q.4. What is the difference between DBMS and RDBMS ? ***(R.G.P.V., Dec. 2016)***

Ans. The differences between DBMS and RDBMS are given in table 5.3.

Table 5.3

No.	*DBMS*	*RDBMS*
(i)	DBMS applications store data as file, so there will be no relation between the tables.	RDBMS applications store data in a tabular form, so relationship between these data values will be stored in the form of a table as well.
(ii)	In DBMS, data is generally stored in either a hierarchical form or a navigational form.	In RDBMS, the tables have an identifier called primary key and the data values are stored in the form of tables.
(iii)	DBMS has to provide some uniform methods to access the stored information.	RDBMS system supports a tabular structure of the data and a relationship between them to access the stored information.
(iv)	DBMS does not support distributed database.	RDBMS supports distributed database.
(v)	DBMS does not apply any security with regards to data manipulation.	RDBMS defines the integrity constraints for the purpose of ACID property.

Q.5. What do you mean by Oracle ? Give its architecture.

Ans. Oracle is one of the most popular relational database management system (RDBMS). It is based on the client/server architecture. A client sends the requests, such as connection to the database or retrieving data from the database, for several operations. A server accepts, processes and returns the results of the request, such as the data requested by the client, back to the client. Oracle provides the software for both the server and the client.

Oracle's popularity is based on large measures of exceptional performance and scalability for users, applications and data.

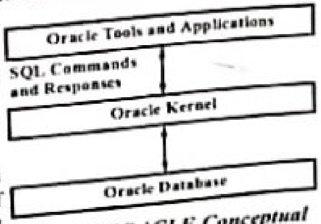


Fig. 5.3 ORACLE Conceptual Structure

Oracle is organized into a central, kernel part and variety of interfaced tools that make use of it. The general organization of the product is represented conceptually in fig. 5.3.

The kernel part of the structure is responsible for the definition and storage of data, database access control and concurrent access handling, backup and recovery of the database and interpretation of SQL commands. Although some of the user interfaces provided by the ORACLE tools do not require the use of SQL on the part of their users, the interface between all tools and the kernel is driven by SQL commands and the responses the kernel makes to them. ORACLE provides full support for communications networking and distributed databases with its SQL*Net and SQL*Star products.

The architecture of Oracle comprises of database and instance. An Oracle database is a collection of files having data and programs to manage the data. The instance comprises of the memory structures and processes. The memory structures are areas of the computer's memory that are used to store the data of the database. The processes operate on the database and memory structures to perform various database operations such as fetching data from the database and storing it in the memory structures. Most of these processes are referred to as the background processes. The memory area allocated for an Oracle instance is called SGA or System Global Area.

Q.6. Discuss about storage organization in Oracle.

Ans. A database is divided into logical storage units called tablespaces, with the following characteristics –

(i) Each database is divided into one or more table-spaces.

(ii) There is system tablespace and users tablespace.

(iii) One or more data-files are created in each table-space. A datafile can be associated with only one database.

(iv) The combined storage capacity of a database's tablespace is the total storage capacity of the database.

Every Oracle database contains a tablespace named SYSTEM to hold the data dictionary's objects which Oracle creates automatically when the database is created.

Physical storage is organized in terms of data blocks, extents, and segments. The finest level of granularity of storage is a ***data block***, which is a fixed number

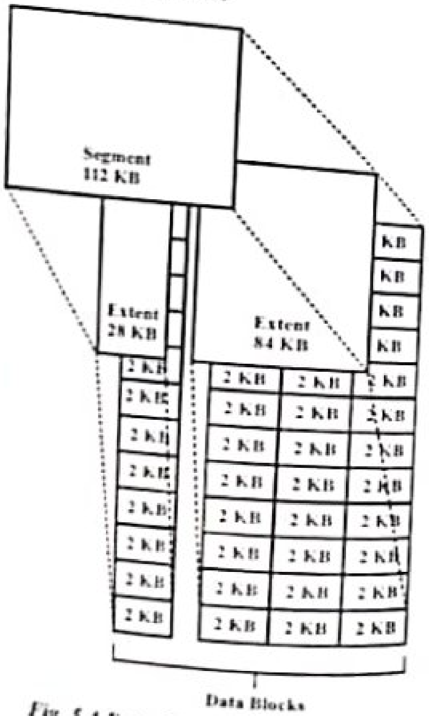


Fig. 5.4 Data Blocks, Extents and Segments in Oracle

extent is a specific number of contiguous data blocks. A segment extents allocated to a specific data structure. For a table, the data stored in a ***data segment*** and the index may be stored in an ***index*** The relationship among these terms are shown in fig. 5.4.

Discuss the various Oracle physical storage structures (or physical files).

Oracle database is made up of a set of physical files that reside on server's disk drives. Some of these files, such as the datafiles, redo and archived redo log files, hold real user data. Other structures, control files, maintain the state of the database objects. Text-based trace files contain logging information for both routine events and conditions in the database.

The physical files are discussed below –

(i) ***Datafiles*** – Oracle database must contain at least one *datafile*. Oracle datafile corresponds to one physical operating system file on the Each datafile in Oracle database is a member of one and only one space. A tablespace, however, can consist of many datafiles. The exception file tablespace, which consists of exactly one datafile.

(ii) ***Redo Log Files*** – Whenever data is added, removed, or changed table, index, or other Oracle object, an entry is written to the current ***redo*** file. Oracle database must have at least two redo log files, because Oracle redo log files in a circular fashion. When one redo log file is filled with log entries, the current log file is marked as ACTIVE, if it is still needed instance recovery, or INACTIVE, if it is not needed for instance recovery. next log file in the sequence is reused from the beginning of the file and is marked as CURRENT.

(iii) ***Control Files*** – Oracle database has at least one *control file* that contains the metadata of the database. Metadata is the data about the physical structure of the database itself (the table and field definitions). Among other things, the control file contains the name of the database, when the database was created, and the names and locations of all datafiles and redo log files. In addition, the control file maintains information used by Recovery Manager (RMAN), such as the persistent RMAN settings and the types of backups that have been performed on the database. Whenever any changes are made to the structure of the database, the information about the changes is immediately reflected in the control file. Because the control file is so critical to the operation of the database, it can also be multiplexed (one or more control files can be copied). However, no matter how many copies of the control file are associated with an instance, only one of the control files is designated as primary for purposes of retrieving database metadata.

(iv) ***Archived Log Files*** – Oracle database can operate in one of two modes – ARCHIVELOG or NOARCHIVELOG mode. When the database is in NOARCHIVELOG mode, the circular reuse of the redo log files (also known as the *on line* redo log files) means that redo entries (the contents of previous transactions) are no longer available in case of a failure to a disk drive or another media-related failure. Operating a NOARCHIVELOG mode does protect the integrity of the database in the event of an instance failure or system crash, because all transactions that are committed but not yet written to the datafiles are available in the online redo log files only. So crash recovery is limited to entries currently in online redo logs. If your last backup of datafiles fails before your earliest redo log file, you cannot recover your database.

In contrast, ARCHIVELOG mode sends a filled redo log file to one or more specified destinations and can be available to reconstruct the database at any given point in time in the event that a database media failure occurs. For example, if the disk drive containing the datafiles crashes, the contents of the database can be recovered to a point in time before the crash, given availability of a recent backup of the datafiles, the redo log files, and archived log files that were generated since the backup occurred.

(v) ***Initialization Parameter Files*** – When a database instance starts, the memory for the Oracle instance is allocated, and one of two types of *initialization parameter files* is opened – either a text-based file called *init <SID>.ora* (known generically as init.ora or a PFILE), or a server parameter file (SPFILE). The instance first looks for an SPFILE in the default location for the operating system ($ORACLE_HOME/dbs on Unix, for example) as either spfile <SID>. ora or spfile.ora. If neither of these files exists, the instance looks for a PFILE with the name init <SID>.ora. Alternatively, the STARTUP command can explicitly specify a PFILE to use for startup of Oracle.

Initialization parameter files, regardless of their format, specify file locations for trace files, control files, filled redo log files, and so forth. They also set limits on the sizes of the various structures in the System Global Area (SGA), as well as how many users can connect to the database simultaneously.

(vi) ***Alert and Trace Log Files*** – When things go wrong, Oracle can and often does write messages to the *alert log*, and in the case of background processes or user sessions, *trace log* files.

The alert log file, located in the directory specified by the initialization parameter BACKGROUND_DUMP_DEST, contains the most significant routine status messages as well as critical error conditions. When the database is started up or shut down, a message is recorded in the alert log, along with a list of initialization parameters that are different from their default values. In addition, any ALTER DATABASE or ALTER SYSTEM commands issued by the DBA are recorded. Operations involving tablespaces and their datafiles are recorded here, too, such as adding a tablespace, dropping a tablespace, and adding a datafile to a tablespace. Error conditions, such as tablespaces running out of space, corrupted redo logs, and so forth, are also recorded here–all critical conditions.

e trace files for the Oracle instance background processes are also a BACKGROUND_DUMP_DEST. For example, the trace files for process monitor) and SMON (system monitor) contain an entry when occurs or when SMON needs to perform instance recovery. The es for QMON (queue monitor) contain informational messages when s a new process. Trace files are also created for individual user sessions ections to the database. These trace files are located in the directory ed by the initialization parameter USER_DUMP_DEST.

(vii) ***Backup Files*** – Backup files can originate from a number of es, such as operating system copy commands or Oracle RMAN. If the performs a "cold" backup, the backup files are simply operating system of the datafiles, redo log files, control files, archived redo log files, and ed.

Q.8. Explain the various Oracle memory structures.

Ans. Oracle uses the server's physical memory to hold many things for racle instance – the Oracle executable code itself, session information, vidual processes associated with the database, and information shared een processes.

In addition, the memory structures contain user and data dictionary SQL ments, along with cached information that is eventually permanently stored

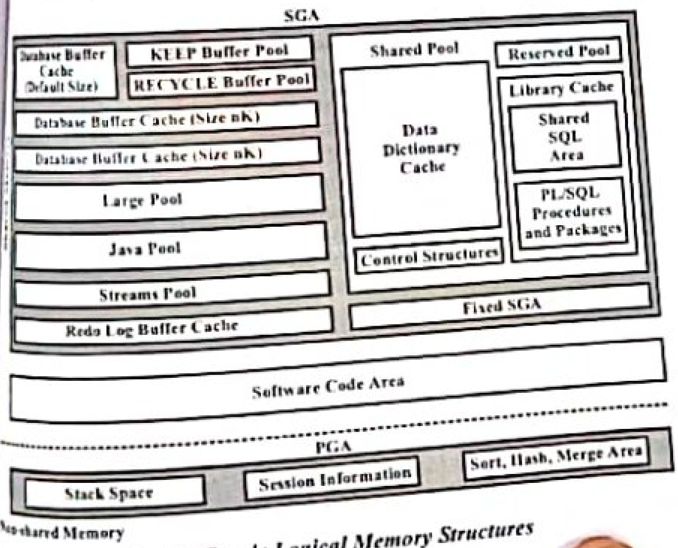


Fig. 5.5 Oracle Logical Memory Structures

on disk, such as data blocks from database segments and information about completed transactions in the database. The data area allocated for an Oracle instance is called the *System Global Area* (SGA). The Oracle executables reside in the software code area. In addition, an area called the Program Global Area (PGA) is private to each server and background process. One PGA is allocated for each user session or server process. Fig. 5.5 shows the relationships between the oracle memory structures.

The following structures are explained –

(i) ***System Global Area*** – It is a group of shared memory structures for an oracle instance, shared by the users of the database instance. When an oracle instance is started, memory is allocated for the SGA based on the values specified in the initialization parameter file or hard-coded in the Oracle software. Some parameters that control the size of the various parts of the SGA are dynamic, if the parameters SGA_MAX_SIZE is specified, the total size of all SGA areas, must not exceed the value of SGA_MAX_SIZE. If SGA_MAX_SIZE is not specified, but the parameter SGA_TARGET is specified, Oracle automatically adjusts the sizes of the SGA components, so that the total amount of memory allocated is equal to SGA_TARGET.

Memory in the SGA is allocated in unit of granules. A granule can be either 4 MB or 16 MB, depending on the total size of the SGA. If the SGA is less than or equal to 128 MB, a granule is 4 MB. Otherwise, it is 16 MB.

(ii) ***Buffer Caches*** – The buffer cache holds blocks of data from disk that have been recently need to satisfy a SELECT statement or that contain modified blocks that have been changed or added from a DML statement.

(iii) ***Shared Pool*** – It contains two subcaches, one is the library cache and other is the data dictionary cache. The shared pool is sized by the SHARED_POOL_SIZE initialization parameter.

(iv) ***Redo Log Buffer*** – It holds the most recent changes to the data blocks in the datafiles. When the redo log buffer is one-third full, or every 3 seconds,Oracle writes redo log records to the redo log files. The entries in the redo log buffer, once written to the redo log files, are critical to database recovery if the instance crashes before the changed data blocks are written from the buffer cache to the datafiles. A user's committed transaction is not considered complete until the redo log entries have been successfully written to the redo log files.

(v) ***Large Pool*** – It is an optional area of the SGA. It is used for transactions that interact with more than one database, message buffers for processes performing parallel queries, and RMAN parallel backup and restore operations. As the name suggest that the large pool makes available large blocks of memory for operations that need to allocate large blocks of memory at a time.

(vi) ***Java Pool*** – The Java pool is used by the Oracle JVM (Java Virtual Machine) for all Java code and data within a user session. Storing Java

d data in the Java pool is similar to SQL and PL/SQL code cached in ed pool.

(vii) ***Streams Pool*** – The streams pool is sized by using the tion parameter STREAMS_POOL_SIZE. The streams pool holds data ntrol structures to support the Oracle Streams feature of Oracle se Edition. Oracle streams manages the sharing of data and events in buted environment.

(viii) ***Program Global Area*** – It is an area of memory allocating ic section of itself, privately for one set of connection processes. The guration of the PGA depends on the connection configuration of the t database, either shared server or dedicated server.

(ix) ***Software Code Area*** – It stores the oracle executable files that are ing as part of an Oracle instance. These code areas are static in nature and ge only when a new release of the software is installed. The Oracle software areas are located in a privileged memory area separate from other user programs.

Q.1 What are background processes ? Explain in details.

Ans. When an Oracle instance starts, multiple background processes start. kground process is a block of executable code designed to perform a cific task. Fig. 5.6 shows the relationship between the background processes, e database, and the oracle SGA.

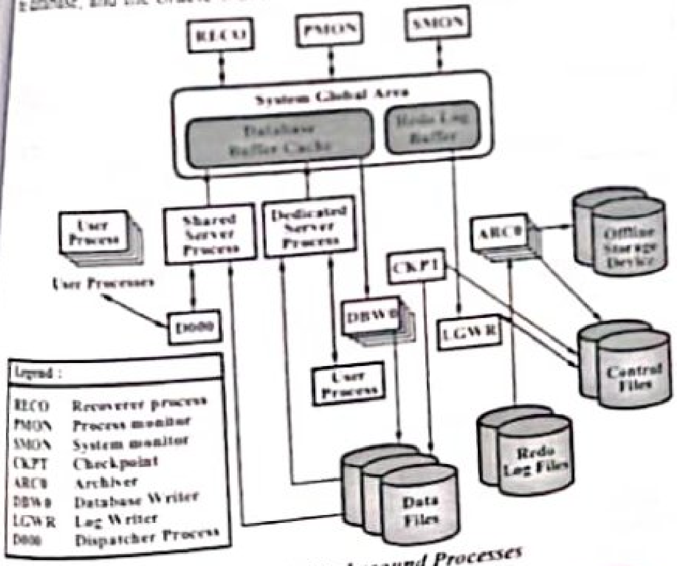


Fig. 5.6 Oracle Background Processes

There are many types of Oracle background processes. Each performs a specific job in helping to manage the instance. Five Oracle background processes are required and several background processes are optional. Table 5.4 describes the required background processes, and table 5.5 describes some of the optional background processes.

Table 5.4 Required Oracle Background Processes

Process Name	*Operating System Process*	*Description*
System Monitor	SMON	Performs instance recovery following an instance crash, coalesces free space in the database, and manages space used for sorting.
Process Monitor	PMON	Cleans up failed user database connections.
Database Writer	DBWn*	Writes modified database blocks from the SGA's Database Buffer Cache to the datafiles on disk.
Log Writer	LGWR	Writes transaction recovery information from the SGA's Redo Log Buffer to the online Redo Log files on disk.
Checkpoint	CKPT	Updates the database files following a Checkpoint Event.

Table 5.5 Optional Oracle Background Processes

Process Name	*Operating System Process*	*Description*
Archiver	ARCn	Copies the transaction recovery information written to disk by LGWR (log writer) to the online Redo Log files and to a secondary location in case it is needed for recovery. Nearly all production databases uṣe this optional process.
Recoverer	RECO	Recovers failed transactions that are distributed across multiple databases when using Oracle's distributed database feature.
Job Queue Monitor	CJQn	Assigns jobs to the Job Queue processes when using Oracle's job scheduling feature.
Job Queue	Jnnn	Executes database jobs that have been scheduled using Oracle's job scheduling feature.
Queue Monitor	QMNn	Monitors the message in the message queue when Oracle's Advanced Queuing feature is used.
Parallel Query Slave	Qnnn	Used to carry out portions of alarger overall query when Oracle's Parallel Query feature is used.
Dispatcher	Dnnn	Assigns user's database requests to a queue where they are then serviced by Shared Server processes when Oracle's Shared Server feature is used.
Shared Server	Snnn	Server Processes that are shared among several users when Oracle's Shared Server feature is used.
Memory Manager	MMAN	Manages the size of each individual SGA component when Oracle's Automatic Shared Memory Management feature is used.
Memory Monitor	MMON	Gathers and analyzes statistics used by the Automatic Workload Repository feature.
Memory Monitor Light	MMNL	Gathers and analyzes statistics used by the Automatic Workload Repository feature.
Recovery Writer	RVWR	Writes recovery information to disk when Oracle's Flashback Database Recovery feature is used.
Change Tracking Writer	CTWR	Keeps track of which database blocks have changed when Oracle's incremental Recovery Manager feature is used.

CONCEPT OF TABLE SPACES, SEGMENTS, EXTENTS AND BLOCK, DEDICATED SERVER, MULTI THREADED SERVER, DISTRIBUTED DATABASE, DATABASE LINKS AND SNAPSHOT

Q.10. Explain the term table spaces.

Ans. A tablespace consists of one or more datafiles. A datafile can be a part of one and only one tablespace. For an installation of Oracle 11g a minimum of two tablespaces are created, the SYSTEM tablespace and the SYSAUX tablespace. A default installation of oracle 11g creates six tablespaces. Oracle 11g allows you to create a special kind of tablespace called a *bigfile* tablespace, which can be as large as 128TB (terabytes).

The DBA (data base administrator) can manage the tablespace as a unit without worrying about the size and structure of the underlying datafiles.

If a tablespace is *temporary*, only the segments saved in the tablespace are temporary. The tablespace itself is permanent. A temporary tablespace can be used for sorting operations and for tables that exists only for the duration of the user's session.

Tablespaces can be either *dictionary managed* or *locally managed*. In a dictionary managed tablespace, extent management is recorded in data dictionary tables. Therefore, even if all application tables are in the USERS tablespace, the SYSTEM tablespace will still be accessed for managing Data Manipulation Language (DML) on application tables. In a locally managed tablespace, Oracle maintains a bitmap in the header of each datafile to track space availability.

As of Oracle 9i, if the SYSTEM tablespace is locally managed, then all other tablespaces must be locally managed if both read and write operations are to be performed on them. Dictionary managed tablespaces must be read only in databases with a locally managed SYSTEM tablespace.

Q.11. Discuss segments in detail.

Ans. A segment is a group of extents that form a database object that oracle treats as a unit, such as a table or index. There are four types of segments in an Oracle database – data segments, index segments, temporary segments, undo segments.

Every table in a database resides in a single data segment consisting of one or more extents. Oracle allocates more than one segment for a table if it is a partitioned table or a clustered table. Data segments include LOB (Large Object) segments that store LOB data referenced by a LOB locator column in a table segment.

Each index is stored in its own index segment. As with partitioned tables, each partition of a partitioned index is stored in its own segment.

When a user's SQL statement needs disk space to complete an operation, such as a sorting operation that cannot fit in memory, Oracle allocates a temporary segment. Temporary segments exists only for the duration of the SQL statement.

As of Oracle 10g, manual rollback segments exist only in the system tablespace, and the DBA does not need to maintain the SYSTEM rollback segment. A rollback segment was created to save the previous values of a database DML operation in case the transaction was rolled back, and to maintain the "before" image data to provide read consistent views of table data for other users accessing the table. Rollback segment were also used during database recovery for rolling back uncommitted transactions that were active when the database instance crashed or terminated unexpectedly.

In Oracle 10g, Automatic Undo Management handles the automatic allocation and management of rollback segments within an undo tablespace. Within an undo tablespace, the undo segments are structured similarly to rollback segments, except that the details of how these segments are managed is under control of Oracle, instead of being managed by the DBA. Automatic undo segments were available starting with Oracle 9i, but manually managed rollback segments are still available in Oracle 10g. In Oracle 11g, Automatic Undo Management is enabled by default.

Q.12. Define segment, extents and block. ***(R.G.P.V., Dec. 2014)***

Ans. **Segment** – Refer to Q.11.

Extents – It is the next level of logical grouping in the database. An extent consists of one or more database blocks. When you enlarge a database object, the space added to the object is allocated as an extent. Extents are managed by Oracle at the datafile level.

Block – A database block is the smallest unit of storage in Oracle. The size of a block is a specific number of bytes of storage within a given tablespace, within the database.

To facilitate efficient disk I/O performance, a block is usually a multiple of the operating system block size. The default block size is specified by the Oracle initialization parameter DB_BLOCK_SIZE. Most operating systems will allow as many as four other block sizes to be defined for other tablespaces in the database. Some high-end operating systems will allow fine block sizes. The blocks in the SYSTEM, SYSAUX, and any temporary tablespaces must be of the size DB_BLOCK_SIZE.

Q.13. Explain dedicated and multi-threaded server. (R.G.P.V., Dec. 2014)

Or

Explain the dedicated server and multi-threaded server.
(R.G.P.V., May 2018)

Ans. **Dedicated Server** – In an upscale restaurant, you may have had your own personal waitperson. That waitperson is there to greet you and escort you to your seat. They take your order for food and drinks. No matter how many other patrons enter the restaurant, your waitperson is responsible for serving only your requests.

A dedicated server environment works in much the same way. Every client connection is associated with a dedicated server process, sometimes called a *shadow process*, on the machine, where the Oracle server exists. No matter how many other connections are made to the server, the same dedicated server is always responsible for processing only your requests. You use the services of that server process until you disconnect from the Oracle server.

In dedicated server connection, every single user connecting to Oracle will have a personal genie handling data retrieval from disk into the buffer cache. If there are 200 users connected to Oracle, there will also be 200 genies out there grabbing data from disk and putting it in the buffer cache for those users. The architectural setup means that every user gets his or her data retrieval requests acted upon immediately. It also means there will be additional memory and CPU overhead on the machine running the Oracle database, and that each dedicated server process will, depending on the workload and the access method, sit idle most of the time. Still, this is the setup chosen by many DBAs for overall performance reasons, when hardware resources are readily available.

Multi-threaded Server – The MTS architecture eliminates the need for a dedicated server process for each connection (see fig. 5.7). A small number of shared servers can perform the same amount of processing as many dedicated servers. Also, since the amount of memory required for each user is relatively small, less memory and process management are required, and more users can be supported.

Features such as connection pooling help increase user scalability by increasing the number of possible simultaneous connections to the database.

To set up your system in a MTS configuration, start the listener process and set the MTS_DISPATCHERS parameter in the initialization file (INITSID.ORA).

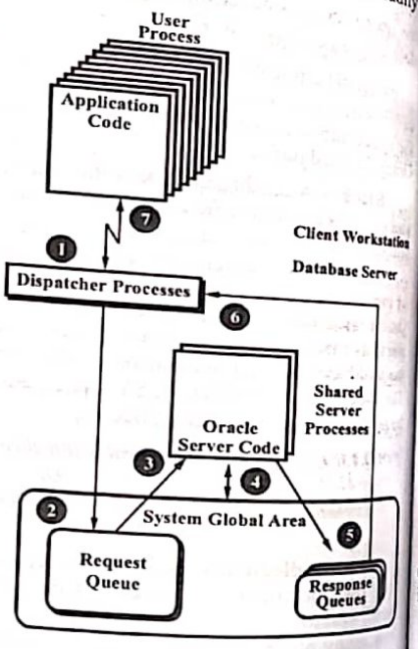


Fig. 5.7

After setting this parameter, restart the instance, which at this point will use the MTS configuration. MTS_DISPATCHERS should be set in the following manner –

mts_dispatchers = "(attribute = value)"

Q.14. Define the following terms –
(i) Distributed system (ii) Catalog.

(R.G.P.V., Dec. 2010)

Ans. (i) Distributed System – Distributed system is a central system connected to intelligent remote devices, each of which can itself be a computer

interconnected, possibly heterogeneous, computers. The distribution of processing power creates a feasible environment for data distribution. All distributed data must still be accessible from each site. Thus, *distributed database* can be defined as consisting of a collection of data with different sites under the control of separate DBMSs, running on independent computer systems. All the computers are interconnected and each system has autonomous processing capability, serving local applications. Each system participates in the execution of one or more global applications. Such applications require data from max than one site.

(ii) ***Catalog*** – The various commercial database products adopt different conventions and terminology with regard to their system catalog. However, in general, the catalogs contain similar metadata describing conceptual, internal and external schemas.

In ORACLE, the collection of metadata is called the data dictionary. The metadata is information about schema objects, such as tables, indexes, views, triggers and more. Access to the data dictionary is allowed through numerous views, which are divided into three categories – *USER, ALL* and *DBA*. These terms are used as prefixes for the various views. The views that have a prefix of USER contain schema information for objects owned by a user as well as objects that the user has been granted access to and those with a prefix of DBA are for the database administrator and contain information about all database objects.

The system catalog contains information about all three levels of database schemas – external (view definitions), conceptual (base tables) and internal (storage and index descriptions).

Q.15. Write short note on the distributed databases.

(R.G.P.V., June 2004, 2005, 2006, 2011)

Ans. The idea of distributed databases was born when the technology of databases was married to the concept of networking. The idea is shown in fig. 5.8.

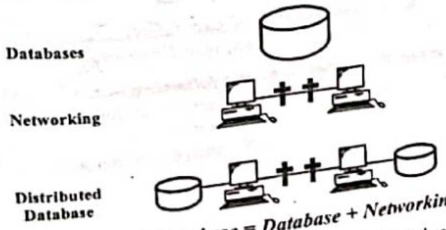


Fig. 5.8 Distributed Database = Database + Networking

The concept of distributed databases took off with the advent of wired and wireless data communication facilities. It assumed greater significance when the Internet was born. Now, that the internet has become an extremely important medium for conducting business worldwide, distributed databases have become the most effective technology for sharing data across distances.

The term distributed databases has nothing to do with the processing logic as such. A distributed database is a collection of multiple logically-interrelated databases, connected to one another over a computer network. This also leads us to the definition of a *Distributed Database Management System* (DDBMS).

The fundamental concept of a distributed database is the fact that a database is physically distributed across many computers should be hidden from the users. The users perceive a distributed database as a single database which is physically located on a single computer. The users have no idea – in fact, they should not have an idea that the database is scattered across many computers. This concept is shown in fig. 5.9.

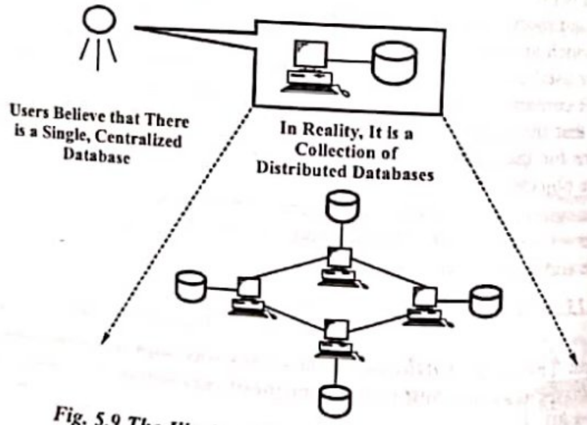

Users Believe that There is a Single, Centralized Database
In Reality, It Is a Collection of Distributed Databases

Fig. 5.9 *The Illusion of Centralised Database in a Distributed Database Environment*

Q.16. What do you mean by the following –

(i) Dedicated server

(ii) Multi-threaded server

(iii) Distributed database.

(R.G.P.V., Dec. 2009)

Ans. ***(i) Dedicated Server*** – Refer to Q.13.

(ii) Multi-threaded Server – Refer to Q.13.

(iii) Distributed Database – Refer to Q.15.

Q.17. Discuss the factor that does not appear in centralized systems that affect concurrency control and recovery in distributed system.

(R.G.P.V., Dec. 2008)

Ans. For concurrrency control and recovery purposes, numerous problems or factors arise in a distributed DBMS environment that are not encountered in a centralized DBMS environment. These are as follows –

(i) Dealing with Multiple Copies of the Data Items – The concurrency control method is responsible for maintaining consistency among these copies. The recovery method is responsible for making a copy consistent with other copies if the site on which the copy is stored fails and recovers later.

(ii) Failure of Individual Sites – The DDBMS should continue to operate with its running sites, if possible, when one or more individual sites fail. When a site recovers, its local database must be brought up-to-date with the rest of the sites before it joins the system.

(iii) Failure of Communication Links – The system must be able to deal with failure of one or more of the communication links that connect the sites. An extreme case of this problem is that ***network partitioning*** may occur. This breaks up the sites into two or more partitions, where the sites within each partition can communicate only with one another and not with sites in other partitions.

(iv) Distributed Commit – Problems arise with committing a transaction that is accessing databases stored on multiple sites if some sites fail during the commit process. The two-phase commit protocol is used to deal with this problem.

(v) Distributed Deadlock – Deadlock may occur among several sites, so techniques for dealing with deadlocks must be extended to take this into account.

Q.18. What is distributed database system ? How it is different from the centralized database system ? Give the uses of distributed system.

(R.G.P.V., Nov. 2018)

Ans. **Distributed Database System** – Refer to Q.15.

Difference between Centralized Database System and Distributed Database System – There are several differences between centralized database system and distributed database system as follows –

(i) The database administrator of the central site controls the database in a centralized system, while in a distributed system, there is a global database administrator responsible for the whole system.

(ii) Recovery from failure is more complex in distributed systems as compare to centralized systems.

(iii) The main difference between centralized and distributed database systems is that in the former, the data reside in one single location, whereas in the latter, the data reside in several locations.

(iv) A type of database that contains a single database located at one location in the network is known as centralized database, while in distributed database refers to a type of database that contains two or more database files located at different locations in the network.

(v) Managing, updating and taking in backups of data is easier because there is only one database file, while as there are various database files in a distributed database, it requires time to synchronize data.

(vi) In centralized database, the requires time for accessing data because various users access the database file, while in distributed database, speed in accessing the data is higher because the data is retrieved from the nearest database file.

(vii) In centralized database, if the database fails, the users do not have access to a database, while in distributed database if one database fails, the users can still access other database files.

(viii) Centralized database has more data consistency and it gives the complete view to the user, while in distributed database can have data replications, and there can be some data inconsistency.

Uses of Distributed System – A distributed DBMS software must be able to provide the following functions –

(i) Security
(ii) Keeping track of data
(iii) Replicated data management
(iv) System catalog management
(v) Distributed query processing
(vi) Distributed database recovery
(vii) Distributed transaction management
(viii) Distributed directory (catalog) management
(ix) Communication services to provide access to remote data.

Q.19. What are the advantages and disadvantages of distributed database management system ?

Or

Explain the advantages and disadvantages of distributed database.

(R.G.P.V., Dec. 2014)

Ans. There are several advantages and disadvantages of distributed database management system.

Advantages –

*(i) **Data are Located Near The Greatest Demand Site*** – The data in a distributed database system are dispersed to match business requirements.

*(ii) **Faster Data Access*** – End users often work with only a locally stored subset of the company's data.

*(iii) **Faster Data Processing*** – A distributed database system spreads out the system's workload by processing data at several sites.

*(iv) **Growth Facilitation*** – New sites can be added to the network without affecting the operations of other sites.

*(v) **Improved Communications*** – Because local sites are smaller located closer to customers, local sites foster better communication among departments and between customers and company staff.

*(vi) **Reduced Operating Costs*** – It is more cost effective to add workstations to a network than to update a mainframe system. Development and is done more cheaply and more quickly or low-cost PCs then on mainframes.

*(vii) **User-friendly Interface*** – PCs and workstations are usually equipped with an easy-to-use graphical user interface (GUI). The GUI simplifies use and training for end users.

*(viii) **Less Danger of a Single-Point-Failure*** – When one of the computers fails, the workload is picked up by other workstations. Data are also distributed at multiple sites.

*(ix) **Processor Independence*** – The end user is able to access any available copy of the data, and an end user's request is processed by any processor at the data location.

Disadvantages –

*(i) **Complexity of Management and Control*** – Applications must recognize data location, and they must be able to stitch together data from different sites. Database administrators must have the ability to coordinate database activities to prevent database degradation due to data anomalies. Transaction management, concurrency control, security, backup, recovery, query optimization, access path selection, and so on, must all be addressed and resolved.

*(ii) **Security*** – The probability of security lapses increases when data are located at multiple sites, the responsibility of data management will be shared by different people at several sites.

*(iii) **Lack of Standards*** – There are no standard communication protocols at the database level. For example, different database vendors employ different and often incompatible techniques to manage the distribution of data and processing in a DDBMS environment.

*(iv) **Increased Storage Requirements*** – Multiple copies of data are required at different sites, thus requiring additional disk storage space.

*(v) **Increased Training Cost*** – Training costs are generally higher in a distributed model than they would be in a centralized model, sometimes even to the extent of offsetting operational and hardware savings.

Q.20. What do you understand by distributed databases ? Give the various advantages and disadvantages of distributed database management system.

(R.G.P.V., Dec. 2017)

Ans. Refer to Q.15 and Q.19.

Q.21. What are the main advantages of a distributed database system?
(R.G.P.V., May 2018)

Ans. Refer to Q.19.

Q.22. Write short note on the database links.

Ans. Oracle databases can reference data that is stored outside of the local database. When referencing such data, you must specify the fully qualified name of the remote object.

To specify an access path to an object in a remote database, you will need to create a database link. Database links can either be public (available to all accounts in that database) or private (created by a user for only that account's use). When you create a database link, you specify the name of the account to connect to, the password for the account, and the service name associated with the remote database. If you do not specify an account name to connect to, Oracle will use your local account name and password for the connection to the remote database.

The following example creates a public link named MY_LINK –

```
create public database link MY_LINK
connect to HR identified by PUFFINSTUFF
using 'DB1';
```

Here, the link specifies that when it is used, it will open up a session in the database identified by the service named DB1. When it opens the session in the DB1 instance, it will log in as the user account HR, with the password PUFFINSTUFF. The service names for instances are stored in configuration files used by Oracle Net. The configuration file for service names is called tnsnames.ora, and it specifies the host, port, and instance associated with each service name.

To use this link for a table, the link must be specified in the *from* clause, as in the following example –

```
select * from EMPLOYEE@MY_LINK ;
```

The preceding query will access the EMPLOYEE table through the MY_LINK database link. You can create a synonym for this table, as shown in the following SQL command –

```
create synonym EMPLOYEE for EMPLOYEE@MYLINK ;
```

Q.23. Write short note on the snapshot.

Ans. Snapshots are objects in Oracle that enable you to replicate data from a table in one database to a copy of the table in another. Privileges include *create snapshot, create any snapshot, alter any snapshot,* and *drop any snapshot.*

In other words, each collection of statistics is called a *snapshot.* Snapshots of statistics have no relation to snapshots or materialized views used in replication. Rather, they are a point-in-time collection of the statistics available through the V$ views, and are given a Snap_ID value to identify the snapshot. You can generate reports on the changes in the statistics between any two snapshots.

To generate a snapshot of the statistics, execute the SNAP procedure of the STATSPACK package, as shown in the following listing. You must be logged in as the PERFSTAT user to execute this procedure.

```
execute STATSPACK.SNAP ;
```

When the SNAP procedure is executed, Oracle populates your SNAPS tables with the current-statistics. You can then query those tables directly, or you can use the standard STATSPACK report.

Snapshots should be taken in one of two ways –

(i) To evaluate performance during specific tests of the system. For these tests, you can execute the SNAP procedure manually.

(ii) To evaluate performance changes over a long period of time. To establish a baseline of the system performance, you may generate statistics snapshots on a scheduled basis. For these snapshots, you should schedule the SNAP procedure execution through Oracle's internal DBMS_JOB scheduler or through an operating system scheduler.

DATA DICTIONARY, DYNAMIC PERFORMANCE VIEW, SECURITY, ROLE MANAGEMENT, PRIVILEGE MANAGEMENT, PROFILES, INVOKER DEFINED SECURITY MODEL, SQL QUERIES

Q.24. With respect to Oracle describe the following –
(i) Data block (ii) Data dictionary (iii) Segments.
(R.G.P.V., Dec. 2010)

Ans. (i) Data Block – Refer to Q.12.

(ii) Data Dictionary – The data dictionary is a collection of database tables, owned by the *SYS* and *SYSTEM* schemas, that contain the metadata about the database, its structures, and the privileges and roles of database users.

A data dictionary is defined as a DBMS component that stores the definition of data characteristics and relationships. The DBMS data dictionary provides the DBMS with its self-describing characteristic. In effect, the data dictionary resembles an X-ray of the company's entire data set, and it is a crucial element in data administration.

There are two main types of data dictionary –

(a) Integrated (b) Stand alone.

An integrated data dictionary is included with the DBMS. For example, all relational DBMSs include a built-in data dictionary or system catalog that is frequently accessed and updated by the RDBMS. Other DBMSs, especially older types, do not have a built-in data dictionary. Instead, the DBA may use third-party standalone data dictionary systems.

Data dictionary can also be classified as active or passive. An *active data dictionary* is automatically updated by the DBMS with every database access, thereby keeping its access information up to date. A *passive data dictionary* is not updated automatically and usually requires running a batch process. Data dictionary access information is normally used by the DBMS for query optimization purposes.

(iii) Segments – Refer to Q.11.

Q.25. What is data dictionary ? What it stores ? How can this information be useful in DBMS. ***(R.G.P.V., May 2018)***

Ans. Refer to Q.24 (ii).

Q.26. Explain the following –

(i) Data dictionary (ii) Table spaces (iii) Segments. ***(R.G.P.V., Dec. 2009)***

Ans. (i) Data Dictionary – Refer to Q.24 (ii).

(ii) Table Spaces – Refer to Q.10.

(iii) Segments – Refer to Q.11.

Q.27. What do you mean by the dynamic performance views ?

Ans. Dynamic performance views are not part of the Oracle dictionary per se, but nevertheless are useful for managing your database. Dynamic performance views are updated constantly by Oracle with important data about database operation. Some examples of dynamic performance views are –

(i) V$DATABASE – It contains information about the database itself, such as the database name and when the database was created.

(ii) V$SYSSTAT – It contains information about the performance of your database.

(iii) V$SESSION, V$SESSTAT – Most information about performance for individual user sessions is stored here.

(iv) VLOG, VLOGFILE – It contains information about online redo logs.

(v) V$DATAFILE – It contains information about Oracle datafiles.

(vi) V$CONTROLFILE – It contains information about Oracle control files.

(vii) V$VERSION – It shows which software version the database is using.

(viii) V$OPTION – It displays which optional components are installed in the database.

(ix) V$SQL – It displays information about the SQL statements that database users have been issuing.

Q.28. Write short note on the security.

Ans. Security means protecting the data against accidental or intentional use by unauthorized users. The security and protection of sensitive information are very important issue. This is handled by a DBMS easily and efficiently. Permissions and access rights can be defined for one user or a group of users so that any unauthorised user cannot update or even read sensitive information.

Oracle makes several levels of security available to the DBA –

(i) Account security for a validation of users

(ii) Access security for database objects

(iii) System-level security for managing global privileges.

Account Security – If you want to access data in an Oracle database, you must have access to an account in that database. This access can be direct through user connections into a database, or indirect. Indirect connections include access through preset authorizations within database links. Each account must have a password associated with it. A database account can also be tied to an operating system account. Passwords are set for a user when user's account is created and may be altered after the account is created. A user's ability to alter the account password will be limited by the tools to which he or she is granted access. The database stores an encrypted version of the password in a data dictionary table. If the account is directly related to an operating system account, it is possible to bypass the password check and rely on the operating system authentication instead.

Object Privileges – Access to objects within a database is enabled through privileges. These allow specific database commands to be used against specific database objects through the *grant* command. For example, if the user KRISHNA owns a table called EMPLOYEE, and executes the command.

grant select on EMPLOYEE to PUBLIC;

Then all users (PUBLIC) will be able to select records from KRISHNA'S EMPLOYEE table. You can create *roles*, named groups of privileges, to simplify the administration of privileges. For applications with large numbers of users, roles greatly reduce the number of *grant* commands needed. Since *roles* can be password protected and can be dynamically enabled or disabled, they add an additional layer of security to the database.

System-level Roles and Privileges – You can use roles to manage the system-level commands available to users. These commands are *create table* and *alter index*. Actions against each type of database object are authorized through separate privileges. For instance, a user may be granted the CREATE TABLE privilege or CREATE INDEX privilege.

Q.29. At what point during query processing does the optimization occur. Discuss the advantages of distributing database. Also explain database security. (R.G.P.V., Dec. 2012)

Ans. **Query Optimization** – Refer to Q.40 (Unit-III).

Advantages of Distributed Database – Refer to Q.19.

Database Security – Refer to Q.28.

Q.30. What is role ? How roles are created and maintained ?

Ans. A role is a tool for administering privileges. Privileges can be granted to a role, and then that role can be granted to other roles and users. Thus, users can inherit privileges through roles.

To create a role, use the CREATE ROLE statement you can optionally include an IDENTIFIED BY clause that requires users to authenticate themselves before enabling the role. Roles requiring authentication are typically used inside an application, where a user's activities are controlled by the application. To create the role APPL_DBA, execute the following –

```
CREATE ROLE appl_data;
```

To enable a role, execute a SET ROLE statement, like this –

```
SET ROLE appl_dba IDENTIFIED BY meow;
```

Q.31. What are privileges ? Give its types.

Ans. Privileges allow a user to access database objects or execute stored programs that are owned by another user. Privileges also enable a user to perform system-level operations, such as connecting to the database, creating a table, or altering the database.

Privileges are assigned to a user, to the special user PUBLIC, or to a role with the GRANT statement and can be rescined with the REVOKE statement.

There are two types of privileges –

(i) System privileges (ii) Object privileges.

(i) System Privileges – System privileges control the creation and maintenance of many database objects, such as rollback segments, synonyms, table, and triggers. In addition, the ability to use the *analyze* command and the Oracle database *audit* capability is governed by system privileges.

There are several sub-categories of system privileges that relate to each object. Those categories determine the scope of ability that the privilege grantee will have. There are following classes or categories of system privileges –

(a) Admin Functions – These privileges relate to activities typically reserved for and performed by the DBA. Privileges include *alter system, audit system, audit any, alter database, analyze any, SYSDBA. SYSOPER*, and *grant any privilege*.

(b) Database Access – These privileges control who accesses the database, when they can access it, and what they can do regarding management of their own session. Privileges include *create session, alter session*, and *restricted session*.

(c) Tablespaces – These privileges are typically reserved for DBAs. Privileges include *create tablespace, alterspace, manage tablespace, drop tablespace*, and *unlimited tablespace*.

(d) Users – These privileges are used to manage users on the Oracle database. These privileges are reserved for DBAs or security administrators. Privileges include *create user, become user, alter user*, and *drop user.*

(e) Undo Segments – These privileges include *create rollback segment, alter rollback segment*, and *drop rollback segment.* If you plan to use automatic undo management, no users need to be granted these privileges.

(f) Tables – These privileges govern which users can create and maintain tables. These privileges include *create table, create any table, alter any table, backup any table, drop any table, lock any table, comment any table, select any table, insert any table, update any table*, and *delete any table.*

(g) Clusters – Clusters are used to store tables commonly used together in close physical proximity on disk. These privileges include *create cluster, create any cluster, alter any cluster*, and *drop any cluster.* The create cluster and create any cluster privileges also enable you to alter and drop those clusters.

(h) Indexes – Indexes are used to improve SQL statement performance on tables containing lots of row data. These privileges include *create any index, alter any index*, and *drop any index.*

(i) Synonyms – A synonym is a database object that enables you to reference another object by a different name. These privileges include *create synonym, create any synonym, drop any synonym, create public synonym*, and *drop public synonym.*

(j) Views – A view is an object containing a SQL statement that behaves like a table in Oracle, except that it stores no data. These privileges include *create view, create any view*, and *drop any view.*

(k) Sequences – A sequence is an object in Oracle that generates numbers according to rules you can define. These privileges include *create sequence, create any sequence, alter any sequence, drop any sequence*, and *select any sequence.*

(*l*) Database Links – These privileges include *create database link, create public database link*, and *drop public database link.*

(m) Roles – Roles are objects that can be used for simplified privilege management. These privileges include *create role, drop any role, grant any role*, and *alter any role.*

(n) Transactions – These privileges are for resolving in-doubt, distributed transactions being processed on the Oracle database. Privileges include *force transaction* and *force any transaction.*

(o) PL/SQL – These privileges enables you to create, run, and manage different types of blocks. The privileges include *create procedure,* *create any procedure, alter any procedure, drop any procedure,* and *execute any procedure.*

(p) Triggers – Triggers are PL/SQL blocks in Oracle that execute when a specified DML activity occurs on the table to which the trigger is associated. These privileges include *create trigger, create any trigger, alter any trigger,* and *drop any trigger.*

(q) Profiles – Profiles are objects in Oracle that enable you to impose limits on resources for users in the machine hosting Oracle. These privileges include *create profile, alter profile, drop profile,* and *alter resource cost.*

(r) Snapshots and Materialized Views – Snapshots are objects in Oracle that enable you to replicate data from a table in one database to a copy of the table in another. These privileges include *create snapshot, create any snapshot, alter any snapshot,* and *drop any snapshot.*

(s) Directories – Directories are objects in Oracle that refer to directories on the machine hosting the Oracle database. They are used to identify a directory that contains objects Oracle keeps track of that are external to Oracle, such as objects of the BFILE type. These privileges include *create any directory* and *drop any directory.*

(t) Types – Types in Oracle correspond to user-defined types you can create in the Oracle 8i objects option. These privileges include create type, create any type, alter an type, drop any type, and execute any type.

(u) Libraries – A library is an object that enables you to reference a set of procedures external to Oracle. These privileges include create library, create any library, alter any library, drop any library, and execute any library.

(ii) Object Privileges – Object privileges govern a user's ability to manipulate database objects owned by other users in the database. Object privileges permit the owner of database objects, such as tables, to administer access to those objects according to the following types of access.

There are following types of object privileges –

(a) SELECT – It permits the grantee of this object privilege to access the data in a table, sequence, view, or snapshot.

(b) INSERT – It permits the grantee of this object privilege to insert data into a table or, in some cases, a view.

(c) UPDATE – It permits the grantee of this object privilege to update data into a table or view.

(d) DELETE – It permits the grantee of this object privilege to delete data from a table or view.

(e) ALTER – It permits the grantee of this object privilege to alter the definition of a table or sequence only. The *alter* privileges on all other database objects are considered system privileges.

(f) INDEX – It permits the grantee of this object privilege to create an index on a table already defined.

(g) REFERENCES – It permits the grantee to create or alter a table in order to create a foreign key constraint against data in the referenced table.

(h) EXECUTE – It permits the grantee to run a stored procedure or function.

Q.32. What is the profile ?

Ans. A profile is a named collection of settings that control how much of the database resource a given user can use. By specifying profiles, the DBA can limit how much storage space a user can use, how long a user can be connected, how much idle time may be used before the user is disconnected, and so on. In an ideal world, all users would have unlimited access to all resources at all times, but in the real world, such access is neither possible nor desirable.

In other words, profiles are a bundled set of resource-usage parameters that the DBA can set in order to limit the users overall host machine utilization. A driving idea behind their use is that many end users of the system only need a certain amount of the host machine's capacity during their session. To reduce the chance that one user could affect the overall database performance with, say, a poorly formulated adhoc report that drags the database to its knees, you may assign profiles for each user that limit the amount of time they can spend on the system.

Q.33. How to combine the definer and invoker rights model ?

Ans. Invoker rights programs allow central code to reflect back to the calling schema. Definer rights programs allow remote schemas to access local data (i.e., data in the same schema as the program). Many applications require a combination of these flavours.

Suppose, for example, that the national Stolen Lives Project also maintains a table of 'perpetrators', law enforcement officers who have killed one or more people in the United States. Due to the sensitivity of the information, the SLP has decided to maintain a single headquarters table that cannot be accessed directly by the city/town schemas. Yet both the location-specific stolen_life table and the systemwide perpetrators table need to be accessed by the check_city_statistics procedure.

What's a code architect to do ? One thought might be to create a public synonym for the perpetrators table and make sure that no city schema has its own perpetrators table. When the city schema runs the central code under invoker rights, the reference to perpetrators would, in fact, be to that central source of data.

That works fine for the check_city_statistics procedure, but what about the rest of the application ? With this approach, any city schema can directly access the perpetrators table, a violation of security. So the synonyms solution is no solution at all.

With Oracle 8i, however, you don't need to do anything more than introduce a layer of code around the shared data structure. You need to do at least that, however, so that you can *change* the model used for resolving external references. If the perpetrators table is accessed directly by the check_city_statistics procedure, the reference can only be resolved by the city schema's having direct access (via a synonym) to the table. The check_city_statistics procedure cannot, therefore, query the perpetrators table directly. Instead, as shown in fig. 5.10, it will call another procedure, compiled under the definer rights model, which, in turn, works with the perpetrators table.

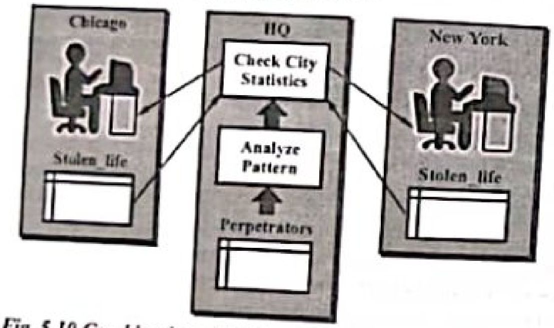


Fig. 5.10 Combination of Definer and Invoker Rights Model

The *authid4.sql* file provides an implementation that reflects this blended approach. It creates a separate procedure, show_perps, to access the perpetrators table.

```
/*Filename on companion disk: authid4.sql*/
CREATE OR REPLACE PROCEDURE show_perps (loc IN VARCHAR2)
AUTHID DEFINER
AS
BEGIN
   FOR rec IN (SELECT*FROM perpetrator WHERE location = loc)
   LOOP
      pl (loc || 'perpetrator is' || rec.rank || '  ' || rec.name);
   END LOOP ;
END,
```

After granting PUBLIC access to this procedure (but not to the table), modify show_descriptions to include this information –

```
CREATE OR REPLACE PROCEDURE show_descriptions
   AUTHID CURRENT_USER
AS
BEGIN
```

```
   HQ.show_perps (USER);
   pl (' ');
   FOR lifestolen IN (SELECT * FROM stolen_life)
   Loop
      show_victim (lifestolen);
   END LOOP;
END;
```

By offering both the definer and invoker rights models in Oracle8i, Oracle demonstrates its continuing commitment to the PL/SQL language.

The invoker rights model gives all of us another tool to use as we construct our applications. By coming up with a simple syntax for applying this model, Oracle makes it easier for us to learn and implement this new approach.

Q.34. How the "GROUP BY" clause works ? ***(R.G.P.V., Dec. 2015)***

Or

Define the GROUP BY clause in context of SQL. ***(R.G.P.V., Dec. 2017)***

Ans. The GROUP BY clause takes the form –

GROUP BY < column name commalist >

The < column name commalist > must not be empty. Let T be the result of evaluating the immediately preceding FROM clause and WHERE clause (if any). Each <column name> mentioned in the GROUP BY clause must be optionally qualified name of a column of T. The result of the GROUP BY clause is a grouped table – i.e., a set of groups of rows, derived from T by conceptually rearranging it into the minimum number of groups such that within any group all rows have the same value for the combination of columns identified by the GROUP BY clause.

Q.35. Define the distinct clause in context of SQL. ***(R.G.P.V., Dec. 2017)***

Ans. Distinct clause is used to eliminate the redundancy of rows in the output. Mean the output has not duplicate rows. For example –

SELECT DISTINCT S.name, S.class

FROM Students S

name	*class*
Ram	5
Raju	8
Deepak	10
Shubham	9
Manoj	7
Shyam	11
Manoj	7
Tyson	6

without DISTINCT

name	*class*
Ram	5
Raju	8
Deepak	10
Shubham	9
Manoj	7
Shyam	11
Tyson	6

with DISTINCT

The difference between both query, is that, when we do not use DISTINCT in query student Manoj class 7 display two times but when we use DISTINCT in query student Manoj class 7 does not display two times. Hence the duplicate row has removed in table.

Q.36. Define the ORDER BY clause in context of SQL.
(R.G.P.V., Dec. 2017)

Ans. The ORDER BY clause is used to sort the data in order. The clause works on a particular column of the given table. Columns can be sorted in ascending order or descending order. For sorting in ascending order use (ASC) keyword and for sorting in descending order use (DESC) keyword. By default the order is in (ASC).

In terms of structure query language (SQL) the query is written to fetch the data from the given table using order by clause.

For Example – Consider a table named employee having column emp_id, emp_name, emp_desg and emp_sal. Now write an SQL query to fetch the record in ascending order consider the column as emp_sal.

emp_id	emp_ name	emp_ desg.	emp_ sal
emp_01	John	HR Manager	50,000
emp_02	SAM	Analyst	30,000
emp_03	Joseph	Developer	40,000

Now, using SQL query according to the given condition.

```
Select * from employee order by (emp_sal);
```

Output

emp_id	emp_ name	emp_desg.	emp_sal
emp_02	SAM	Analyst	30,000
emp_03	Joseph	Developer	40,000
emp_01	John	HR Manager	50,000

Q.37. Explain the sequence feature of ORACLE.
(R.G.P.V., June 2008, Dec. 2008, 2010)

Ans. A sequence is a special provision of a data type in oracle for attribute value generation. An attribute may derive its value from a sequence, which is an automatically generated internal number. The same sequence may be used for one or more tables. For example, an attribute EMPIO for the EMPLOYEE table may be internally generated as a sequence.

The SQL statement used for creating a sequence is –

```
CREATE SEQUENCE sequence_name
INCREMENT BY integervalue
START WITH integervalue
MAXVALUE integervalue/NOMAXVALUE
MINVALUE integervalue/NOMINVALUE
CYCLE/NOCYCLE
CACHE integervalue/NOCACHE
ORDER/NOORDER
```

Example – Create a sequence by the name order_seq which will generate number from 1 upto 9999 in ascending order with an interval of 1. The sequence must start from the number 1 after generating number 9999.

```
CREATE SEQUENCE order_seq
INCREMENT BY 1
START WITH 1
MINVALUE 1
MAXVALUE 9999
CYCLE;
```

Q.38. Explain the SQL Loader feature of ORACLE.
(R.G.P.V., Dec. 2008, 2010)

Ans. Oracle has a direct load utility, *SQL Loader*, that supports fast parallel loads of large amounts of data from external files. It supports a variety of data formats and it can perform various filtering operation such as control file and data file as the input on the data being loaded.

DATA EXTRACTION FROM SINGLE, MULTIPLE TABLES EQUI-JOIN, NON EQUI-JOIN, SELF-JOIN, OUTER-JOIN, USAGE OF LIKE, ANY, ALL, EXISTS IN SPECIAL OPERATORS, HIERARCHICAL QUERIES, INLINE QUERIES, FLASHBACK QUERIES

Q.39. What do you mean by equi-join and non equi-join ?

Ans. Refer to Q.54 (Unit-II).

Q.40. Explain self join.

Ans. A self-join is a query in which a table is joined (compared) to itself. Self-joins are used to compare values in a column with other values in the same column in the same table.

One practical use for self-joins is obtaining running counts and running totals in an SQL query.

To write the query, select from the same table listed twice with different aliases, set up the comparison, and eliminate cases where a particular value would be equal to itself.

We can use a self-join to simplify nested SQL queries where the inner and outer queries reference the same table. These joins allow us to retrieve related records from the same table. The most common case where we would use a self-join is when you have a table that references itself, such as the employees table shown below –

```
id first_name last_name manager
-------- ---------------- ------------------ ----------------
1  Pat Crystal NULL
2  Dennis Miller 1
3  Jacob Smith 1
4  Allen Hunter 2
5  Mary Underwood 3
6  Joy Needham 3
```

In this table, the manager attributes simply references the employee ID of another employee in the same table. For example, Dennis Miller reports to Pat Crystal. Pat is apparently the president of this company, as she reports to no one.

Suppose, we are tasked with writing a SQL query to retrieve a list of employees and their managers. We cannot write a basic SQL SELECT statement to retrieve this information, as we need to cross reference information contained in other records within the same table. Fortunately, we can use a self-join to solve this dilemma by joining the table to itself.

Here is the SQL statement that will retrieve the desired results –

```
SELECT e.first_name AS 'Employee FN', e.last_name AS 'Employee
LN', m.first_name AS 'Manager FN', m.last_name AS 'Manager LN'
FROM employees AS e LEFT OUTER JOIN employees AS m
ON e.manager = m.id
```

And the corresponding output –

```
Employee FN Employee LN Manager FN Manager LN
------------- --------------- ---------------- -----------------
Pat Crystal NULL NULL
Dennis Miller Pat Crystal
Jacob Smith Pat Crystal
Allen Hunter Dennis Miller
Mary Underwood Jacob Smith
Joy Needham Jacob Smith
(6 row(s) affected)
```

Notice that it is extremely important to select the correct join type when writing a self-join. In this case, we used a LEFT OUTER JOIN to ensure we had output records corresponding to each employee. If we used an INNER JOIN instead, we would have omitted Pat Crystal, the company president, from our list, as she does not have a manager.

Q.41. What are inner joins and outer joins ?

Ans. An inner join is the traditional join in which only rows that meet a given criteria are selected. An outer join returns the matching rows as well as the rows with unmatched attribute values for one table or both tables to be joined.

Q.42. Write short notes on –
(i) RDBMS (ii) Distributed database (iii) Self join and outer join. (R.G.P.V., Dec. 2012)

Ans. (i) *RDBMS* – Refer to Q.1.

(ii) *Distributed Database* – Refer to Q.15.

(iii) *Self Join* – Refer to Q.40.

Outer Join – Refer to Q.41.

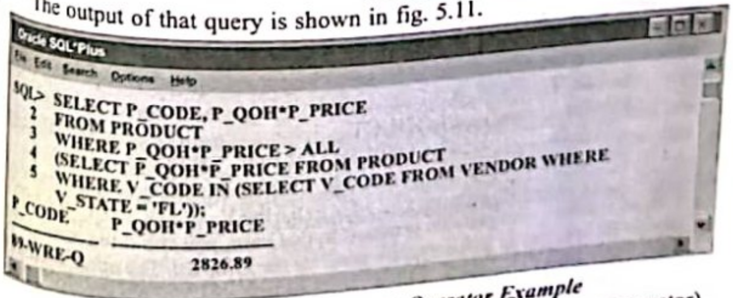
Q.43. Explain the usage of ANY and ALL special operators.

Ans. The use of the ALL operator allows us to compare a single value with a list of values returned by the first subquery, using a comparison operator (> or <) other than equals.

For example, suppose you want to know what products have a product cost that is greater than all individual product costs for products provided by vendors from Florida.

```
SELECT      P_CODE, P_QOH * P_PRICE
FROM        PRODUCT
WHERE       P_QOH * P_PRICE > ALL (SELECT
            P_QOH * P_PRICE
            FROM      PRODUCT
            WHERE     V_CODE IN (SELECT V_CODE
                      FROM VENDOR
                      WHERE V_STATE = 'FL'));
```

The output of that query is shown in fig. 5.11.



```
SQL> SELECT P_CODE, P_QOH*P_PRICE
     FROM PRODUCT
     WHERE P_QOH*P_PRICE > ALL
     (SELECT P_QOH*P_PRICE FROM PRODUCT
     WHERE V_CODE IN (SELECT V_CODE FROM VENDOR WHERE
     V_STATE = 'FL'));
P_CODE      P_QOH*P_PRICE
----------  -------------
89-WRE-Q          2826.89
```

Fig. 5.11 Multirow Subquery Operator Example

The ANY multirow operator (near cousin of the ALL multirow operator). The ANY multirow operator will allow you to compare a single value to a list of values, selecting only the rows for which the inventory cost is greater than any value of the list or less than any value of the list. We can use the equal to ANY operator, which will be the equivalent of the IN operator.

Q.44. Explain the usage of LIKE, IN and EXISTS special operator.

Ans. **LIKE** – The LIKE special operator is used in conjunction with wildcards to find patterns within string attributes. Standard SQL allows you to use the percent sign (%) and underscore (_) wildcard character to make matches when the entire string is not known.

% means any and all following characters are eligible. For example, 'J%' includes Johnson, Jones, Jernigan, Juley and J-231Q 'J0%' includes Johnson and Jones.

Underscore (_) means any one character may be substituted for the underscore. For example, '_23 – 456 – 6789' includes 123 – 456 – 6789, 223 – 456 – 6789 and 323 – 456 – 6789 '_23 – _56 – 678_' includes 123 – 156 – 6781, 123 – 256 – 6782, and 823 – 956 – 6788.

For example, the following query would find all VENDOR rows for contacts whose last names begin with Smith.

```
SELECT      V_NAME, V_CONTACT, V_AREACODE,
            V_PHONE
FROM        VENDOR
WHERE       V_CONTACT LIKE 'Smith%';
```

IN – IN operator uses a value list. All of the values in the list must be of the same data type. Each of the values in the value list is compared to the attribute.

For example –

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN (21344, 24288);
```

Here, if the V_CODE value matches any of the values in the list, the row is selected. In this example, the rows selected will be only those in which the V_CODE is either 21344 or 24288.

If the attribute used is of a character data type, the list values must be enclosed in single quotation marks. For instance, if the V_CODE had been defined as CHAR(5) during the table-creation process, the preceding query would have read,

```
SELECT      *
FROM        PRODUCT
WHERE       V_CODE IN ('21344', '24288');
```

The IN operator is especially valuable when it is used in conjunction with subqueries. For example, suppose you want to list the V_CODE and V_NAME of only those vendors who provide product. In that case, you could use a subquery within the IN operator to automatically generate the value list. The query would be

```
SELECT      V_CODE, V_NAME
FROM        VENDOR
WHERE       V_CODE IN (SELECT V_CODE
            FROM PRODUCT);
```

EXISTS – EXITS can be used whenever there is a requirement to execute a command based on the result of another query. That is, if a subquery returns any rows, run the main query. Otherwise, don't. For example, the query will list all vendors, but only if there are products to order –

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT
            WHERE P_QOH <= P_MIN);
```

The EXISTS special operator is used in the following example to list all vendors, but only if there are products with the quantity on hand, less than double the minimum quantity –

```
SELECT      *
FROM        VENDOR
WHERE       EXISTS (SELECT * FROM PRODUCT
            WHERE P_QOH < P_MIN * 2);
```

Q.45. What are hierarchical queries ?

Ans. A hierarchical query is a type of SQL query that handles hierarchical model data, and returns the rows of the result set in a hierarchical order based upon data forming a parent-child relationship. A hierarchy is typically represented by an inverted tree structure. The tree is comprised of interconnected nodes. Each node may be connected to none, one, or multiple child nodes. Each node is connected to one parent node except for the top node which has no parent. This node is the root node. Each tree has exactly one root node. Nodes that don't have any children are called leaf nodes. A tree always has at least one leaf node.

In a hierarchical query the rows of the result set represent the nodes of one or more trees. It is possible that a single, given row may appear in more than one tree and thus appear more than once in the result set.

The hierarchical relationship in a query is described by the CONNECT BY clause which forms the basis of the order in which rows are returned in the result set. The context of where the CONNECT BY clause and its associated optional clauses appear in the SELECT command is shown below.

```
SELECT select_list
FROM table_expression
[WHERE ...]
[START WITH start_expression]
CONNECT BY {PRIOR parent_expr = child_expr | child_expr = PRIOR
                        parent_expr}
[ORDER SIBLINGS BY column1 [ASC|DESC] [, column2 [ASC|DESC]] ...
[GROUP BY ...]
[HAVING ...]
[other ...]
```

Q.46. Explain inline subqueries with an example.

Or

What is inline queries ? Where it is used ? ***(R.G.P.V., Dec. 2015)***

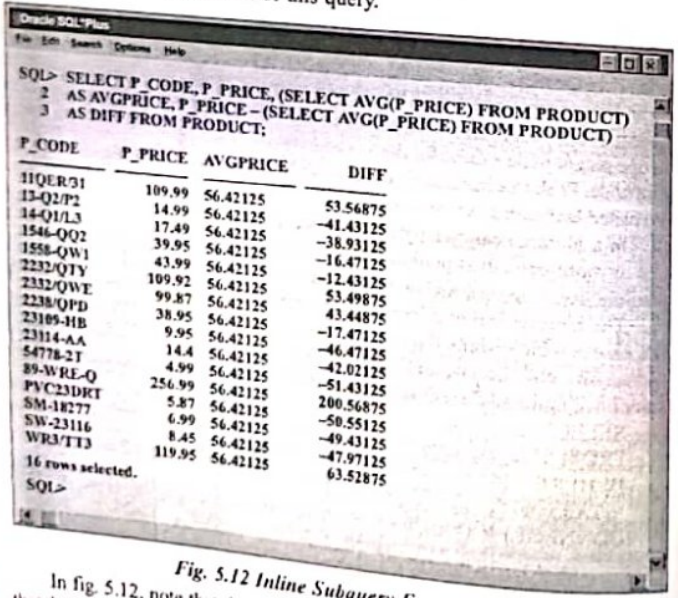
Or

What is inline queries ? ***(R.G.P.V., June 2016)***

Ans. The SELECT statement uses the attribute list to indicate what columns to project in the resulting set. Those columns can be attributes of base tables or computed attributes or the result of an aggregate function. The attribute list can also include a subquery expression, known as an *inline subquery*. A subquery in the attribute list must return one single value, otherwise an error code is raised. For example, a simple inline query to list the difference between each product's price and the average product price is as follows –

SELECT P_CODE, P_PRICE, (SELECT AVG (P-PRICE) FROM PRODUCT) AS AVGPRICE, P_PRICE – (SELECT AVG(P_PRICE) FROM PRODUCT) AS DIFF FROM PRODUCT;

Fig. 5.12 shows the result of this query.



```
SQL> SELECT P_CODE, P_PRICE, (SELECT AVG(P_PRICE) FROM PRODUCT)
  2  AS AVGPRICE, P_PRICE – (SELECT AVG(P_PRICE) FROM PRODUCT)
  3  AS DIFF FROM PRODUCT;

P_CODE      P_PRICE  AVGPRICE        DIFF
--------   -------- ---------  ----------
11QER/31     109.99  56.42125
13-Q2/P2      14.99  56.42125    53.56875
14-Q1/L3      17.49  56.42125   -41.43125
1546-QQ2      39.95  56.42125   -38.93125
1558-QW1      43.99  56.42125   -16.47125
2232/QTY     109.92  56.42125   -12.43125
2332/QWE      99.87  56.42125    53.49875
2238/QPD      38.95  56.42125    43.44875
23109-HB       9.95  56.42125   -17.47125
23114-AA       14.4  56.42125   -46.47125
54778-2T       4.99  56.42125   -42.02125
89-WRE-Q     256.99  56.42125   -51.43125
PVC23DRT       5.87  56.42125   200.56875
SM-18277       6.99  56.42125   -50.55125
SW-23116       8.45  56.42125   -49.43125
WR3/TT3      119.95  56.42125   -47.97125
16 rows selected.                63.52875
SQL>
```

Fig. 5.12 Inline Subquery Example

In fig. 5.12, note that the inline query output returns one single value and that the value is the same in every row. Also, that the query used the full expression instead of the column aliases when computing the difference. In fact, if you try to use the alias in the difference expression, you will get an error message.

Q.47. Explain flashback query with an example.

Or

What is flashback queries ? Where it is used ? ***(R.G.P.V., Dec. 2014)***

Ans. Flashback query allows us to query data in one or more tables in a SELECT query as of a time in the past. Any changes to data in a table generates undo (or optionally data in a Flashback Data Archive), which can give us a snapshot of the entire database down to the granularity of a transaction.

Flashback query uses the AS OF clause to specify the previous point in time as a timestamp or System Change Number (SCN). For example, the user HR is cleaning up the EMPLOYEE table and deletes two employees who no longer work for the company –

```
SQL > delete from employee
2       where employee_id in (5, 6);
2 rows deleted.
SQL > commit;
Commit complete.
SQL >
```

Normally, the user HR will copy these rows to the EMPLOYEE_ARCHIVE table first, but she forgot to do that this time. The user HR does not need to put those rows back into the EMPLOYEE table, but she needs to put the two deleted rows into the archive table. Since the user HR knows she deleted the rows less than an hour ago, she can use a relative timestamp value with Flashback query to retrieve the rows –

```
SQL > insert into hr.employee_archive
2       select * from hr.employee
3       as of timestamp systimestamp – interval '60' minute
4       where hr.employee.employee_id not in
5       (select employee_id from hr.employee);
2 rows created
SQL > commit;
commit complete.
SQL >
```

Q.48. Explain hierarchical queries, inline queries and flashback queries. ***(R.G.P.V., Dec. 2013)***

Ans. (i) ***Hierarchical Queries*** – Refer to Q.45.

(ii) ***Inline Queries*** – Refer to Q.46.

(iii) ***Flashback Queries*** – Refer to Q.47.

INTRODUCTION OF ANSI SQL, PL SQL, ANONYMOUS BLOCK, NESTED ANONYMOUS BLOCK, BRANCHING AND LOOPING CONSTRUCTS IN ANSI SQL, CURSOR MANAGEMENT – NESTED AND PARAMETERIZED CURSORS

Q.49. Give the definiton of ANSI SQL.

Ans. We know that SQL is easy to learn. SQL is a nonprocedural language. Generally we command what is to be done; we need not to worry about how it is to be done. It means that we have not to follow the procedure.

A standard SQL was given by American National Standards Institute (ANSI) - the most recent version is SQL-99 or SQL3. It is also known as ANSI SQL. The standards of ANSI SQL are also accepted by the International Organization for Standardization (ISO). Although adherence to the ANSI/ISO SQL standard is usually required in commercial and government contract database specificaitons, many RDBMS vendors add their own special enhancements. Generally, it is possible to move a SQL-based application from one RDBMS to another without making some changes.

Q.50. What are the different constraints of ANSI SQL ?

Ans. It is difficult to maintain referential integrity and entity integrity rules in a relational database environment. But most of the SQL implementations support both integrity rules. When a primary key is specified in the CREATE TABLE command sequence, the entity integrity is enforced automatically.

For Example – If we create the VENDOR table structure and set the stage for the enforcement of entity integrity rules by using –

PRIMARY KEY (V_CODE)

As we see the PRODUCT table's CREATE TABLE sequence, note that referential integrity has been enforced by specifying in the PRODUCT table –

FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE

This foreign key constraint definition ensures that –

(i) A vendor cannot be deleted from the VENDOR table if at least one product row references that vendor.

(ii) If a change is made in an existing VENDOR table's V_CODE, that change must be reflected in any PRODUCT table V_CODE reference (ON UPDATE CASCADE) automatically. In other words, ON UPDATE CASCADE specification ensures the preservation of referential integrity. Generally, ANSI SQL permits the use of ON DELETE and ON UPDATE clauses to cover the CASCADE, SET NULL, or SET DEFAULT actions, but ORACLE does not support ON UPDATE CASCADE.

Besides the PRIMARY KEY and FOREIGN KEY constraints, the ANSI SQL standard also defines the following constraints –

(i) The NOT NULL constraint ensures that a column does not accept nulls.

(ii) The UNIQUE constraint ensures that all the entries in a column are unique.

(iii) The DEFAULT constraint assigns a value to an attribute when a new row is added to a table. The end user may also enter a value other than the default value.

(iv) The CHECK constraint validates data when an attribute value is entered.

Q.51. What is procedural SQL (PL/SQL) ? Also Explain the anonymous block.

Ans. **PL/SQL** – Procedural SQL (PL/SQL) is a language that makes it possible to use and store procedural code and SQL statements within the database. Procedural SQL makes it possible to merge SQL and traditional programming constructs such as variables, conditional processing (IF-THEN-ELSE), basic loops (FOR and WHILE Loops) and error trapping. The procedural code is executed as a unit by the DBMS when it is invoked (directly or indirectly) by the end user. End users can use PL/SQL to create –

(i) Anonymous PL/SQL blocks
(ii) Trigger
(iii) Stored procedures
(iv) PL/SQL functions.

The basic structure of a PL/SQL statement is –

```
DECLARE
    Declare Variables, Constants, Functions and Procedures;
BEGIN
    Define SQL Block;
EXCEPTION
    Exception Expression
END;
```

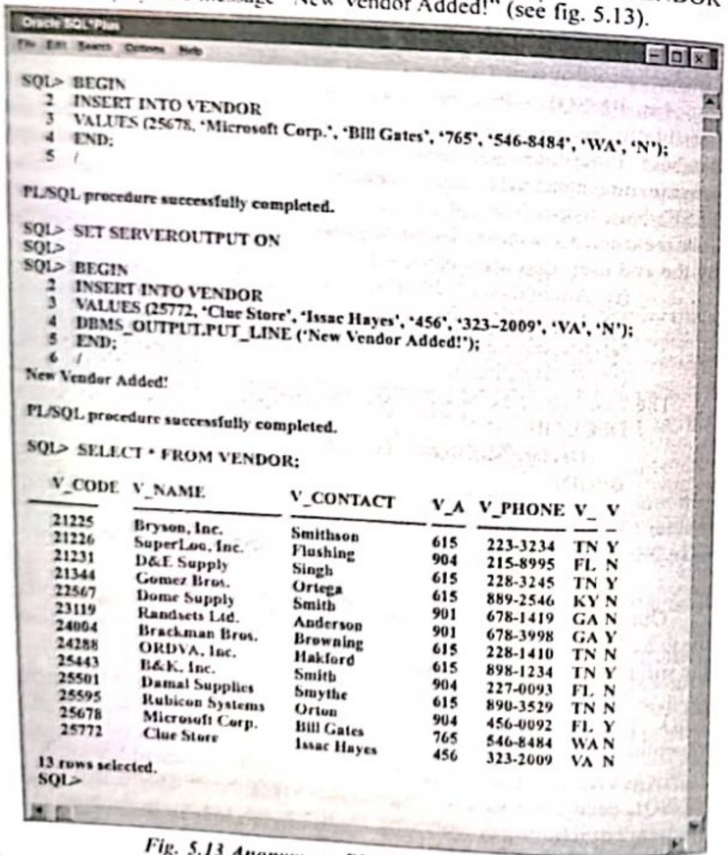
Out of these four clauses DECLARE, BEGIN, EXCEPTION, and END, the two compulsory clauses are BEGIN and END. The DECLARE block defines the variables and their data types used in the SQL block. The BEGIN statement defines the start of the SQL block and END statement defines the end of the block. The EXCEPTION statement allows the developer to capture all the exceptions and handle them with proper error messages.

Anonymous PL/SQL Blocks – Using Oracle SQL*Plus, you can write a PL/SQL code block by enclosing the commands inside BEGIN and END clauses. For example, the following PL/SQL block inserts a new row in the VENDOR table (see fig. 5.13).

```
BEGIN
    INSERT INTO VENDOR
    VALUES (25678, 'Microsoft Corp.',
    'Bill Gates', '765', '546-8484', 'WA', 'N');
END;
```

The PL/SQL block shown in fig. 5.13 is known as an anonymous PL/SQL block because it has not been given a specific name.

The following anonymous PL/SQL block inserts a row in the VENDOR table and displays the message "New Vendor Added!" (see fig. 5.13).



```
SQL> BEGIN
  2  INSERT INTO VENDOR
  3  VALUES (25678, 'Microsoft Corp.', 'Bill Gates', '765', '546-8484', 'WA', 'N');
  4  END;
  5  /

PL/SQL procedure successfully completed.

SQL> SET SERVEROUTPUT ON
SQL>
SQL> BEGIN
  2  INSERT INTO VENDOR
  3  VALUES (25772, 'Clue Store', 'Issac Hayes', '456', '323-2009', 'VA', 'N');
  4  DBMS_OUTPUT.PUT_LINE ('New Vendor Added!');
  5  END;
  6  /
New Vendor Added!

PL/SQL procedure successfully completed.

SQL> SELECT * FROM VENDOR;

  V_CODE V_NAME            V_CONTACT      V_A  V_PHONE  V_ V
-------- ----------------- -------------- ---  -------- -- -
   21225 Bryson, Inc.      Smithson       615  223-3234 TN Y
   21226 SuperLoo, Inc.    Flushing       904  215-8995 FL N
   21231 D&E Supply        Singh          615  228-3245 TN Y
   21344 Gomez Bros.       Ortega         615  889-2546 KY N
   22567 Dome Supply       Smith          901  678-1419 GA N
   23119 Randsets Ltd.     Anderson       901  678-3998 GA Y
   24004 Brackman Bros.    Browning       615  228-1410 TN N
   24288 ORDVA, Inc.       Hakford        615  898-1234 TN Y
   25443 B&K, Inc.         Smith          904  227-0093 FL N
   25501 Damal Supplies    Smythe         615  890-3529 TN N
   25595 Rubicon Systems   Orton          904  456-0092 FL Y
   25678 Microsoft Corp.   Bill Gates     765  546-8484 WA N
   25772 Clue Store        Issac Hayes    456  323-2009 VA N

13 rows selected.
SQL>
```

Fig. 5.13 Anonymous PL/SQL Block Examples

The following example of an anonymous PL/SQL block demonstrates of the constructs supported by the procedural language.

```
DECLARE
W_P1 NUMBER (3) : = 0;
W_P2 NUMBER (3) : = 10;
W_NUM NUMBER (2) = 0;
BEGIN
WHILE W_P2 < 300 LOOP
SELECT COUNT (P_CODE) INTO W_NUM
FROM PRODUCT
WHERE P_PRICE BETWEEN
W_P1 AND W_P2;
DBMS_OUTPUT.PUT_LINE ('There are '|| W_NUM ||' Products with
price between '|| W_P1 ||' and '||W_P2);
 W_P1 : = W_P2 + 1 ;
 W_P2 : = W_P2 + 50;
 END LOOP;
 END ;
```

Q.52. Explain the advantages of PL/SQL. What do you mean by database security? What the objective while designing secure database?

(R.G.P.V., Dec. 2016)

Ans. The advantages of PL/SQL are as follows –

(i) The main advantage of PL/SQL is its capability to define and implement 3GL constructs with embedded SQL statements.

(ii) Programs written in PL/SQL are hardware independent and operating system independent.

(iii) Oracle has integrated SQL and PL/SQL. Programs written in PL/SQL can make use of almost all SQL features.

(iv) PL/SQL enables you to write programs as independent modules that you can integrate. You can implement this modularity by using features such as procedures, functions and packages.

(v) PL/SQL provides object oriented features. Various features introduced in PL/SQL enable object orientation to be defined and implemented in PL/SQL.

(vi) PL/SQL consists of blocks of code which can be nested within each other. Blocks can be stored in the database and reused.

(vii) PL/SQL engine process multiple SQL statement simultaneously as a single block thereby reducing network traffic.

(viii) PL/SQL handles errors or exceptions effectively during the execution of a PL/SQL program.

Security – Refer to Q.28.

Some of the objectives to be considered in the database design are as follows –

(i) The database design should ensure that there is no data redundancy.

(ii) There is a integrity in the database, it means the database should be as accurate as possible.

(iii) The database should not allow unauthorized access to files.

(iv) Given a piece of hardware on which the database will run and a piece of software to run it, the design should make full and efficient use of the facilities provided.

(v) The conceptual model should be simple and effective so that mapping from conceptual model to logical model is easy.

(vi) The database should not be implemented in a rigid way because changes will occur. The database must be capable of responding readily to such change.

(vii) The database should be safe from physical corruption.

Q.53. Discuss branching and looping constructs in PL/SQL.

Or

Discuss the control-of-flow statements provided by PL/SQL.

Or

Write the branching and looping constructs with example.

(R.G.P.V., June 2016)

Ans. Control-of-flow language controls the flow of execution of SQL statements in batches, stored procedures, triggers and transactions. It is typically used when statements have to be conditionally or repeatedly executed. The control of flow statements provided by PL/SQL for programming are –

Branching Constructs in PL/SQL – PL/SQL allows the use of IF statement to control the execution of a block of code. In PL/SQL, the IF-THEN-ELSIF-ELSE-END IF construct in code blocks allow specifying certain conditions under which a specific block of code should be executed.

Syntax –

```
IF <condition> THEN
        <Action>
ELSIF <condition> THEN
        <Action>
ELSE
        <Action>
END IF ;
```

For example, consider the table Accounts given in table 5.6. Then, a PL/SQL code block that will accept an account number from the user and debit an amount of Rs. 2000 from the account if the account has a minimum balance of 500 after the amount is debited, is as follows –

```
DECLARE
        acct_balance number (11, 2);
        acct_no varchar 2 (6);
        debit_amt number (5) : = 2000;
        min_bal constant number (5, 2) : = 500.00
BEGIN
        acct_no = &acct_no;
           SELECT bal INTO acct_balance FROM accounts
                        WHERE account_id = acct no;
           acct_balance : = acct_balance–debit_amt;
        IF acct_balance > = min_balance THEN
        UPDATE accounts SET bal = bal – debit_amt
                        WHERE account_id = acct_no;

        END IF;
END;
```

Table 5.6 Accounts

Account_ID	Name	Bal
AC001	Anuj	5000
AC002	Robert	10000
AC003	Mita	5000
AC004	Sunita	15000

Looping Constructs in PL/SQL – PL/SQL supports the following structures for iterative control –

(i) The while Loop – The syntax is as follows –

```
WHILE      <condition>
LOOP
              <Action>
END LOOP;
```

For example, a PL/SQL code block to calculate the area of a circle for a value of radius varying from 3 to 7 and store the radius and the corresponding values of calculated area in a table areas having columns Radius and Area, is as follows –

```
DECLARE
   pi constant number (4, 2) : = 3.14;
      radius number (5);
      area number (4, 2);
BEGIN
   radius : = 3 ;
   WHILE radius < = 7
   LOOP
         area : = pi * power (radius, 2);
         INSERT INTO areas VALUES (radius, area);
      radius = radius + 1
   END LOOP;
END;
```

(ii) The FOR LOOP – The syntax is as follows –

```
FOR variable IN [REVERSE] start.....end
   LOOP
      <Action>
   END LOOP;
```

Consider, for example, a PL/SQL block of code for inverting a number 5639 to 9365, is given below –

```
DECLARE
   given_number varchar(5) : = '5639';
   str_length number(2);
   inverted_number varchar (5);
BEGIN
   str_length : = length (given_number);
   For entr IN REVERSE 1....str_length
      LOOP
         inverted_number : = inverted_number || substr
                                   (given_number, cntr, 1);
   END LOOP  dbms_output.put_line ('The Given number is' ||
             given_number);
             dbms_output.put_line ('The Inverted number is
             || inverted_number);
END;
Output –     The Given number is 5639
             The Inverted number is 9365
```

Q.34. Explain cursors in PL/SQL.

Or

Explain the importance of cursors. How are they declared, opened and closed? Explain with example. ***(R.G.P.V., Dec. 2009)***

Or

What is cursors ? ***(R.G.P.V., June 2016, Dec. 2016)***

Ans. The set of rows returned by a query can consist of zero, one, or multiple rows, depending on how many rows meet the search criteria. When query returns multiple rows, it is necessary to explicitly declare a **cursor** to process the rows. A cursor is a special construct used in PL/SQL to hold the rows returned by an SQL query. A cursor can be thought of as a reserved area of memory in which the output of the query is stored, like an array holding columns and rows. Cursors are held in a reserved memory area in the DBMS server, not in the client computer.

There are two types of cursors – implicit and explicit. An implicit cursor is automatically created in procedural SQL when the SQL statement returns only one value. An explicit cursor is created to hold the output of an SQL statement that may return two or more rows. The syntax to create an explicit cursor inside a PL/SQL DECLARE section is as follows –

```
CURSOR cursor_name IS select-query;
```

Once you have declared a cursor, you can use specific PL/SQL cursor processing commands OPEN, FETCH, and CLOSE anywhere between the BEGIN and END keywords of the PL/SQL block. Table 5.7 summarizes the main use of each of those commands.

Table 5.7 Cursor Processing Commands

Cursor Command	*Explanation*
OPEN	Opening the cursor executes the SQL command and populates the cursor with data, opening the cursor for processing. The cursor declaration command only reserves a named memory area for the cursor; it doesn't populate the cursor with the data. Before you can use a cursor, you need to open it. For example – OPEN *cursor_name*
FETCH	Once the cursor is opened, you can use the FETCH command to retrieve data from the cursor and copy it to the PL/SQL variables for processing. The syntax is – FETCH cursor_name INTO variable1 [, variable2, ...] The PL/SQL variables used to hold the data must be declared

	in the DECLARE section and must have data types compatible with the columns retrieved by the SQL command. If the cursor's SQL statement returns five columns, there must be five PL/SQL variables to receive the data from the cursor. This type of processing resembles the "one-record-at-a-time" processing used in previous database models. The first time you fetch a row from the cursor, the first row of data from the cursor is copied to the PL/SQL variables. the second time you fetch a row from the cursor, the second row of data is placed in the PL/SQL variables, and so on.
CLOSE	The CLOSE command closes the cursor for processing.

There are four cursor attributes, which are as follows –

(i) % ISOPEN returns TRUE if the cursor is already open.

(ii) % FOUND returns TRUE if the last FETCH returned a row, and returns FALSE if the last FETCH failed to return a row.

(iii) % NOTFOUND is the logical opposite of % FOUND.

(iv) %ROWCOUNT yields the number of rows fetched.

The following example displays the SSN of employees whose salary is greater than their supervisor's salary –

```
DECLARE
    emp_salary NUMBER;
    emp_super_salary NUMBER;
    emp_ssn CHAR(9);
    emp_superssn CHAR(9);
    CURSOR salary_cursor IS
        SELECT ssn, salary, superssn FROM employee;
    BEGIN
        OPEN salary_cursor;
        LOOP
        FETCH salary_cursor INTO emp_ssn, emp_salary, emp_superssn;
        EXIT WHEN salary_cursor % NOT FOUND;
        IF emp_superssn is NOT NULL THEN
            SELECT salary INTO emp_super_salary
            FROM employee WHERE ssn = emp_superssn;
            IF emp_salary > emp_super_salary THEN
            dbms_output.put_line (emp_ssn);
            END IF;
        END IF;
    END LOOP;
    IF salary_cursor%ISOPEN THEN CLOSE salary_cursor;
```

```
EXCEPTION
    WHEN NO_DATA_FOUND THEN
    dbms_output.put_line ('errors with ssn' || emp_ssn);
IF salary_cursor % ISOPEN THEN CLOSE salary_cursor;
END;
```

Q.55. *Explain implicit and explicit cursors.* ***(R.G.P.V., Dec. 2014)***

Or

Differentiate between implicit and explicit cursor. (R.G.P.V., Dec. 2015)

Ans. Refer to Q.54.

Q.56. *What are parameterized cursors ?*

Ans. Usually there are situations where we may require using a cursor a number of times inside our procedure. But if the cursor definition is vast in it's tedious to write the same cursor multiple times for varying conditions and in a where clause. There the parameterized cursor which can be used frequently being declared only once in the code is useful.

Instead of using several cursors definitions of similar nature which take different values in the where clause we can use one parameterized cursor where the value to be used in the where clause is passed through the input parameter to the cursor.

Example –

```
CREATE   OR   REPLACE
PROCEDURE    parameterized_cursor_p as
    /* Parameterized Cursor declaration*/
      Cursor multiple_dept(dept_name    varchar) is
      select a.EMPLOYEE_ID, a.FIRST_NAME, b.DNAME
      from emp a, dept b
      where a. DEPARTMENT_ID = b.DEPTNO
    and    b. DNAME =dept_name;
BEGIN
    /*First use of cursor in selecting the SALES Guys*/
    dbms_output.put_line('First use of cursor in selecting the SALES
                                        Guys');
    For Sales_Emp in multiple_dept('SALES')
    loop
    dbms_output.put_line (Sales_Emp.FIRST_NAME||
        'with Emp_number' ||Sales_Emp.EMPLOYEE_ID||
        'is in' ||Sales_Emp.DEPARTMENT_NAME);
    end loop;
```

```
/* Second use of cursor in selecting the RESEARCH Guys*/
dbms_output.put_line ('Second use of cursor in selecting the
RESEARCH Guys');
For Sales_Emp in multiple_dept ('RESEARCH')
loop
dbms_output.put_line (Sales_Emp.FIRST_NAME ||'with EMP
          number'|| Sales_Emp.EMPLOYEE_ID ||'is in'
          || Sales_Emp. DEPARTMENT_NAME);
end loop;
END  parameterized_cursor_p;
```

Analysis – As per the code we are trying to use the same cursor twice depending upon the parameters passed to it. By using the conventional method of using cursor we would have to write the same cursor twice, first for fetching the SALE guys and next for the RESEARCH guys.

Execution of the Procedure –

```
First use of cursor in selecting the SALES Guys
Den with Emp number 114 is in SALES
Alexander with Emp number 115 is in SALES
Shelli with Emp number 116 is in SALES
Sigal with Emp number 117 is in SALES
Guy with Emp number 118 is in SALES
Karen with Emp number 119 is in SALES
Test1 with Emp number 1 is in SALES
Second use of cursor in selecting the RESEARCH Guys
Michael with Emp number 201 is in RESEARCH
Pat with Emp number 202 is in RESEARCH
```

Q.57. *What are cursors ? Explain nested and parameterized cursors.*

(R.G.P.V., Dec. 2013)

Ans. Cursors – Refer to Q.54.

Nested Cursors – Cursors can be nested inside each other. Although this may sound complex, it is really just a loop inside a loop. Much like nested loops. If you have one parent cursor and two child cursors, each time the parent cursor makes a single loop, it loops through each child cursor once and then begins a second round. The following example shows a nested cursor.

```
SET SERVEROUTPUT ON
-- ch09_9a.sql
DECLARE
   v_amount course.cost%TYPE;
   v_instructor_id instructor.instructor_id%TYPE;
CURSOR c_inst IS
   SELECT first_name, last_name, instructor_id
   FROM instructor;
CURSOR c_cost IS
   SELECT c.cost
     FROM course c, section s, enrollment e
    WHERE s.instructor_id = v_instructor_id
      AND c.course_no = s.course_no
      AND s.section_id = e.section_id;
BEGIN
   FOR r_inst IN c_inst
   LOOP
      v_instructor_id := r_inst.instructor_id;
      v_amount := 0;
      DBMS_OUTPUT.PUT_LINE(
      'Amount generated by instructor'||
      r_inst.first_name||' '||r_inst.last_name
      ||' is');
     FOR r_cost IN c_cost
     LOOP
        v_amount := v_amount + NVL(r_cost.cost, 0);
     END LOOP;
     DBMS_OUTPUT.PUT_LINE
     ('  '||TO_CHAR(v_amount, '$999,999'));
   END LOOP;
END;
```

Parameterized Cursors – Refer to Q.56.

ORACLE EXCEPTION HANDLING MECHANISM, STORED PROCEDURES, IN, OUT, IN OUT TYPE PARAMETERS, USAGE OF PARAMETERS IN PROCEDURES, USER DEFINED FUNCTIONS THEIR LIMITATIONS, TRIGGERS, MUTATING ERRORS, INSTEAD OF TRIGGERS

Q.58. *What is PL/SQL exception handling and why it is needed ? Also explain user-defined exceptions and system-defined exceptions.*

(R.G.P.V., Dec. 2014)

Or

***Explain exception handling mechanism of Oracle with example.* (R.G.P.V., Dec. 2009)**

Or

***Explain ORACLE exception handling mechanism.* (R.G.P.V., June 2011)**

Or

***What is PL/SQL exception handling and why it is needed ?* (R.G.P.V., Dec. 2015)**

Or

***Explain the Oracle exception handling mechanism.* (R.G.P.V., June 2016)**

Ans. Exceptions in Oracle are powerful error handling mechanism. PL/SQL exception handling is a mechanism for dealing with run-time errors encountered during procedure execution. Use of this mechanism enables execution to continue if the error is not severe enough to cause procedure termination. The decision to enable a procedure to continue after an error condition is one you have to make in development as you consider possible errors that could arise. You must define the exception handler within a subprogram specification. Errors cause the program to raise an exception within a transfer of control to the exception-handler block. After the exception handler executes, control returns to the block in which the handler was defined. If there are no more executable statements in the block, control returns to the caller.

Need – The error occurred during the execution of a PL/SQL is called run time error. In general, run time errors are abnormal and crucial which causes your PL/SQL program or task uncompleted. So, in general or by standards, it was a good practice to anticipate them and record a log of them. This is where the role of exceptions comes in your PL/SQL.

User-Defined Exceptions – PL/SQL enables the user to define exception handlers in the declarations area of subprogram specifications. You accomplish this by naming an exception as in the following example –

```
My_Exception EXCEPTION;
```

In this case, the exception name is My_Exception. Code associated with this handler is written in the EXCEPTION specification area as follows –

```
EXCEPTION
when MY_Exception then
    out_status_code := g_out_status_code;
    out_msg         := g_out_msg;
```

This exception is defined to capture status and associated data for any NO_DATA_FOUND exceptions encountered in a subprogram. The following is an example of a subprogram exception –

```
EXCEPTION
when NO_DATA_FOUND then
    g_out_status_code : = 'FAIL';
    RAISE My_Exception;
```

Within this exception is the RAISE statement that transfers control back to My_Exception exception handler. This technique of raising the exception used to invoke all user-defined exceptions.

System-Defined Exceptions

Exceptions internal to PL/SQL are raised automatically upon error. NO_DATA_FOUND from the previous example is a system-defined exception (see table 5.8).

Table 5.8 PL/SQL Internal Exceptions

Exception Name	*Oracle Error*
CURSOR_ALREADY_OPEN	ORA-06511
DUP_VAL_ON_INDEX	ORA-00001
INVALID_CURSOR	ORA-01001
INVALID_NUMBER	ORA-01722
LOGIN_DENIED	ORA-01017
NO_DATA FOUND	ORA-01403
NOT_LOGGED_ON	ORA-01012
PROGRAM_ERROR	ORA-06501
STORAGE_ERROR	ORA-06500
TIMEOUT_ON_RESOURCE	ORA-00051
TOO_MANY_ROWS	ORA-01422
TRANSACTION_BACKED_OUT	ORA-00061
VALUE_ERROR	ORA-06502
ZERO_DIVIDE	ORA-01476

In addition to this list of exceptions, there is catch-all exception named OTHERS that traps all errors for which specific error handling has not been established. This exception is illustrated in the following example –

```
when OTHERS then
    out_status_code : = 'FAIL';
    out_msg : = g_out_msg || ' ' || SUBSTR (SQLERRM, 1, 60);
```

The information passed back to the caller in out_msg is the subprogram name contained in g_out_msg concatenated with the first 60 characters returned from the SQLERRM function by the SUBSTR function.

SQLERRM only returns a valid message when called inside an exception handler unless an argument is passed to the function that is a valid SQL error number. The Oracle error code is the first part of the message returned from SQLERRM. Next is the text associated with that Oracle error code. In this manner, all errors encountered during procedure execution are trapped and passed back to the application for debug purposes.

FAIL : init_line_items ORA-01001 : invalid cursor

This error message (formatted by the application) reveals an illegal cursor operation in the subprogram init_line_items. The portion of the message returned from SQLERRM begins with the ORA-01001 SQL error code. Another error message is illustrated in the following example –

FAIL : calc_ship_charges

In this case, the subprogram calc_ship_charges had a NO_DATA_FOUND error. This is determined by the fact that no SQL error messages are concantenated with the message text.

Q.59. Write short note on the stored procedures.

Ans. A stored procedure is a named collection of procedural and SQL statements. Just like database triggers, stored procedures are stored in the database. The major advantage of stored procedure is that they can be used to encapsulate and represent business transactions. For example, you can create a stored procedure to represent a product sale, a credit update, or the addition of a new customer. By doing that, you can encapsulate SQL statements within a single stored procedure and execute them as a single transaction.

The following syntax is used to create stored procedures –

```
CREATE OR REPLACE PROCEDURE
    Procedure_name [(argument [IN/OUT/IN OUT] data_type, ...)]
                                                          [IS/AS]
    [variable_name  data type [: = initial_value]]
BEGIN
    PL/SQL or SQL statements ;
    ...
    END ;
```

Here, *argument* specifies the parameters that are passed to the stored procedure. A stored procedure could have zero or more arguments or parameters.

IN/OUT indicates whether the parameter is for input or output or both.

Data_type is one of the procedural SQL data types used in the RDBMS. The data types normally match those used in the RDBMS table creation statement.

Variables can be declared between the keywords IS and BEGIN. You must specify the variable name, its data type, and (optionally) an initial value.

Consider, for example, that you want to create a procedure (PRC_PROD_DISCOUNT) to assign an additional 5 percent discount for all products when the quantity on hand is more than or equal to twice the minimum quantity, then the stored procedure is created as –

```
CREATE OR REPLACE PROCEDURE PRC_PROD_DISCOUNT
AS BEGIN
    UPDATE PRODUCT
        SET P_DISCOUNT = P_DISCOUNT*.05
        WHERE P_QOH >= P_MIN*2;
    DBMS_OUTPUT.PUT_LINE ('Update Finished');
END;
```

To execute the stored procedure, you must use the following syntax –

```
EXEC procedure_name[(Parameter_list)];
```

For example, to see the results of running the PRC_PROD_DISCOUNT stored procedure, we can use the EXEC PRC_PROD_DISCOUNT command.

Q.60. What is parameter ? What are the usage of parameters IN, OUT and INOUT in procedures ?

Ans. A parameter is a placeholder in a query or a stored procedure that accepts a user-defined value whenever the query or stored procedure is executed.

The parameters used for creating stored procedures are explained below–

(i) Input (IN) Parameter – It specifies that a value for the argument must be specified when calling the procedure.

(ii) Output (OUT) Parameter – It specifies that the procedure passes a value for this argument back to its calling environment after execution.

(iii) Input Output (INOUT) Parameter – It specifies that a value for the argument must be specified when calling the procedure and that the procedure passes a value for this argument back to its calling environment after execution.

Q.61. Explain user defined functions and their limitations.

(R.G.P.V., June 2016)

Or

What are user defined functions in Oracle ? (R.G.P.V., Dec. 2013)

Ans. Using programmable or procedural SQL, you can also create your own stored functions. Stored procedures and functions are very similar. A stored function is basically a named group of procedural and SQL statements that returns a value (indicated by RETURN statement in its program code). To create a function, the following syntax is used –

```
    CREATE FUNCTION function_name (argument IN data-type, ....)
RETURN data-type [IS]
    BEGIN
    PL/SQL statements;
    .....
    RETURN (value or expression);
    END;
```

For example, a PL/SQL function to get the salary of an employee when the ID is given, is a follows –

```
CREATE FUNCTION Get_Salary (empid IN number)
RETURN number
IS Employee_Salary number (10, 2);
BEGIN
    SELECT empsalary
    INTO Employee_salary
    FROM employee
    WHERE id = empid;
    RETURN (Employee_salary);
END;
```

Limitations – Stored functions can be invoked only from within stored procedures or triggers and cannot be invoked from SQL statements (unless the function follow some very specific compliance rules). Remember not to confuse built-in SQL functions (such as MIN, MAX, and AVG) with stored function.

Q.62. What are the advantages of using functions, triggers and stored procedures? ***(R.G.P.V., June 2010)***

Or

Explain the advantages of stored procedures. ***(R.G.P.V., Dec. 2014)***

Ans. Functions, triggers and stored procedures provide multiple advantages to SQL developers and database administrators to develop and maintain database intensive applications in many ways. Most valuable advantages are as follows –

(i) Securing the organization business logic by implementing those inside the procedures rather than one or two layer above as the application components.

(ii) Improves the productivity by designing the application with a modular approach and re-using the same procedure or function in many other procedures.

(iii) Stored procedures and functions help in improving the integrity and consistency of applications.

(iv) Automatic execution of triggers on various database events.

(v) DBAs can write once and use multiple times some basic procedures which help them manage day-to-day database maintenance jobs.

(vi) Maintains database security by accessing the production data via procedures and functions indirectly.

(vii) Improves performance by reducing network traffic and decreasing the processing time.

Q.63. What are mutating errors?

Ans. Mutating table errors occur when a trigger attempts to access a "mutating" table, like the table from which the trigger has been called. In MySQL, triggers are not initiated by cascading foreign keys, and they cannot modify the table from which they are called, so neither of those issues can cause a mutating table error. However, foreign keys appear to operate before 'AFTER' triggers, so it is possible to create a trigger that will cause a statement to fail.

Mutating errors occurs while using triggers. Table which invoked the trigger is being updated in trigger block. This will give the error.

One more condition is if you are updating a table using a function and if the function is updating same invoking table, this will cause this error. Mutating errors also occur while using functions in a sql query. Also for, example, if you create a function to do any changes on table "abc" and if you use the same function in a sql query where you are using the same "abc" table in the from clause of the query, then mutating errors occur.

There are several causes of mutating tables, but in general a table mutation error will result if you create a trigger on a table that attempts to examine the current table. Also, the error is given when an attempt is made to examine a parent table in a cascading update/delete.

Q.64. What is instead of triggers?

Ans. INSTEAD OF triggers describe how to perform insert, update, and delete operations against views that are too complex to support these operations natively. INSTEAD OF triggers allow applications to use a view as the sole interface for all SQL operations (insert, delete, update and select). Usually, INSTEAD OF triggers contain the inverse of the logic applied in a view body. For example, consider a view that decrypts columns from its source table. The INSTEAD OF trigger for this view encrypts data and then inserts it into the source table, thus performing the symmetrical operation.

Using an INSTEAD OF trigger, the requested modify operation against the view gets replaced by the trigger logic, which performs the operation on behalf of the view. From the perspective of the application this happens transparently, as it perceives that all operations are performed against the view.

Only one INSTEAD OF trigger is allowed for each kind of operation on a given subject view.

The view itself must be an untyped view or an alias that resolves to an untyped view. Also, it cannot be a view that is defined using WITH CHECK OPTION (a symmetric view) or a view on which a symmetric view has been defined directly or indirectly.

Q.65. What is triggers ? Write various types of triggers. **(R.G.P.V., Dec. 2015)**

Or

Explain triggers and its types with examples. **(R.G.P.V., Dec. 2017)**

Ans. **Triggers** – Refer to Q.38 (Unit-II).

Oracle provides several types of triggers –

(i) Row Triggers – Row triggers execute once for every row that is affected by the DML operation.

(ii) Statement Triggers – A statement trigger is executed just once per statement.

(iii) Instead of Triggers – Refer to Q.64.

RGPV

B.E. (Fifth Semester) EXAMINATION, June, 2010
(New Scheme)
(Computer Science Engg. Branch)
DATABASE MANAGEMENT SYSTEM
[CS-503(N)]

Note : Attempt all questions.

1. (a) Define the following terms –

(i) Data models

(ii) Domain constraints. 5 (See Unit-I, Page 10, Q.16)

(b) Draw an E-R diagram for a library management system, make suitable assumptions. (See Unit-I, Page 21, Q.35) 05

(c) Discuss the candidate key, primary key, super key, composite key and alternate key. (See Unit-II, Page 50, Q.12) 10

Or

(a) A university database contains information about professors (identified by social security number) and courses (identified by courseid) professors teach courses, each of the following situations concerns the teaches relationship set. For each situation draw an E-R diagram – 10

(i) Professors can teach the same course in several semesters and each offering must be recorded.

(ii) Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded.

(iii) Every professor must teach some course.

(iv) Every professor teaches exactly one course.

(v) Every professor teaches exactly one course and every course must be taught by some professor.

(b) Explain the component modules of a DBMS and their interactions with the architecture. (See Unit-I, Page 5, Q.7) 10

2. (a) Consider the following relational database –

Employee (EMP_name, Street, City)

Works (EMP_Name, Company_Name, salary)

Company (Company_Name, City)

Manages (EMP_Name, Manager_Name)

For each of the following queries, give an expression in the relational algebra. Tuple relational calculus –

(i) Find the names of all employees who work for Satyam.

(ii) Find the names and cities of residence who work for Satyam.

(iii) Find the names of all employees who live in the same city as the company for which they work.

(iv) Find the names of all employees who do not work for Satyam.

(v) Find the names of all employees who live in the same city and on the same street as do their managers.

(See Unit-II, Page 104, Prob.15)

(b) Explain the basic relational algebra operations with the symbol used and example for each. **(See Unit-II, Page 90, Q.50)** 10

Or

(a) Let the following relational schema be given – 10

Employee (SSN, name, age, dno)

Salary (SSN, salary)

Works_on (Projects #, SSN)

Project (Projects #, project_name, location)

For each of the following queries give an expression in SQL –

(i) Display the name of projects at "Delhi".

(ii) Find the project_name of employee whose salary is greater than 10000.

(iii) Retrieve the name and SSN of employees working on project# A 100. **(See Unit-II, Page 73, Prob.1)**

(b) Write short notes on the following – 10

(i) Data manipulation language (DML) **(See Unit-I, Page 17, Q.28)**

(ii) Data definition language (DDL) **(See Unit-I, Page 16, Q.27)**

(iii) Transaction control statement (TCS) **(See Unit-IV, Page 153, Q.7)**

(iv) Data control language (DCL) **(See Unit-I, Page 17, Q.29)**

(v) Data administration statements (DAS)

3. (a) What is normalization ? Discuss various normal forms with the help of examples. **(See Unit-III, Page 139, Q.38)** 10

(b) Consider the relation r(A, B, C, D, E) and the set F = {AB → CE, E → AB, C → D}. What is the highest normal form of this relation ? **(See Unit-III, Page 127, Prob.4)** 10

Or

(a) Define the following terms – 10

(i) Functional dependency **(See Unit-III, Page 121, Q.17)**

(ii) Lossless decomposition. **(See Unit-III, Page 130, Q.27)**

(iii) Dependency preservation **(See Unit-III, Page 129, Q.26)**

(iv) Third normal form. **(See Unit-III, Page 114, Q.6)**

(b) Given the relation r(X, Y, W, Z, P, Q) and the set F = {XY → W, XW → P, PQ → Z, XY → Q}, consider the decomposition R_1(Z, P, Q), R_2(X, Y, Z, P, Q). Is this decomposition lossless or lossy ? Use the lossless-join algorithm. **(See Unit-III, Page 139, Prob.5)** 10

4. (a) Explain 2-phase commitment protocol and the behaviour of this protocol during lost messages and site failures. **(See Unit-IV, Page 185, Q.51)** 10

(b) What do you mean by time stamping protocol for concurrency control ? Discuss multi-version scheme of concurrency control also. **(See Unit-IV, Page 184, Q.49)** 10

Or

(a) Describe strict two-phase locking protocol. 10

(b) Define serializability and differentiate conflict and view serializability. **(See Unit-IV, Page 176, Q.40)**

5. (a) What are the advantages of using functions, triggers and stored procedures ? **(See Unit-IV, Page 154, Q.9)** 10

(b) Describe the different types of file organization. Explain using a sketch of each of them with their advantages and disadvantages. **(See Unit-V, Page 254, Q.62)** 10

(See Unit-II, Page 65, Q.36)

Or

(a) State with examples desirable properties of a transaction. What is the system log used for ? **(See Unit-IV, Page 163, Q.22)** 10

(b) How does a DBMS represent a relational query evaluation plan ? 10 **(See Unit-III, Page 147, Q.42)**

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec., 2010
(New Scheme)
(Computer Science Engg. Branch)
DATABASE MANAGEMENT SYSTEM
[CS–503(N)]

Note : The question paper is divided into five Units. Each unit carries an internal choice. Attempt one question from each Unit. Thus attempt five questions in all. All questions carry equal marks. Assume suitable data whenever necessary.

Unit-I

(i) What is DBMS and what are the components of DBMS ? What are the advantages of DBMS over file oriented approach ? 7

(See Unit-I, Page 10, Q.15)

(ii) Explain the steps for reduction of E-R model into relational model. 7 **(See Unit-I, Page 38, Q.59)**

(iii) What is the difference between a database schema and a database state ? **(See Unit-I, Page 16, Q.24)** 6

Or

2. (i) A database is to be constructed to keep track of the teams and games of a sport league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game of each team and the result of the game. Create an ER diagram completely with attributes, keys and constraints, for the above description. State any assumptions that you make. 10 **(See Unit-I, Page 31, Q.46)**

(ii) Design a generalization-specification hierarchy for a motor-vehicle sales company. The company sells motorcycles which have an engine number and cost; cars which have a chassis number, and engine number, seating capacity and cost; trucks which have chassis number, an engine number and cost. **(See Unit-I, Page 37, Q.56)** 10

Unit-II

3. (i) Consider the relation – 12

PROJECT (proj#, pro_name)
EMPLOYEE (emp#, emp_name)
ASSIGNED (proj#, emp#)

Use relational algebra to express the following queries –

(a) Find the employee number of employees who work on at least all of the project that employee 107 works on.

(b) Get the employee number of employees who work on all projects. **(See Unit-II, Page 105, Prob.16)**

Use relational algebra to express the following queries – Compile a list of employee number of employees who work on all projects.

(ii) Describe entity integrity and referential integrity. Give an example of each. **(See Unit-II, Page 52, Q.17)** 8

Or 10

4. (i) Consider the relations –

EMP (ENO, ENAME, AGE, BASIC)
WORK_ON (ENO, DNO)
DEPT (DNO, DNAME, CITY)

Express the following queries in SQL –

(a) Find names of employees whose basic pay is greater than average basic pay.

(b) Find the sum of the basic pay of all the employees, the maximum

basic pay, the minimum basic pay and the average basic pay. **(See Unit-II, Page 74, Prob.2)**

(ii) Let R(A, B) and S(A, C) be two relations. Give relational algebra expressions for the following domain calculus expressions – 01

(a) $\{<a> \mid \exists b(< a, b > \in r \wedge b = 17)\}$

(b) $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$

(c) $\{< a > \mid \exists b(\exists c\{< a, b > \exists r) \wedge < a, c > \exists s)\}\}$

(See Unit-II, Page 105, Prob.17)

Unit-III

5. (i) Consider the relation Student (stid, name, course, year). Given that a student may take more than one course but has unique name and the year of joining. 12

(a) Identify the functional and multivalued dependencies for student.

(b) Identify a candidate key using the functional and multivalued dependencies arrived at in step (a).

(c) Normalize the relation so that every decomposed relation is in 4NF. **(See Unit-III, Page 142, Prob.7)**

(ii) What is NULL ? Give an example to illustrate testing for NULL in SQL. **(See Unit-II, Page 64, Q.35)** 8

Or

6. (i) What is irreducible set of dependencies ? Relation R with attributes A, B, C, D and FDs. 10

$A \rightarrow BC$
$B \rightarrow C$
$A \rightarrow B$
$AB \rightarrow C$
$AC \rightarrow D$

Compute an irreducible set of FDs that is equivalent to this given set. **(See Unit-III, Page 125, Q.24)**

(ii) Explain Boyce-Codd Normal form with example and also compare BCNF and 3NF. **(See Unit-III, Page 116, Q.9)** 10

7. (i) Discuss the timestamp ordering protocol for concurrency control. How does strict timestamp ordering differ from basic timestamp ordering ? **(See Unit-IV, Page 178, Q.43)** 10

(ii) Consider the following two transactions – 10

T1 :	read (A);	T2 :	write (A)
	read (B);		read (B)
	B = A + B;		
	write (B)		

Add lock and unlock instructions so that the transaction T1 and T2 observe two-phase locking protocol. Is it deadlock free ? (See Unit-IV, Page 178, Q.41)

Or

8. (i) Explain the concept of two phase locking and show that it guarantees serializability. (See Unit-IV, Page 175, Q.38) 7

(ii) Explain following –

(a) Recoverable schedule (See Unit-IV, Page 159, Q.15)

(b) Cascadeless schedule (See Unit-IV, Page 159, Q.16)

(iii) Explain the recovery process after system failure using checkpoint. (See Unit-IV, Page 165, Q.26) 6

Unit-V

9. (i) With respect to Oracle describe the following – 10

(a) Data block (b) Data dictionary (c) Segments. (See Unit-V, Page 221, Q.24)

(ii) What is a view ? Create a view of EMP table named DEPT 20, to show the employees in department 20, and their annual salary. 10 (See Unit-II, Page 99, Q.63)

Or

10. (i) Explain the following features of ORACLE – 10

(a) Sequence (See Unit-V, Page 230, Q.37)

(b) Triggers (See Unit-II, Page 70, Q.38)

(c) SQL loader. (See Unit-V, Page 231, Q.38)

(ii) Define the following terms – 10

(a) Distributed system (b) Catalog. (See Unit-V, Page 214, Q.14)

RGPV

B.E. (Fifth Semester) EXAMINATION, June, 2011
(Computer Science & Engg. Branch)
DATABASE MANAGEMENT SYSTEM
[CS–503(N)]

Note : Attempt *one* question from each Unit. All questions carry equal marks.

Unit-I

1. Discuss overall system structure of a database management system. Explain the difference between logical and physical schema. What is view level in this context ? (See Unit-I, Page 16, Q.22)

Or

2. Construct an ER diagram for a hospital with a set of patients and a set of doctors. Associate with each patient a log of the various tests and examination conducted. Also show the tables for various entities with attributes. (See Unit-I, Page 31, Q.47)

Unit-II

3. Explain the following keys with suitable example –

(i) Primary key (ii) Secondary key (iii) Foreign key (iv) Super key. Explain referential integrity and integrity constraints.

(See Unit-II, Page 54, Q.20)

Or

4. What is SQL ? Which data models implements this language ? Write a SQL server program segment to write rules and defaults. How can you connect and disconnect them to a database ? (See Unit-II, Page 57, Q.27)

Unit-III

5. What do you mean by normalization ? Illustrate BCNF with suitable example. How BCNF is better than 3NF ? Justify your answer. Also explain functional dependency. (See Unit-III, Page 122, Q.18)

Or

6. Suppose that we decompose the scheme R = (A, B, C, D, E) into (A, B, C) and (A, D, E). Show that this decomposition is a lossless join decomposition if the following functional dependencies hold :

$$A \rightarrow BC, CD \rightarrow E, B \rightarrow D$$

Show that this decomposition is dependency preserving decomposition.

(See Unit-III, Page 140, Prob.6)

Unit-IV

7. What is a transaction in database ? Discuss the property of transaction along with various states of transaction. Briefly discuss purpose of log file in database recovery. (See Unit-IV, Page 164, Q.23)

Or

8. What is concurrent schedule ? What is conflict serializability in this context ? What is time-stamp based protocol in context of concurrent schedules ?

(See Unit-IV, Page 180, Q.46)

Unit-V

9. Write short notes on the following :

(i) Distributed database (See Unit-V, Page 215, Q.15)

(ii) Triggers. (See Unit-II, Page 70, Q.38)

Or

10. Explain the following (any *two*) –

(i) RDBMS (See Unit-V, Page 201, Q.2)

(ii) ANSI SQL

(iii) ORACLE exception handling mechanism. (See Unit-V, Page 249, Q.58)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec., 2011
(Computer Science Engg. Branch)
DATABASE MANAGEMENT SYSTEM
(CS–503)

Note : Attempt all questions. All questions carry equal marks.

1. (a) Define the following terms : 5

   (i) Data redundancy and inconsistency (See Unit-I, Page 3, Q.3)

   (ii) Referential Integrity (See Unit-II, Page 54, Q.18)

   (b) Draw an E-R diagram for a hospital with a set of patients and a set of medical doctors. With each patient a log of the various conducted tests is also associated. (See Unit-I, Page 31, Q.47) 5

   (c) When is the concept of weak entity used in data modelling ? Define the terms owner entity, weak entity, identifying relationship, partial key. (See Unit-I, Page 23, Q.38) 10

Or

2. (a) A university database contains information about professors (identified by social security number) and courses (identified by courseid) professors teach courses; each of the following situations concerns the teaches relationship set.

   For each situation draw an E-R diagram – 10

   (i) Professors can teach the same course in several semesters and each offering must be recorded.

   (ii) Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded.

   (iii) Every Professor must teach some course.

   (iv) Every Professor teaches exactly one course.

   (v) Every Professor teaches exactly one course and every course must be taught by some professor. 10

   (b) Define the following terms –

   (i) Data Models (See Unit-I, Page 10, Q.16)

   (ii) Domain Constraints. (See Unit-I, Page 21, Q.35) 10

3. (a) Given the relation schema –

   ENROLL (S#, C#, Section), S# is student number.

   TEACH (Prof, C#, Section), C# is course number

   ADVISE (Prof, S#), Prof. is Thesis adviser of S#.

   PRE-REQ (C#, Pre-C#) pre-C# is prerequisite course.

   GRADE (S#, C#, grade, year)

   STUDENT (S#, Sname), Sname is student name.

   Give queries expressed in SQL and tuple calculus –

   (i) List of students taking course with Smith or Jones.

   (ii) List of students taking at least one course that their advisor teaches.

   (iii) List those professors who teach more than one section of the same course.

   (iv) List all students number and course number.

   (v) List the student number and course number who get grade A.

   (b) What are the various characteristics of SQL ? Discuss *five* aggregate functions with a suitable example. (See Unit-II, Page 59, Q.28) 5

   (c) Discuss the selection, projection and join operator of relational algebra with a suitable example. (See Unit-II, Page 93, Q.55) 5

Or

4. (a) Specify the following query in relational algebra – 10

   Supplier (sid, sname, address)

   Part (sid, pname, color)

   Catalog (sid, pid, cost)

   (i) Find the name of suppliers who supply some red or green part.

   (ii) Find the sids of suppliers who supply every part.

   (iii) Find the sids of suppliers who supply red and green part. (See Unit-II, Page 104, Prob.14)

   (b) List the operations of relational algebra and the purpose of each. 10 (See Unit-II, Page 96, Q.59)

5. (a) What do you mean by Normalization ? Explain BCNF and 3NF with suitable example. (See Unit-III, Page 118, Q.12) 10

   (b) What do you mean by decomposition of a relation ?

   Consider the relational scheme –

   R (A, B, C, D, E, F) and FD.

   $A \rightarrow BC, C \rightarrow A, D \rightarrow E, F \rightarrow A, E \rightarrow D$

   Is the decomposition of R into R_1(A, C, D), R_2 (B, C, D) and R_3 (E, F, D) lossless ?

   Explain the requirements for lossless decomposition and dependency preserving. (See Unit-III, Page 133, Q.30) 10

Or

6. (a) Define the following terms – 10

   (i) Functional dependency (See Unit-III, Page 121, Q.17)

   (ii) Lossless decomposition (See Unit-III, Page 130, Q.27)

   (iii) Dependency preservation (See Unit-III, Page 129, Q.26)

   (iv) Third Normal form. (See Unit-III, Page 114, Q.6)

   (b) Consider the following relation –

Book (book_title, authorname, book_type, listprice, author_affiliation, publisher)

suppose the following functional dependencies exist –

book_title → publisher, book_type

book_type → listprice

authorname → author_affiliation

(i) What normal form is the relation in ? Explain your answer.

(ii) Apply normalization until you cannot decompose the relations further. State the reasons behind each decomposition. 10

7. (a) What do you mean by schedule in the context of concurrent execution of transactions in RDBMS ? What is serializable schedule ?

(See Unit-IV, Page 160, Q.17)

Discuss the various types of serializability with a suitable example.

(See Unit-IV, Page 159, Q.14) 10

(b) Compare and contrast the features of log based recovery mechanism versus check pointing based recovery. Suggest applications where you prefer log based recovery scheme over check pointing. Give an example of check pointing based recovery scheme. Discuss the recoverable schedule also. **(See Unit-IV, Page 166, Q.27)** 10

Or

8. (a) Explain the following terms – 10

(i) Deadlock detection and recovery **(See Unit-IV, Page 168, Q.30)**

(ii) Multiversion techniques. **(See Unit-IV, Page 183, Q.48)**

(b) What do you mean by locking techniques of concurrency control ? Discuss the various locking techniques and recovery with concurrent transaction also in detail. **(See Unit-IV, Page 187, Q.56)** 10

9. (a) What are triggers ? Explain with suitable example. How can they help in building Robust Database ? **(See Unit-II, Page 72, Q.39)** 10

(b) Describe the different types of file organization. Explain using a sketch of each of them with their advantages and disadvantages. 10

(See Unit-II, Page 65, Q.36)

Or

10. (a) Explain the index schemas used in DBMS. 10

(b) How does a DBMS represent a relational query evaluation plan ? **(See Unit-III, Page 147, Q.42)** 10

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2012

DATABASE MANAGEMENT SYSTEM

(CS-503)

Note : (i) Attempt *one* question from each Unit.

(ii) All questions carry equal marks.

Unit-I

1. What are the main difference between a file processing system and a database management system ? Describe the overall system architecture of a database system. Also show the connection system. Also show the connection amongst its various components. **(See Unit-I, Page 9, Q.12)**

2. Construct an ER diagram of customer account relationship. Customer entity with attributes SS#, customer name, street, customer City and account entity with attributes account No. and balance. The Customer Account relationship with date attributes. **(See Unit-I, Page 33, Q.49)**

Unit-II

3. What do you mean by query language ? Explain following operations by taking suitable examples.

(i) Select (ii) Project (iii) Cartesian Product (iv) Rename (v) Union.

Also explain superkey and candidate key. **(See Unit-II, Page 91, Q.51)**

4. What are database languages ? Why were the DDL and DML called data sublanguage ? Also, explain functional dependency.

(See Unit-II, Page 63, Q.32)

Unit-III

5. What is the utility of normalization & various normal forms ? Explain the concept of transaction atomicity and serializability.

(See Unit-IV, Page 158, Q.13)

Or

6. Explain why 1NF is acceptable for data processing applications. How 2NF is better than 1NF. Is BCNF better than 3NF ? Give an example of a relation that is in 3NF but not in BCNF. **(See Unit-III, Page 120, Q.15)**

Unit-IV

7. What is transaction in database ? Discuss properties of transaction. Explain transaction logging in databases and its use in two phase commit policy.

(See Unit-IV, Page 186, Q.52)

Or

8. What are database locks and how do they lead to deadlock ? Briefly discuss strategies for preventing deadlocks. How do you detect deadlocks ?

(See Unit-IV, Page 175, Q.37)

Unit-V

9. Write short notes on any two –
(i) RDBMS (ii) Distributed database (iii) Self join & outer join.
(See Unit-V, Page 233, Q.42)

Or

10. At what point during query processing does the optimization occur. Discuss the advantages of distributing database. Also, explain database security.
(See Unit-V, Page 223, Q.29)

B.E. (Fifth Semester) EXAMINATION, Dec. 2013
DATABASE MANAGEMENT SYSTEM
(CS-503)

Note : (i) Attempt any *one* question from each Unit.
(ii) All questions carry equal marks.

Unit-I

1. (a) Explain system structure of DBMS. (See Unit-I, Page 5, Q.7) 4
(b) Explain the following terms – 6
(i) Database schema (See Unit-I, Page 14, Q.19)
(ii) Data independence. (See Unit-I, Page 16, Q.25)
(c) Differentiate between two tier and three tier client/server architecture.
(See Unit-I, Page 5, Q.6) 4

Or

2. (a) Explain the following – 6
(i) Mapping cardinalities
(ii) Participation constraints
(iii) Attribute inheritance.
(See Unit-I, Page 37, Q.57)
(b) Explain the tabular representation of the following – 8
(i) Strong entity set (ii) Weak entity set
(iii) Relationship sets (iv) Generalization.
(See Unit-I, Page 38, Q.59)

Unit-II

3. (a) Explain the following with examples –
(i) Super key (ii) Primary key
(iii) Alternate key (iv) Extensions and Intensions.
(See Unit-II, Page 56, Q.24)
(b) What is union compatibility ? Why do the union intersection and set difference operations require that the relations on which they are applied are union compatible ? (See Unit-II, Page 89, Q.48) 7

Or

4. (a) Explain natural join, outer join, full outer join, left outer join and theta join with examples.
(See Unit-II, Page 98, Q.61) 7
(b) Consider the following database with primary key under lined. 7
(i) Employee-(ENO, DOB, Name, Address, Sex, Salary, Dept-No)
(ii) Department-(Dept-No, Dept-Name):
For each of the following queries give expression in SQL
(i) Retrieve the names of employees in department-5
(ii) Retrieve names of all employees who are not in department-5
(iii) Retrieve the average salary of all female employees
(iv) Write SQL DDL statements of shove database.
(See Unit-II, Page 79, Prob.10)

Unit-III

5. (a) Consider the relation R (A, B, C, D, E, F, G, H, I, J) and set of dependencies. 7

$$F = \{\{A, B\} \to \{C\}, \{A\} \to \{D, E\}, \{B\} \to \{F\}, \{F\} \to \{G, H\}, \{D\} \to \{I, J\}\}$$

What are the keys of R, decompose R in 2NF and 3NF ?
(See Unit-III, Page 126, Prob.2)
(b) Differentiate between 3NF and BCNF with examples. 7
(See Unit-III, Page 118, Q.11)

Or

6. (a) Consider the relational schema R(A, B, C) with FD's AB → C, and C → A. Show that the schema R is in 3NF but not in BCNF. Also determine minimal keys of R. 7
(b) Explain various steps of query optimization. Also discuss optimization methods. (See Unit-III, Page 143, Q.40) 7

Unit-IV

7. (a) Explain various transaction states with their description. Also discuss its state diagram. (See Unit-IV, Page 152, Q.5)
(b) State the write-ahead log rule. Why is the rule necessary ? 4
(See Unit-IV, Page 164, Q.24)
(c) Explain checkpoint record. (See Unit-IV, Page 165, Q.25) 3

Or

8. (a) State two phase locking theorem. Explain how two phase locking deals with RW, WR and WW conflicts. 7
(b) Transaction usually cannot be nested inside one another. Why not ? 3
(See Unit-IV, Page 154, Q.8)
(c) What are the recovery implication of physical writing database buffers at COMMIT ? (See Unit-IV, Page 162, Q.20)

Unit-V

9. (a) Explain hierarchical queries, inline queries and flashback queries. 5 (See Unit-V, Page 237, Q.48)

(b) What are user defined functions in Oracle ? 4 (See Unit-V, Page 253, Q.61)

(c) Explain data dictionary. (See Unit-I, Page 4, Q.5) 4

Or

10. (a) What are cursors ? Explain nested and parameterized cursors. (See Unit-V, Page 248, Q.57)

(b) Explain –
(i) Hierarchical queries
(ii) Inline queries
(iii) Flashback queries. (See Unit-V, Page 237, Q.48)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2014
DATABASE MANAGEMENT SYSTEM
(CS-503)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.
(ii) All parts of each question are to be attempted at one place.
(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.
(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) How is the data-based approach different from the file based approach ? (See Unit-I, Page 8, Q.10)

(b) Differentiate the term data, information and knowledge. (See Unit-I, Page 3, Q.2)

(c) What is schema and subschema ? Explain these two concepts through examples. (See Unit-I, Page 14, Q.19)

(d) What are the problem caused by data redundancies ? Can data redundancy be completely eliminated when a database approach is used. (See Unit-I, Page 4, Q.4)

Or

Define the concept of aggregation. Give several examples of where this concept is useful. (See Unit-I, Page 33, Q.51)

Unit-II

2. (a) What is aggregate functions of SQL ? (See Unit-II, Page 61, Q.29)

(b) What is the difference between procedural DML and non procedural DML ? (See Unit-II, Page 64, Q.34)

(c) Explain the characteristics of relation. Also explain the relational databases. (See Unit-II, Page 52, Q.16)

(d) What are integrity constraints ? Define the term primary key constraint and foreign key constraint. How are these constraints expressed in SQL ? (See Unit-II, Page 54, Q.19)

Or

Explain the Join operator, its relevance and its various types. (See Unit-II, Page 98, Q.61)

Unit-III

3. (a) Discuss how dangling tuple may arise ? (See Unit-III, Page 137, Q.33)

(b) What is query optimization ? (See Unit-III, Page 143, Q.40)

(c) What is join dependency ? How is it different to that of multivalued dependency and functional dependency ? (See Unit-III, Page 139, Q.37)

(d) "Redundancy is the back bone of reliability, therefore, a reliable database system should not attempt normalization beyond 3 NF". Comment on the above statement. Give reason in support of or against the above statement.

Or

What is Normalization ? Explain second normal form with the help of an example. (See Unit-III, Page 114, Q.5)

Unit-IV

4. (a) What is web databases ? (See Unit-IV, Page 199, Q.67)

(b) Define the concept of data warehousing. (See Unit-IV, Page 193, Q.59)

(c) What is schedule ? What is an interleaved schedule ? How is schedule related to the term serializability ? (See Unit-IV, Page 161, Q.18)

(d) Explain the advantages and disadvantages of distributed database. (See Unit-V, Page 218, Q.19)

Or

What is time stamping ? Explain a mechanism of concurrency control that uses time stamping with the help of an examples. (See Unit-IV, Page 180, Q.45)

Unit-V

(a) Define segment, extents and block. (See Unit-V, Page 213, Q.12)

(b) What is flashback queries ? Where it is used ? (See Unit-V, Page 237, Q.47)

(c) Explain the advantages of stored procedures. (See Unit-V, Page 254, Q.62)

(d) What is PL/SQL exception handling and why it is needed ? Also explain user defined exception and system. Defined exceptions. (See Unit-V, Page 249, Q.58)

Or

Explain the following –

(i) Implicit and explicit cursors (See Unit-V, Page 247, Q.55)

(ii) Dedicated and multi-threaded server. (See Unit-V, Page 213, Q.13)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2015
DATABASE MANAGEMENT SYSTEM
(CS-503)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) Why is the hierarchical data model considered inflexible ? (See Unit-I, Page 44, Q.63)

(b) Differentiate the term data, information and knowledge. (See Unit-I, Page 3, Q.2)

(c) What is weak entity set and strong entity set ? (See Unit-I, Page 23, Q.38)

(d) Discuss the three level architecture of DBMS. Explain how does it lead to data independence. (See Unit-I, Page 15, Q.21)

Or

Construct an E-R Diagram for hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted. (See Unit-I, Page 31, Q.47)

Unit-II

2. (a) Differentiate between relational calculus and relational algebra. (See Unit-II, Page 100, Q.66)

(b) State two integrity rules. (See Unit-II, Page 52, Q.17)

(c) What are the various types of inner join operators ? Why theta join is required ? (See Unit-II, Page 94, Q.56)

(d) Explain the characteristics of relation. Also explain the relational databases. (See Unit-II, Page 52, Q.16)

Or

What are DDL, DML and DCL ? Differentiate among the three and give one command for each of these. (See Unit-I, Page 18, Q.30)

Unit-III

3. (a) What do you mean by the terms lossless decomposition ?

(b) What is multivalued dependencies ? (See Unit-III, Page 130, Q.27)

(c) Write a brief notes on trival and non-trival dependencies. (See Unit-III, Page 137, Q.34)

(See Unit-III, Page 123, Q.19)

(d) Prove that a relation which is in 4NF must be in BCNF. (See Unit-III, Page 120, Q.16)

Or

What is normalization ? Justify the need for normalization with examples. (See Unit-III, Page 109, Q.1)

Unit-IV

4. (a) What is web databases ? (See Unit-IV, Page 199, Q.67)

(b) Compare OODBMS and DBMS. (See Unit-IV, Page 196, Q.62)

(c) What are challenges in designing object-oriented databases ? Discuss.

(d) What problems occur in the database when transactions do not satisfy ACID properties ? Explain explicitly using suitable examples.

Or

Explain the significance of multimedia and mobile database. Also give two examples for each.

Unit-V

5. (a) What is inline queries ? Where it is used ? (See Unit-V, Page 236, Q.46)

(b) How the "GROUP BY" clause works ? (See Unit-V, Page 229, Q.34)

(c) Differentiate between implicit and explicit cursor. (See Unit-V, Page 247, Q.55)

(d) What is triggers ? Write various types of triggers. (See Unit-V, Page 256, Q.65)

Or

What is PL/SQL exception handling and why it is needed ? (See Unit-V, Page 249, Q.58)

RGPV

B.E. (Fifth Semester) EXAMINATION, June 2016
DATABASE MANAGEMENT SYSTEM
(CS-503)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

Unit-I

1. (a) What is schemas ? (See Unit-I, Page 14, Q.19)

(b) What is aggregation and specialization ? (See Unit-I, Page 36, Q.54)

(c) What is entity and attribute ? Explain the entity types. (See Unit-I, Page 26, Q.41)

(d) Explain the various data models briefly with an example. (See Unit-I, Page 11, Q.17)

Or

Draw an E-R diagram for a small marketing company database. Assume suitable data. (See Unit-I, Page 33, Q.48)

Unit-II

2. (a) Write the commands of DDL. (See Unit-II, Page 61, Q.30)

(b) What is SQL triggers ? (See Unit-II, Page 70, Q.38)

(c) Explain integrity constraints. (See Unit-II, Page 52, Q.17)

(d) What are different types of relational query languages ? Discuss the different techniques for optimising the queries. (See Unit-III, Page 150, Q.45)

Or

Explain select, project, join and division with example. (See Unit-II, Page 96, Q.58)

Unit-III

3. (a) State the purpose of query optimization. (See Unit-III, Page 142, Q.39)

(b) Define the third normal form. (See Unit-III, Page 114, Q.6)

(c) What do you mean by weak entity set ? (See Unit-I, Page 25, Q.39)

(d) Explain non loss decomposition and functional dependencies with example. (See Unit-III, Page 131, Q.28)

Or

Consider the universal relation R{A, B, C, D, E, F, G, H, I, J} and the set of functional dependencies F = {A, B} → {C}, {A} → {D, E}, {B} → {F}, {F} → {G, H}, {D} → {I, J}. What is the key for R ? Decompose R into 2NF, then 3NF relations. (See Unit-III, Page 126, Prob.2)

Unit-IV

4. (a) What is serializability ? (See Unit-IV, Page 158, Q.10)

(b) What is the difference between view and table ? (See Unit-II, Page 99, Q.64)

(c) What are the problems of lock based protocols ? (See Unit-IV, Page 172, Q.35)

(d) State and explain the three concurrency problems. (See Unit-IV, Page 169, Q.31)

Or

Explain immediate update and deferred update of recovery techniques. (See Unit-IV, Page 190, Q.57)

Unit-V

5. (a) What is cursor ? (See Unit-V, Page 245, Q.54)

(b) What is inline queries ? (See Unit-V, Page 236, Q.46)

(c) Explain user defined functions and their limitations. (See Unit-V, Page 253, Q.61)

(d) Explain the Oracle exception handling mechanism. (See Unit-V, Page 249, Q.58)

Or

Write the branching and looping constructs with example. (See Unit-V, Page 242, Q.53)

RGPV

B.E. (Fifth Semester) EXAMINATION, Dec. 2016
DATABASE MANAGEMENT SYSTEM
(CS-503)

Note : (i) Answer five questions. In each question part A, B, C is compulsory and D part has internal choice.

(ii) All parts of each question are to be attempted at one place.

(iii) All questions carry equal marks, out of which part A and B (Max. 50 words) carry 2 marks, part C (Max. 100 words) carry 3 marks, part D (Max. 400 words) carry 7 marks.

(iv) Except numericals, Derivation, Design and Drawing etc.

1. (a) Define entity and entity set. (See Unit-I, Page 26, Q.41

(b) What is data independency ? (See Unit-I, Page 16, Q.2

(c) What do you mean by DBMS architecture ? (See Unit-I, Page 5, Q.7

(d) What is DBA ? What are the various roles of DBA ? (See Unit-I, Page 19, Q.3

Or

Draw E-R diagram for Hospital management system and convert into set of table schema. (See Unit-I, Page 31, Q.4

2. (a) What do you mean by attributes ? (See Unit-II, Page 46, Q.

(b) Define referential integrity constraints. (See Unit-II, Page 54, Q.18)

(c) What do you mean by integrity constraints ? (See Unit-II, Page 52, Q.

(d) Describe the following –

(i) Relational algebra (See Unit-II, Page 84, Q.

(ii) Relational calculus. (See Unit-II, Page 99, Q.65)

Or

Consider the employee data. Give an expression in SQL for the following query –

Employee (employee-name, street, city)
Works (employee-name, company-name, salary)
Company (company-name, city)
Manages (employee-name, manager-name)

(i) Find the name of all employees who work for State Bank.
(ii) Find the names and cities of residence of all employees work for State Bank.
(iii) Find all employee in the database who do not work for State Bank.
(iv) Find all employee in the database who earn more than every employee of UCO Bank.

(See Unit-II, Page 77, Prob.7)

3. (a) What do you mean by normalization ? (See Unit-III, Page 109, Q.1)
(b) What is data inconsistency ? (See Unit-I, Page 3, Q.3)
(c) What is null value problem ? (See Unit-III, Page 134, Q.31)
(d) How query optimization is performed ? (See Unit-III, Page 143, Q.40)

Or

Consider a reaction R with five attributes A, B, C, D, E having following dependencies : $A \rightarrow B$, $BC \rightarrow E$ and $ED \rightarrow A$

(i) List all keys for R
(ii) In which normal form table is, justify your answer.

(See Unit-III, Page 127, Prob.3)

4. (a) Explain durability in transaction. (See Unit-IV, Page 152, Q.3)
(b) What do you mean by consistency in transaction ?
(See Unit-IV, Page 152, Q.4)
(c) Define views. (See Unit-II, Page 98, Q.62)
(d) Explain ACID properties. Explain serializability of schedule.
(See Unit-IV, Page 158, Q.12)

Or

Explain concurrency control techniques. Also discuss about locking technique. (See Unit-IV, Page 187, Q.54)

5. (a) Explain basic component of RDBMS. (See Unit-V, Page 202, Q.3)
(b) What is the difference between DBMS and RDBMS ?
(See Unit-V, Page 203, Q.4)
(c) What is cursors ? (See Unit-V, Page 245, Q.54)
(d) Explain the advantages of PL/SQL. What do you mean by security ? What the objective while designing secure database ?
(See Unit-V, Page 241, Q.52)

Or

Discuss join operation and its types with example.
(See Unit-II, Page 92, Q.54)

RGPV

CS-5003 (CBGS)

B.E. (Fifth Semester) EXAMINATION, Dec. 2017

Choice Based Grading System (CBGS)

DATABASE MANAGEMENT SYSTEM

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Briefly explain about Database system architecture.
(See Unit-I, Page 5, Q.7) 7
(b) Explain E-R Model in detail with suitable example.
(See Unit-I, Page 30, Q.45) 7

2. (a) Consider the following tables – 7

Employee (Emp_no, Name, Emp_city)
Company (Emp_no, Company_name, Salary)

(i) Write a SQL query to display employee name and company name.
(ii) Write a SQL query to display employee name, employee city, company name and salary of all the employees whose salary > 10000.
(iii) Write a query to display all the employees working in "XYZ" company. (See Unit-II, Page 106, Prob.18)

(b) Consider the following relational schema – 7

Employee (empno, name, office, age)
Books (isbn, title, authors, publisher)
Loan (empno, isbn, data)

Write the following queries in relational algebra –

(i) Find the names of employees who have borrowed a book published by McGraw-Hill.
(ii) Find the names of employees who have borrowed all books published by McGraw-Hill.
(iii) Find the names of employees who have borrowed more than five different books published by McGraw-Hill.

(See Unit-II, Page 106, Prob.19)

3. (a) What is 1NF, 2NF, 3NF and BCNF (Boyce-Codd Normal Form) ?
(See Unit-III, Page 118, Q.10) 7
(b) Explain functional dependency and Trivial functional dependency with examples. (See Unit-III, Page 123, Q.20) 7

4. (a) What is two-phase locking and how does it guarantee serializability ?
(See Unit-IV, Page 175, Q.38) 7
(b) Discuss the concurrency control mechanism in detail using suitable example. (See Unit-IV, Page 187, Q.54) 7

5. (a) Explain Triggers and its types with examples.
(See Unit-V, Page 256, Q.65) 7
(b) Discuss the various type of join operations. Why are these join required ? (See Unit-II, Page 98, Q.6

6. (a) Explain the purpose of checkpoint mechanism. How often should checkpoints be performed ? (See Unit-IV, Page 166, Q.28) 7

(b) What do you understand by distributed databases ? Give the various advantages and disadvantages of distributed database management system. (See Unit-V, Page 219, Q.20) 7

7. (a) What is structured query language ? How the DDL and DML are different ? Explain. (See Unit-II, Page 64, Q.33) 7

(b) Describe the three-level architecture of DBMS. Also explain its importance in a database environment. (See Unit-I, Page 14, Q.20) 7

8. (a) What do you mean by mapping cardinalities ? Explain various type of cardinalities. (See Unit-I, Page 38, Q.58)

(b) Define the following in context of SQL – 7
(i) Distinct clause (See Unit-V, Page 229, Q.35)
(ii) Group by clause (See Unit-V, Page 229, Q.34)
(iii) Union (See Unit-II, Page 87, Q.45)
(iv) Natural join (See Unit-II, Page 91, Q.53)
(v) Order by clause. (See Unit-V, Page 230, Q.36)

RGPV

CS-5003 (CBGS)
B.E. V Semester
EXAMINATION, May 2018
Choice Based Grading System (CBGS)
DATA BASE MANAGEMENT SYSTEM

Note : (i) Attempt any five questions. All questions carry equal marks.
(ii) Assume suitable value for missing data, if any.

1. (a) Lisr four significant differences between a file-processing system and a DBMS. (See Unit-I, Page 9, Q.11)

(b) Discuss in detail about database system architecture with neat diagram. (See Unit-I, Page 5, Q.7)

2. (a) Draw an E-R diagram for a banking enterprise with almost all components and explain. (See Unit-I, Page 42, Q.60)

(b) What are the responsibilities of a DBA ? Describe the three levels of data abstraction. (See Unit-I, Page 21, Q.34)

3. (a) Explain various key constraints with example. (See Unit-II, Page 51, Q.13)

(b) What is data dictionary ? What it stores ? How can this information be useful in DBMS. (See Unit-V, Page 222, Q.25)

4. (a) What is weak entity set and strong entity set and degree of a relation with example ? (See Unit-II, Page 48, Q.7)

(b) Consider the relation schema. Find out the relational algebra of the following queries –
Employee (employee name, street, city),
Work (employee name, company_name, salary)
Company (company name, city),
Manages (employee name, manager_name).
(i) Find the names and cities of residence of employees working for TCS.

(ii) Find the names, street and cities of residence of employees working for infosys and earning more than 20,000.
(iii) Find the name of employees working in the same city where they live.
(iv) Find the name of employees who are not working for WIPRO.
(v) Find the total and average salary paid by each company.
(See Unit-II, Page 107, Prob.21)

5. (a) How does tuple-oriented relational calculus differ from domain-oriented relational calculus ? (See Unit-II, Page 104, Q.69)

(b) Explain dangling tuple ? Is BCNF is a stronger normal form than 3NF ? Mention reason. (See Unit-III, Page 137, Q.32)

6. (a) Given the following set of relation schema R (A, B, C, D, E, F) {A → BC, E → CF, B → E and C → EF} compute closure of attribute set {A}.

(b) What is meant by lossless join decomposition ?
(See Unit-III, Page 130, Q.27)
A relation R = {A, B, C, D} has FD's F → {AB → C, C → D, D → A} is R is in 3NF.

7. (a) Explain two phase locking protocol and also list advantages of this protocol. (See Unit-IV, Page 176, Q.39)

(b) Why we need to do recovery in DBMS ? Explain various method use for recovery. (See Unit-IV, Page 162, Q.21)

8. (a) What are the main advantages of a distributed database system ?
(See Unit-V, Page 220, Q.21)

(b) Explain the following (any two) –
(i) Techniques for concurrency control
(See Unit-IV, Page 187, Q.54)
(ii) Web and mobile database (See Unit-IV, Page 200, Q.68)
(iii) Dedicated server and multi-threaded server.
(See Unit-V, Page 213, Q.13)

RGPV

CS-5003 (CBGS)
B.E. V Semester
EXAMINATION, November 2018
Choice Based Grading System (CBGS)
DATA BASE MANAGEMENT SYSTEM

Note : (i) Attempt any five questions.
(ii) All questions carry equal marks.

1. (a) Define Database Managemnt System (DBMS). What are the major components of this system ? Explain each component.
(See Unit-I, Page 7, Q.8)

(b) What are the different modulus present ? Explain in detail.
(See Unit-I, Page 7, Q.9)

2. (a) Draw an E-R diagram of university by determine entities of interest and the relationship that exist between these entities.

(b) Briefly explain the following –
(i) Functions of DBA (See Unit-I, Page 21, Q.22)
(ii) Generalization, aggregation and specialization.
(See Unit-I, Page 36, Q.53)

3. (a) Discuss different types of keys. For each case, give a suitab example. What is foreign key constraint ? Why is such constrai important ? (See Unit-II, Page 54, Q.2

(b) Discuss the different relational algebra operations. (See Unit-II, Page 90, Q.5

4. (a) Consider the following relations (primary keys are underlined) –

(i) account (<u>acc-no</u>, balance, branch-name)
(ii) depositor (<u>acc-no</u>, <u>cust-no</u>)
(iii) customer (<u>cust-no</u>, name, city)
(iv) loan (<u>loan-no</u>, amt, branch-name)
(v) borrower (<u>cust-no</u>, <u>loan-no</u>).

Solve the following queries using SQL –

(i) Find all customer-no and loan-no who have a loan at t 'Perryridge' branch.
(ii) Find all customer who has an account but no loan at the ban
(iii) Find branch-name and average account balance where avera account balance is greater than 1000. (See Unit-II, Page 84, Prob.

(b) Discuss the correspondence between ER model constructs a relational model constructs. Show how each ER model constru can be mapped to relational model and also discuss alternati mappings. (See Unit-II, Page 46, Q.

5. (a) Compute the clousure of the following FD for the relation sche

$R = \{A, B, C, D, E\}$
$A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$

List the candidate key R, reduce it in 3NF also.

(b) What is meant by term heuristic optimization ? Discuss the m heuristic that is applied during query optimization. What is cost bas optimization ? (See Unit-III, Page 148, Q.

6. (a) Define BCNF. How does it differ from 3NF ? Explain briefly. (See Unit-III, Page 116, Q.

(b) What is meant by concurrent execution of database transaction multiuser system ? Discuss why concurrency control is needed in give example. (See Unit-IV, Page 171, Q.

7. (a) Explain how strict 2-phase locking is implemented. Show with example. (See Unit-IV, Page 176, Q.

(b) What is distributed database system ? How it is different from centralized database system ? Give the uses of distributed syst (See Unit-V, Page 217, Q.

8. Write short notes on the following (any four) –

(i) Serializability (See Unit-IV, Page 158, Q.
(ii) Triggers (See Unit-II, Page 70, Q.
(iii) Comparison between OODMBS and DBMS (See Unit-IV, Page 196, Q.
(iv) Normalization (See Unit-III, Page 109, Q.
(v) Timestamp ordering protocol for concurrency control. (See Unit-IV, Page 180, Q.

CASH TRADERS INDORE *

DESIRE FOR SUCCESS ENDS HERE® **Shivani**®

Celebrating **19th** Anniversary

Ranked No. 1 in M.P. Since 2000

- Question & Answer Form
- Thoroughly Revised Edition
- Interactive Education Mode
- Complete Topics
- Simple Language
- Easy Solutions
- Errorless Book
- Teach Yourself Technique
- Special Tips on Important Topics

As it is same questions Asked in Previous RGPV Examination Papers

Helpline or any business enquiry through only
SMS/whatsapp on Mobile No: 7771088111 or
email us at: shivanipublications@yahoo.com
Kindly Visit our website at: www. shivanipublications.com
Authors are also welcome to communicate with us.

® Registered Trade Mark of

SHIVANI PUBLICATIONS®
Publisher of Technical Books
An ISO 9001: 2008 Certified Publication House

FOLLOW US ON...

@brocodeengineeringofficial

Brocode Engineering

@brocodeengineering

Brocode Engineering