Database Management System

SECOND YEAR DIPLOMA

(Code : 22319)

Maharashtra State Board of Technical Education (MSBTE)

Semester III – Computer Engineering Program Group (CO/CM/CW)

Strictly as per new revised 'I' Scheme w.e.f. academic year 2018-2019

Prof. Mahesh Mali

Ph.D. (Computer Engineering) (Pursuing),
M.E. (Computer Engineering), B.E. (Information Technology),
Oracle Certified PL/SQL Developer Associate (OCA),
SAS Certified Data Analyst.
Mumbai University.

TechKnowledge™ Publications

MDO13B Price ₹ 175/-

(Book Code : MDO13B)

Database Management System

Prof. Mahesh Mali

Second Year Diploma : Semester III (MSBTE)

[Computer Engineering Program Group(CO / CM / CW)]

First Printed in India : April 2010**(Mumbai University)**

First Edition : June2019**(As per 'I' Scheme)**

Second Revised Edition : September 2020

This edition is for sale in India, Bangladesh, Bhutan, Maldives, Nepal, Pakistan, Sri Lanka and designated countries in South-East Asia. Sale and purchase of this book outside of these countries is unauthorized by the publisher.

Printed at : 37/2, Ashtavinayak Industrial Estate,
Near Pari Company,
Narhe, Pune, Maharashtra State, India.
Pune – 411041

ISBN : 978-93-89233-95-7

Published by
TechKnowledge Publications

Head Office : B/5, First floor, Maniratna Complex,
Taware Colony, Aranyeshwar Corner,
Pune - 411 009. Maharashtra State, India
Ph : 91-20-24221234, 91-20-24225678.

[22319] (FID : MDO13) (Book Code : MDO13B)

(Book Code : MDO13B)

We dedicate this Publication soulfully and wholeheartedly,

in loving memory of our beloved founder director,

Late Shri.Pradeepji Lalchandji Lunawat,

who will always be an inspiration, a positive force and strong support behind us.

"My work is my prayer to God"

- Lt. Shri. Pradeepji L. Lunawat

Soulful Tribute and Gratitude for all Your
Sacrifices, Hardwork and 40 years of Strong Vision...

Preface

My Dear Students,

I am extremely happy to come out with this book on **"Database Management System"** for you. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

I present this book in the loving memory of **Late Shri. Pradeepji Lunawat,** our source of inspiration and a strong foundation of **"TechKnowledge Publications"**. He will always be remembered in our heart and motivate us to achieve our milestone.

I am thankful to Shri. J. S. Katre, Mr. Shital Bhandari, Shri. Arunoday Kumar and Shri. Chandroday Kumar for the encouragement and support that they have extended. I am also thankful to Seema Lunawat for technology enhanced reading, E-books support and the staff members of TechKnowledge Publications for their efforts to make this book as good as it is. I have jointly made every possible efforts to eliminate all the errors in thisbook. However if you find any, please let me know, because that will help me to improve further.

I am also thankful to my family members and friends for patience and encouragement.

- Prof. Mahesh Mali

❑❑❑

Syllabus

Unit	Unit Outcomes (UOs) (incognitivedomain)	TopicsandSub-topics
Unit - I **Database System Concept** Refer Chapters 1 to 4	1a. State the importance of DBMS over file processing in the given situation. 1b. Describe the overall structure of the given DBMS. 1c. Identify the relevant database model in the given situation. 1d. Draw the E-R diagram of the given database and identify relationship between the entities.	1.1 Concept of Data, Database, DBMS, Advantages of DBMS over File Processing System, Application of database. 1.2 Three Level Architecture for Database System. 1.3 Data Abstraction : Different Levels of Data Abstraction, Instance and Schema, Data Independence - Logical and Physical Independence. 1.4 Overall structure of DBMS. 1.5 Data Modeling : Record Based Logical Model- Relational, Network, Hierarchical. 1.6 Data modeling using the E-R Model : Entity Relationship Model, Strong Entity Set, Weak Entity Set, Types of Attributes, E-R Diagrams
Unit -II **Relational Data Model** Refer Chapters 5 to 7	2a. Explain the concept of RDBMS also appropriateness for the given problem. 2b. Design Normalized database structure in the given problem. 2c. Design SQL Queries to create Relational database and apply in the given data constraints. 2d. Identify the operators for queries implementation of the given problem.	2.1 Fundamentals of RDBMS - Record, fields, Data types, tables and database 2.2 Concept of RDBMS, E.F Codd's rule for RDBMS, Key concepts – Candidate key, Primary key, Foreign key. 2.3 Normalization : Normalization Concepts, Need of Normalization, Types of Normalization - 1NF, 2NF,3NF 2.4 Introduction to Structure Query Language, Data Types in SQL, Components ofSQL : DDL, DML, DCL, DQL 2.5 DDL Commands : CREATE, ALTER, DROP, TRUNCATE, DESC, RENAME 2.6 Data Integrity Constraint : Types of Data Integrity Constraint : IO Constraint – Primary Key, Foreign Key, Unique Key Constraint , Business Rule Constraint : Null, Not Null and Check Constraint. 2.7 DML Commands : INSERT, UPDATE, DELETE 2.8 DCL commands : COMMIT, SAVEPOINT, ROLLBACK, GRANT AND REVOKE 2.9 DQL Commands : SELECT 2.10 SQL Operators : Arithmetic Operators, Comparison Operators, Logical Operators, Set Operators, Range Searching Operators - Between, Pattern Matching operators - Like

Unit	Unit Outcomes (UOs) (incognitivedomain)	TopicsandSub-topics
Unit -III **Interactive SQL and Advance SQL : SQL Performance Tuning** Refer Chapters 8 to 11	3a. Write the given queries using relevant functions. 3b. Write query to combine the given multiple table using join. 3c. Design SQL queries to implement VIEWS on the given tables. 3d. Apply and drop INDEXES and SYNONYM on the given table.	3.1 In-built Functions : String, Arithmetic, 3.2 Date and Time, Aggregate Functions, 3.3 Queries Using Group By, Having and Order by Clause, Joins – Inner and Outer Joins, Sub Queries. 3.4 Views : Concept of View, The create view command, updating views, views and joins, views and sub-queries, Dropping views. 3.5 Sequences : Creating Sequences, Altering Sequences, Dropping Sequences 3.6 Indexes : Index Types, Creating of an Index - simple, unique and 3.7 Composite Index, Dropping Indexes. 3.8 Synonyms : Creating Synonyms, Dropping Synonyms
Unit -IV **PL/SQL Programming** Refer Chapters 12 to 16	4a. Write simple PL/SQL code using control structure and handle various exceptions in the given situation. 4b. Create cursor for retrieving multiple records in the given situation. 4c. Create and Execute stored procedures and functions in the given situation. 4.d. Create and apply database trigger using PL/SQL in the given situations.	4.1 Introduction of PL/SQL, Advantages of PL/SQL, The PL/SQL Block Structure, PL/SQL execution environment, PL/SQL data Types, Variables, Constants. 4.2 Control Structure : Conditional Control, Iterative Control, Sequential Control. 4.3 Exception Handling : Predefined Exception, User defined Exception. 4.4 Cursors : Implicit and Explicit Cursors, Declaring, Opening and closing a cursor, Fetching a Record from Cursor, Cursor for loops, Parameterized Cursors. 4.5 Procedures : Advantages, Creating, Executing and Deleting a Stored Procedure. 4.6 Functions : Advantages, Creating, Executing and Deleting a Function. 4.7 Database Triggers : Use of Database Triggers, how to apply database Triggers, Types of Triggers, Syntax for Creating Trigger, Deleting Trigger.

Unit	Unit Outcomes (UOs) (incognitivedomain)	TopicsandSub-topics
Unit -V **Database Security and Transaction Processing** **Refer Chapters 17 to 19**	5a. Provide security to the given database by assigning various privileges to the user. 5b. Create and manage the given database users. 5c. Explain the importance of Transaction in the given situation. 5d. Explain advantages of Database Backup and Recovery in the given situation.	5.1 Database Security : Introduction to Database security, Data security requirements, Types of Database Users-Creating, altering and Deleting Users. 5.2 Protecting the data within database - Database Privileges : Systems privileges and Object Privileges, Granting and Revoking Privileges : Grant and Revoke command. 5.3 Transaction : Concept, Properties and States of Transaction. 5.4 Database Backup - Types of failures, Causes of failures, Database Backup introduction, Types of Database Backups - Physical and Logical. 5.5 Database Recovery - Recovery concept, Recovery Techniques-Roll forward, Rollback

❑❑❑

INDEX

UNIT I

Syllabus : Concept of Data, Database, DBMS, Advantages of DBMS over File Processing System, Application of database. Three Level Architecture for Database System. Data Abstraction : Different Levels of Data Abstraction, Instance and Schema, Data Independence - Logical and Physical Independence. Overall structure of DBMS. Data Modeling : Record Based Logical Model-Relational, Network, Hierarchical. Data modeling using the E-R Model : Entity Relationship Model, Strong Entity Set, Weak Entity Set, Types of Attributes, E-R Diagrams

UNIT II

Syllabus : Fundamentals of RDBMS - Record, fields, Data types, tables and database, Concept of RDBMS, E.F Codd's rule for RDBMS, Key concepts - Candidate key, Primary key, Foreign key. Normalization : Normalization Concepts, Need of Normalization, Types of Normalization - 1NF, 2NF,3NF. Introduction to Structure Query Language, Data Types in SQL, Components of SQL : DDL, DML, DCL, DQL. DDL Commands: CREATE, ALTER, DROP, TRUNCATE, DESC, RENAME. Data Integrity Constraint : Types of Data Integrity Constraint : IO Constraint - Primary Key, Foreign Key, Unique Key Constraint , Business Rule Constraint : Null, Not Null and Check Constraint. DML Commands : INSERT, UPDATE, DELETE. DCL commands : COMMIT, SAVEPOINT, ROLLBACK, GRANT AND REVOKE. DQL Commands : SELECT. SQL Operators : Arithmetic Operators, Comparison Operators, Logical Operators, Set Operators, Range Searching Operators - Between, Pattern Matching operators - Like

UNIT III

Syllabus : In-built Functions : String, Arithmetic, Date and Time, Aggregate Functions, Queries Using Group By, Having and Order by Clause, Joins – Inner and Outer Joins, Sub Queries. Views : Concept of View, The create view command, updating views, views and joins, views and sub-queries, Dropping views. Sequences : Creating Sequences, Altering Sequences, Dropping Sequences. Indexes : Index Types, Creating of an Index - simple, unique and Composite Index, Dropping Indexes. Synonyms : Creating Synonyms, Dropping Synonyms

UNIT IV

Syllabus : Introduction of PL/SQL, Advantages of PL/SQL, The PL/SQL Block Structure, PL/SQL execution environment, PL/SQL data Types, Variables, Constants. Control Structure : Conditional Control, Iterative Control, Sequential Control. Exception Handling : Predefined Exception, User defined Exception. Cursors : Implicit and Explicit Cursors, Declaring, Opening and closing a cursor, Fetching a Record from Cursor, Cursor for loops, Parameterized Cursors. Procedures : Advantages, Creating, Executing and Deleting a Stored Procedure. Functions : Advantages, Creating, Executing and Deleting a Function. Database Triggers : Use of Database Triggers, how to apply database Triggers, Types of Triggers, Syntax for Creating Trigger, Deleting Trigger.

UNIT V

Syllabus : Database Security : Introduction to Database security, Data security requirements, Types of Database Users- Creating, altering and Deleting Users. Protecting the data within database - Database Privileges: Systems privileges and Object Privileges, Granting and Revoking Privileges : Grant and Revoke command. Transaction : Concept, Properties and States of Transaction. Database Backup - Types of failures, Causes of failures, Database Backup introduction, Types of Database Backups - Physical and Logical. Database Recovery - Recovery concept, Recovery Techniques-Roll forward, Rollback

❑❑❑

Database System Concept

Syllabus

Concept of Data, Database, DBMS, Advantages of DBMS over File Processing System, Application of database.

1.1 Introduction to DBMS

(MSBTE - W-13, S-15, W-15, S-17)

Q. Define the following terms :

(i) Data (ii) Database (iii) DBMS

W-13, S-15, W-15, S-17, 3 Marks

- Many of us are very much familiar with the term called as data. We come across term data regularly in our day to day life. The name of a person, price of book, number of students in a college, pin code of a city, etc. are some examples of data.
- In our daily life we have to remember bulk amount of data. But it is easier for us to remember only some amount of data.

Example :

- We may be in a position to tell accurately the age, height, income, educational qualification, residential address, etc. of our close friends.
- But it could be very difficult for us to memorize all this information for a large number of individuals in an organisation.

1. Data

- The facts and figures that can be recorded in system (computer system) and that have some special meaning assigned to it is called as data.
- **Example :** Data of a customer like name, telephone number, address, product purchased date etc.
- As need of data increased, there was need to develop a computer-based system for storing and managing data as a file system or information system.

2. Database

- A database is a collection of related data items stored at one place.
- **Example :** College database stores information about students, teachers, classes, subjects (All related data).
- A database is nothing but set of data having some relation between them.
- Database acts like a logical collection of relevant data. It is designed to offer an organized mechanism for storing, managing and retrieving stored information.
- Sample database structure,

Student table

Sid	Name	Class	Major

Course table

Cid	Name	Hours

Department table

Did	Name

Marks table

Sid	Cid	Marks	Grade

Fig. 1.1.1 : Sample student database

3. Database Management System (DBMS)

- A Database Management System (DBMS) is a collection of software or programs which helps user in creation and maintenance of a database (set of information). Hence it is also known as a **computerized record-keeping system.**
- DBMS is software system that helps in the process of defining, constructing, manipulating the database.
- Database management system has become an integral part of the information system of many organizations as it is used to handle huge amount of data.
- Computer-based Information Systems (IS) is capable of serving many complex tasks in a coordinated manner. Such systems handle large volume of data, multiple users and several applications in a centralized database environment.
- The heart of an Information System (IS) is database management system. This is because most IS have to handle huge amounts of data. This core module of an Information System is also called as Database Management System (DBMS).

  Examples

  - MS Access, Fox Pro by Microsoft.
  - Oracle by Oracle corp.
  - SQL Server by Microsoft.
  - Ingres, DB2 by IBM.

1.2 Characteristics of DBMS

Q. Explain the Characteristics of DBMS.

Q. Explain various advantages of Databases.

- The database approach has many characteristics due to which nowadays database has become an integral part of software industry.
- The characteristics we are going to learn in many different units are mentioned as follows :

1. Data integrity
2. Data security
3. Data independence
4. Transaction control - rollback
5. Concurrency control
6. Data recovery -backup and restore

1. Data integrity

- Integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency and correctness.
- Database integrity concern with the correctness and completeness of data in the database.
- This objective can never be guaranteed, one cannot ensure that every entry made in the database is accurate.
- Some examples of incorrect data are as follows :
  1. Student taking admission to branch which is not available in college.
  2. Employee assigned with non existing department.
  3. Sometime inconsistency introduced due to system failures.

2. Data security

- A DBMS system always has a separate system for security which is responsible for protecting database against accidental or intentional loss, destruction or misuse.
- Data in database should be given to only authorized users.
- Only authorized users should be allowed to modify the data.
- Authorized users are able to access data any time they want.

3. Data independence

Data Independence can be defined as the capacity to change data kept at one place without changing data kept at other locations.

4. Transaction control - rollback

- The changes made in database can be reverted back with help of rollback command.
- The changes can be saved successfully with help of commit data command.

5. Concurrency control

- The data in database can be accessed by multiple users at same point of time.

- Such operations allowed by sharing same data between multiple users.

6. Data recovery backup and restore

- Database recovery is the process of restoring the database to original (correct) state after database failure.
- The main element of database recovery is the most recent database backup.
- If you maintain database backup efficiently, then database recovery is very straight forward process.

1.3 File System v/s Database System

(MSBTE - W-14, W-15, W-16)

Q. State and explain four advantages of DBMS over file processing system. W-14, W-15, W-16, 4 Marks

1. Redundancy can be reduced

- As we are using relational approach for data organization, data is not stored in more than one location.
- Repetition of information can be avoided which in turn saves storage space.

2. Inconsistency can be avoided

With the usage of database, it is assured that all the users access actual or true data present in the database.

3. Data can be shared

- Multiple users can login at a time into the database to access information.
- They can manipulate the database in a controlled environment.
- **Example :** In yahoo portal, many users are accessing data in database in a controlled manner.

4. With a **centralized control of data**, the database system may be designed for an overall optimal performance for entire organization.

5. Standards can be enforced

- Standards (rules and regulations for coding and designing) can be enforced on the database to regulate the access to the database.
- Primary Key constraint or foreign key constraint can be enforced on database which will be helpful for accessing data from database.

6. Security restrictions can be applied

- Security is the process of limiting access to the database server itself for some users.
- It is the most important for security and needs to be carefully planned.

7. Integrity can be maintained

Through integrity, one can ensure only accurate data is stored within the database.

8. Data independence can be provided

- None of the users need to know the technical aspects of the database to access it.
- They are physically as well as logically independent to access the database.

9. New applications may be developed using the existing database.

1.3.1 Difference between File Processing and DBMS

Q. Differentiate between file system and database systems.

Database Management System	File Processing System
Computerized record - keeping system is used in DBMS	Collection of individual files accessed by applications programs is called File Processing System
DBMS allows flexible access to data	File - Processing System is designed to allow predetermined access to data
It co-ordinates both the physical and logical	It co-ordinates only the physical access to data
DBMS provides multiple user interface	Data is isolated in the file system
Unauthorized access is restricted in DBMS	Unauthorized access cannot be restricted
Redundancy can be controlled	Redundancy cannot be controlled

1.4 Applications of DBMS

(MSBTE - W-13, S-15, W-15)

Q. List any four applications of DBMS.

W-13, S-15, W-15, 2 Marks

Q. Explain various applications of Databases.

- The database approach also has some drawbacks which may create problems.

1. Cost of Operation
2. System Complexity
3. Requirements of Technical Staff
4. Database Failure
5. Extra Hardware
6. Size of Data
7. Data Conversion
8. Maintenance Cost

1. Cost of Operation

- DBMS software requires higher initial cost for hardware, software and trained staff.
- Based upon size and functionality of organization there may be significant investment is involved.
- Also, organization needs to invest recurring cost on annual maintenance.

2. System Complexity

- To achieve all advantages of Database, DBMS need to be extremely complex software.
- Database developer, designer, DBA and End user of database must have necessary skills.
- There may be loss of data or database failure due to complex data environment.

3. Requirements of Technical Staff

To use system, we need a skilled technical staff for efficient use of available data.

4. Database Failure

- The database system can fail at any moment which may cause loss of data.
  - (i) Hardware Failure
  - (ii) Memory Failure
  - (iii) Network Failure
  - (iv) Database System Failure

5. Extra Hardware

To use advance uses we need lot of extra hardware to increase storage and processing power of computer.

6. Size of Data

The velocity of data growth is enormous so, day by day the memory requirements will go on increasing.

7. Data Conversion

- The data migration may be required if DBMS software is upgraded
- Lot of challenges are involved in data migration process.

8. Maintenance Cost

- The cost of maintenance is recurring cost.
- To keep data secure we need protection of data, it needs maintenance cost
- To achieve all advantages of Database, DBMS need to be extremely complex software.

Review Questions

Q. 1 Define data, database and DBMS.

Q. 2 State five main advantages of DBMS.

Q. 3 State the characteristics of DBMS.

Q. 4 Write the advantages of Database System over a file system.

Q. 5 Differentiate between file system and database management system.

Q. 6 List significant difference between file processing system and database management system.

Q. 7 Write the characteristics of DBMS.

Q. 8 List applications of DBMS.

❑❑❑

Architecture of Database

Syllabus

Three Level Architecture for Database System

Data Abstraction : Different Levels of Data Abstraction, Instance and Schema, Data Independence - Logical and Physical Independence

Overall structure of DBMS

2.1 Three Level Architecture for Database System

Q. Explain three-level architecture of DBMS.

Q. State and explain various levels of database abstraction.

Q. Explain physical, conceptual and view level abstraction of DBMS.

1. Introduction

- The goal of the three-schema architecture is to separate the front end (Presentation Tier) and the back end (Database Tier).
- The three-schema architecture is a tool with which the user can visualize the schema levels in a DBMS.
- Many DBMS systems do not separate the three levels completely, but support the three-schema architecture to some extent.
- A description of data in terms of a data model is called a schema.
- The description of a database is called **database schema**, which is specified during database design and it is not expected to change frequently.

2. Database architecture

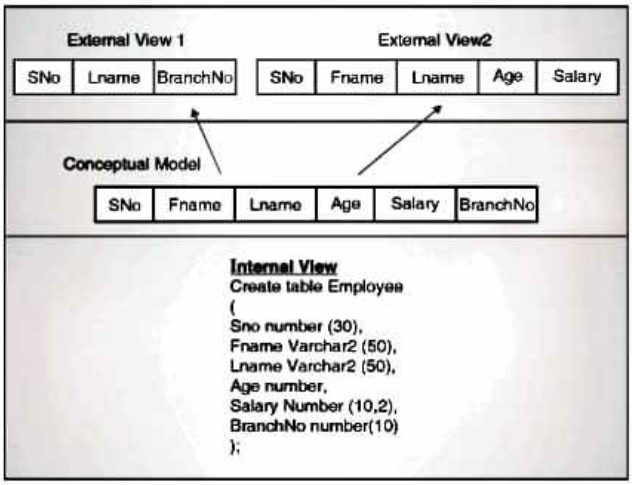


Fig. 2.1.1 : Database schema levels

(I) Internal Level or Physical Level

- The internal level is very close to physical storage of data.
- This level describes the physical storage structure of the data in database.
- The internal (or physical) database is stored on secondary storage devices, mainly the magnetic disk.
- Describes the complete details of data storage and various available access methods for the database.

- At its ground level, it is stored in the form of bits with the physical addresses on the secondary storage device.
- At its highest level, it can be viewed in the form of files and simple data structures.
- **Internal View/ Schema**
  - The internal view defines the various stored data types and specifies the indexes which exist, how that stored fields are represented and so on.
  - The internal schema uses a physical data model.

Example :

```
Create table Employee
{
    Sno number (30),
    Fname varchar2 (50),
    Lname varchar2 (50),
    Age number,
    Salary number (10, 2),
    BranchNo number (10)
}
```

(II) Conceptual Level

- This level describes the structure of the whole database for a group of users.
- The conceptual model is also called as the data model or we can say data model is used to describe the conceptual schema when a database system is implemented.
- The conceptual schema hides the internal details of physical storage and targets on describing entities, data types, relationships and constraints.
- The conceptual schema contains all the information to build relevant external records. As the conceptual model is derived from the physical model.
- **Conceptual View / Schema**
  - The conceptual view is a representation of the entire content of the database.
  - The conceptual view includes definitions of each of the various conceptual data types.

(III) External Level or View Level

- The external level is the one closest to the user, i.e., it is related with the way data is viewed by individual end users.
- The external level includes a number of user views or external schemas.
- Each external schema describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group.
- External views are the proper interface between the user and the database, as an individual user can hardly be expected to be interested in the entire database.
- The external model is derived from the conceptual model.
- **External View / Schema :** External schema consists of definitions of each of the various external data types in that external view.

2.2 Data Abstraction

(MSBTE – S-14, S-15, W-17, W-18)

Q.	What is data abstraction ?	S-14, S-15, 2 Marks
Q.	Describe data abstraction with neat diagram.	W-17, 4 Marks
Q.	Define data abstraction.	W-18, 1 Mark

Introduction

- A description of data in terms of a data model is called a schema.
- The description of a database is called **database schema**, which is specified during database design and it does not expect to change frequently.

(I) Internal Schema

- This level describes the physical storage structure of the data in database.
- Describes the complete details of data storage and various available access methods for the database.
- **Internal View/Schema :** The internal view defines the various stored data types and specifies the indexes

which exist, how that stored fields are represented and so on.

(II) Conceptual Schema

- The conceptual schema hides the internal details of physical storage and targets on describing entities, data types, relationships and constraints.
- The conceptual schema contains all the information to build relevant external records. As the conceptual model is derived from the physical model.
- **Conceptual View / Schema**
  - The conceptual view is a representation of the entire contents of the database.
  - The conceptual view includes definitions of each of the various conceptual data types.

(III) External Schema

- The external level is the one closest to the user, i.e., it is related with the way data is viewed by individual end users.
- External schema describes the segment of the database that is required for a particular user group and hides the rest of the database from that user group.
- External views are the proper interface between the user and the database, as an individual user can hardly be expected to be interested in the entire database.
- **External View / Schema :** External schema consists of definitions of each of the various external data types in that external view.

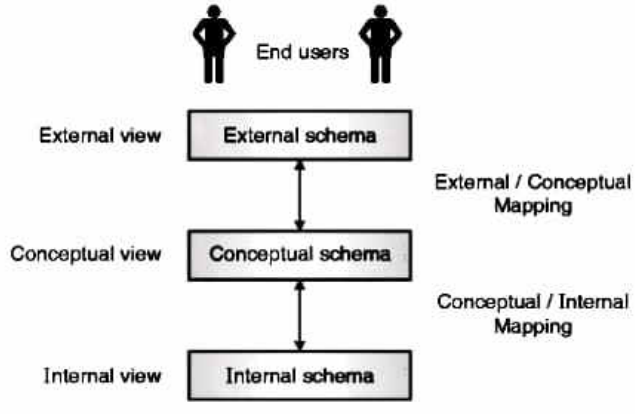


Fig. 2.2.1 : Data Abstraction

(IV) Mapping

- The processes of transforming requests and results between various levels of architecture are called mappings.
- These mappings may be time-consuming, so small databases do not support external views.
- **External / Conceptual Schema Mapping :** The DBMS must transform a request on an external schema into a request against the conceptual schema.
- **Conceptual / Internal Schema Mapping :** A certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

2.3 Instance and Schema

Q. What do you mean by Database Instance?
Q. Explain type of database schema in details.

- The description or design of a database is called **database schema**, which is specified during database design and it does not expect to change frequently.
- The data stored in database at any fixed timestamp is called **instance** of database.
- Database schema defines the attributes (columns) in tables whereas the value of these attributes (Columns) at any moment is called the instance of that database.

1. Schema

- A description of data in terms of a data model is called a database schema.
- The description of a database is called **database schema**, which is specified during database design and it does not expect to change frequently.
- A database schema is structure of database represents the logical view of the entire database.
- It defines how the data is arranged and how the relations among them are associated.
- It also represents all the constraints applied on the data.
- A database schema defines its database entities and the relationship among them.

2. Types of database schema

(a) Physical/Internal Schema
(b) Logical Schema
(c) View/External Schema

- The design of a database at physical level is called physical schema, how the data stored in storage is explained at this level.
- The design of database at logical level is called logical schema, this level gives tabular structure of data.
- The design of database at view level is called view schema. This generally describes user and database system interaction.

3. Example of database schema

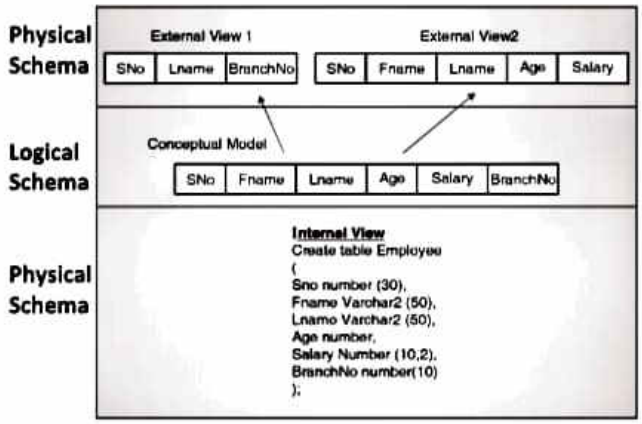


Fig. 2.3.1

4. Instance

- The data stored in database at any fixed timestamp is called **instance** of database.
- The data stored in table at any moment of time is called the instance of that database.

Example

- A Student table has 500 records, so today the instance of the database has 100 records.
- If we add another 100 records in this table by tomorrow so the instance of database tomorrow will have 600 records in table.
- The data stored at a particular moment is called the instance of database, instance will change over time when we add or delete data from the database.

2.4 Data Independence

(MSBTE - S-14, W-15)

Q. Describe data independence with its types.
S-14, W-15, 4 Marks

Q. What do you mean by data independence?

- Concept of data independence can be explained with help of 3 schema architecture.
- The three-schema architecture can make it easier to achieve true data independence.

Definition

Data Independence can be defined as the capacity to change one level of schema without changing the schema at the next higher level.

2.4.1 Types of Data Independence

Types of data independence are as follows :

(a) Logical data independence
(b) Physical data independence

(a) Logical data independence

- Logical data independence is a capacity to change the conceptual schema without any changes to external schemas. (or application programs)
- Separating the external views from the conceptual view enables us to change the conceptual view without affecting the external views. This separation is sometimes called logical data independence.
- **Example :** We may change the conceptual schema by removing a data item. In this case the external schemas that refer only to the remaining data should not be affected.

(b) Physical data independence

- Physical data independence is a capacity to change the internal schema without any changes to conceptual schema.

- The separation of the conceptual view from the internal view enables us to provide a logical description of the database without the need to specify physical structures. This is often called physical data independence.
- **Example :** By creating additional access paths to improve the performance of retrieval. If the data remains same as before in the database, then we should not change the conceptual schema.

2.4.2 Difference between Logical Data Independence and Physical Data Independence

Q. Differentiate between Logical data independence and Physical data Independence.

Q. Compare logical data independence and Physical data Independence.

Logical data independence	Physical data independence
The capacity to change the conceptual schema without any changes to external schemas.	The capacity to change the internal schema without any changes to conceptual schema.
Modify conceptual schema when structure of database is altered.	Modifications are performed to improve performance.
It only provides immunity to external schema and application program.	It provides independence and immunity to conceptual and external schema.
It is difficult.	It is easy.

2.5 Overall Structure of Database

(MSBTE – W-13, W-14, S-15, S-16, W-16)

Q. Draw the overall structure of DBMS.

W-13, W-14, S-15, W-16, 4 Marks

Q. Enlist different components of DBMS.

S-16, 2 Marks

Components of a database system :

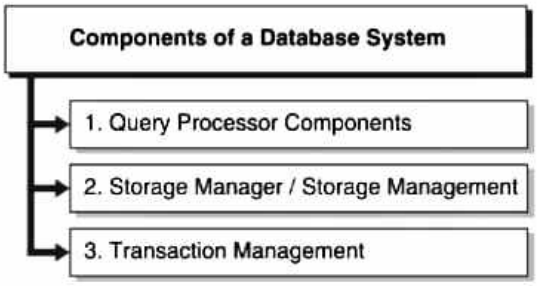


Fig. 2.5.1

- The storage manager is important because databases typically require a huge amount of storage space.

2.5.1 Query Processor Components

Q. Write a short notes on : Query processor.

1. Introduction

The query processor will accept query from user and solves it by accessing the database.

2. Parts of query processor

(i) DDL interpreter
(ii) DML compiler
(iii) Query evaluation engine

(i) DDL interpreter

This will interpret DDL statements and fetch the definitions in the data dictionary.

(ii) DML compiler

- This will translate DML statements in a query language into low level instructions that the query evaluation engine understands.
- A query can usually be translated into any number of alternative evaluation plans for same query result DML compiler will select best plan for **query optimization**.

(iii) Query evaluation engine

This engine will execute low-level instructions generated by the DML compiler on DBMS.

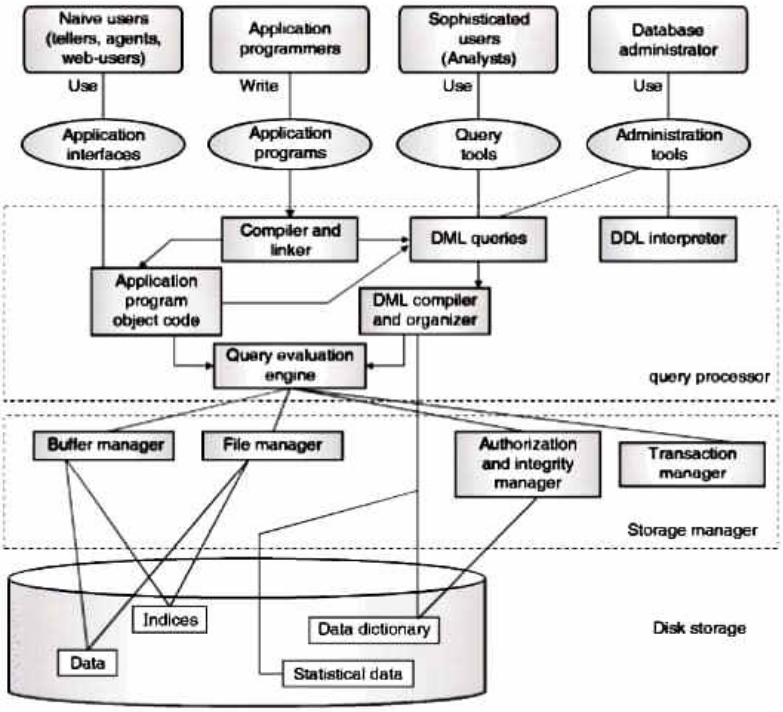


Fig. 2.5.2 : Components of DBMS

2.5.2 Storage Manager / Storage Management

Q. Write a short note on : Storage management.

- A **storage manager** is a program module which acts like interface between the data stored in the database and the application programs and queries submitted to the system.
- The data is stored on the disk using the file system.
- The storage manager is programme which is responsible for the interaction with the file manager.
- The storage manager translates the various databases language statements into low level file system commands.
- Thus, the storage manager is responsible for storing, retrieving and updating data in the database.
- The storage manager components include :
  1. **Authorization and integrity manager :** Checks for integrity constraints and authority of users to access data.
  2. **Transaction manager** : Which ensures that the database remains in a consistent (correct) state although there is system failures.
  3. **File manager** : Which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
  4. **Buffer manager** : Which is responsible for retrieving data from disk storage into main memory. The buffer manager is an important part of the database system, as it enables the database to handle data sizes that are much larger than the size of main memory.
- Data structures implemented by storage manager,
  1. **Data files :** Stored in the database itself.
  2. **Data dictionary** : Stores metadata about the structure of the database.
  3. **Indices :** Provide fast access to data items.

2.5.3 Transaction Management

Q. Write a short note on : Transaction management.

- A transaction is a series of small database operations that together form a single large operation.
- A transaction is started by issuing a BEGIN TRANSACTION command. Once this command is executed the DBMS starts monitoring the transaction.

- All operations executed after a BEGIN TRANSACTION command are treated as a single large operation.
- Application programs use transactions to execute sequences of operations when it is important that all the operations are successfully completed.
- Transaction management component will ensure the atomicity and durability properties.

2.6 Database Users

(MSBTE - W-18)

Q. Explain the four roles of database administrator. **W-18, 4 Marks**

Q. What are the different types of database system users ?

Q. Write short note on : Responsibilities of database administrator.

Q. What is role of DBA in database management system ?

There are four broad categories of users as per their needs to access data :

1. Naive users
2. Application programmers
3. Sophisticated users
4. Specialized users

1. Naive users

- Naive users are users who interact with the system using application programs that have been developed previously.
- For example, Student wants to pay fees Rs.50 then accountant will invokes a program called fees_payment. This program asks the accountant for the amount of fees to be paid.
- The typical graphical user interface for naive users is a kind of form interface, where the user can fill in appropriate fields of the form.
- A given end user can access the database via one of the applications or can use an interface provided as an integral part of the database system software (such interfaces are also supported by means of applications, of course, but those applications are built-in, not user-written, e.g., query language processor)
- Naive users can read reports generated from the database.

2. Application programmers

- Application programmers responsible for writing application programs that use the database.
- Application programmers are developers or computer professionals who write application programs.
- Application programmers develop user interfaces using any preferred language.
- Rapid Application Development (RAD) tools are available nowadays that enable an application programmer to construct application without writing code.
- Some programming languages combine control structures with database language statements. Such languages, sometimes called fourth-generation languages.

3. Sophisticated users

- Sophisticated users interact with application without writing programs by using a database query language.
- This query will be solved by query processor.
- Online Analytical Processing (OLAP) tools is used to view summaries of data in different ways which helps analysts (e.g. sales of region, city etc.) with OLAP analysts can use data mining tools, which help them find certain kinds of patterns in data.

4. Specialized users

- Creates the actual database and implements technical controls needed to enforce various policy decisions.
- Specialized users are sophisticated users who develop database applications.
- The DBA is also responsible for ensuring that the system operates with adequate performance and for providing a variety of other related technical services.

2.6.1 Database Administrator (DBA)

- The database administrator is responsible for the overall planning of the company's data resources, for the design of data and for the day-to-day operational aspects of data management.

- A database administrator is a person responsible for the installation, configuration, up gradation, maintenance and monitoring databases in an organization.
- The overall planning of corporate data is the strategic aspect of the database administration function and involves company-wide planning of existing data and assessment of organization-wise data standards.
- Responsibilities of DBA in enterprises :
  - Designing schema.
  - Deciding on the storage and access methods.
  - Selecting database software and hardware.
  - Designing the means of reorganizing databases periodically.
  - Designing database searching strategies.
  - Designing authorization checks and validation procedures.
  - Designing restart and recovery procedures to take care of system crashes.
  - Specifying techniques for monitoring database performance.
- The operations management of database administration deals with data problems arising on a day-to-day basis. Specifically, the responsibilities include :
  - Investigation of errors found in the data.
  - Supervision of restart and recovery procedures in the event of a failure.
  - Supervision of reorganization of databases.
  - Initiation and control of all periodic dumps of data.
- Skills required for DBA
  - Good communication skills.
  - Excellent knowledge of databases architecture and design and RDBMS (Oracle, SQL Server etc.)
  - Knowledge of Structured Query Language (SQL).
- In addition, this aspect of database administration includes maintenance of data security, which involves maintaining security authorization tables, conducting periodic security audits, investigating all known security breaches.
- To carry out all these functions, it is crucial that the DBA has all the accurate information about the company's data readily on hand. For this purpose, he maintains a data dictionary.
- The data dictionary contains definitions of all data items and structures, the various schemes, the relevant authorization and validation checks and the different mapping definitions.
- It should also have information about the source and destination of a data item and the flow of a data item as it is used by a system. This type of information is a great help to the DBA in maintaining centralized control of data.

Review Questions

Q. 1 Explain three level architecture of DBMS.

Q. 2 Explain Abstraction of database management system.

Q. 3 Explain various database schema.

Q. 4 Explain the following terms : Data independence and its types.

Q. 5 Explain the various components of database schema.

Q. 6 Write short note on: Query processing and Storage Manager.

Q. 7 What are the role of a DBA.

Q. 8 Write short note on various database users.

❑❑❑

CHAPTER 3

UNIT I

Data Model

Syllabus

Data Modeling : Record Based Logical Model-Relational, Network, Hierarchical.

3.1 Database Model

(MSBTE - S-16, W-16)

Q.	**List various data models.**	**S-16, 2 Marks**
Q.	**List any two data models.**	**W-16, 2 Marks**
Q.	**What is data model?**	

- Data model will give an idea how your final system or software will look like after development is completed.
- This concept is exactly like real world modelling in which before constructing any project (Bridges, Buildings, Towers etc.) engineers create a model for it, this model gives an idea about how your project will look like after construction.
- A data model is an overview of a software system which describes how data can be represented and accessed from software system after its complete implementation.
- Data models define data elements and relationships among various data elements for a specific system.
- Different types of data models :
  1. Record based Logical Model -Relational Model
  2. Network Model
  3. Hierarchical Model

3.1.1 Relational Database Model

Q. Explain Record based logical model.

(a) Introduction

- The relational model first proposed by **E. F. Codd** hence he is known as father of Relational Model.
- Relational database was an attempt to simplify database structure by making use of tables and columns.
- Tables are known as **"relations"**, columns are known as **"attributes"** and rows (or records) are known as **"tuples"**.
- A relational database is a collection of 2-dimensional tables which consists of rows and columns.

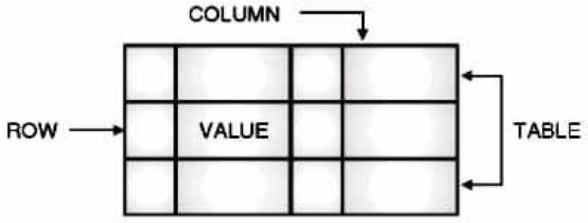


Fig. 3.1.1 : Relational model

(b) Basic building blocks / structure

- Tables are known as **"relations"**, columns are known as **"attributes"** and rows (or records) are known as **"tuples"**.
- This Model uses collection of tables to represent relationships amongst the data.
- In this model, each database item is viewed as a record with attributes. A set of records with similar attributes is called a **table**. Each table contains a record of a particular type.
- The database uses Relational model called as **RDBMS** (Relational Database Management System)

- Let us consider a college database it contains information about,
  - **Student table :** Contains information of all students.
  - **Faculty table :** Contains information of all faculties (Teaching and non-teaching).
  - **Account table :** Contains information about all fees transactions happened in college, salary records for faculties and all other accounts details.
  - **Class table :** Maintains information about various classes available in college.

Student table				
STUD_ID	**Name**	**Age**	**Std**	**Div**
105	Mahesh	25	BE	A
106	Suhas	28	FE	B
107	Jay	29	SE	A
108	Sachin	30	TE	D

Faculty table				
FAC_ID	**Name**	**Type**	**Age**	**Salary**
205	Jay	Teaching	25	20000
206	Om	NONT	26	12000
207	Yogesh	NONT	24	15000

Account table				
ACC_ID	**Tran**	**Date**	**Receiver**	**Amt**
1	Cash	1/1/2011	Jay	12000
2	Cheque	1/1/2011	Mahesh	15000
3	Cash	1/1/2011	Yogesh	1000

Class table			
CLASS_ID	**Year**	**Branch**	**Room**
1	FE	IT	102
2	SE	IT	103
3	FE	CS	105

(c) Examples

Most of the popular commercial DBMS products like Oracle, Sybase, MySQL etc. are based on relational model.

(d) Advantages

(i) Relational algebra : A relational database supports relational algebra and various operations of the set theory (like union, intersection etc.).

(ii) Dynamic views : In a RDBMS, a view is not a part of the physical schema, it is always dynamic. Hence changing the data in a table also changes the data present in view.

(iii) SQL (Structured Query Language) : For data access in RDBMS we have English like query language called as structured Query language (SQL) which can be used for accessing data from RDBMS. Most of the database vendors support the SQL standard.

(iv) Excellent data security : Relational databases support the concept of user rights (every user is assigned with some database permission called as user rights), thus meeting the security needs of databases.

(v) The other advantages of relational databases are performance and support to new hardware technologies and also flexible for all types of data needs.

(vi) Relational databases are scalable and provide good support for the implementation of distributed systems and other advanced database systems.

3.1.2 Network Model

Q. Explain network database model.

(a) Introduction

- Like the hierarchical model, this model also uses pointers toward data but there is no need of parent to child association so it does not necessarily use a downward tree structure.
- This model is used in network databases.

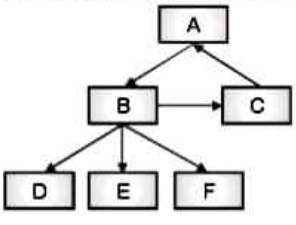


Fig. 3.1.2

(b) Basic building blocks / structure

- This database model is similar to hierarchical model in some aspect.

- There are some concepts involved in network model as follows :
  - **SET :** A relationship between any two record types is called as a set.
  - **RECORD :**
    (a) Owner record is like parent record in hierarchical model.

    Example : Professor teaches to students in this case professor is owner entity.

    (b) Parent record is like child record in hierarchical model.

    Example : Professor teaches to students in this case student is parent entity.
- A relationship can be 1: N or in this model we can show M: N relationship.
- **Example:** Many professors in college teaches too many students.

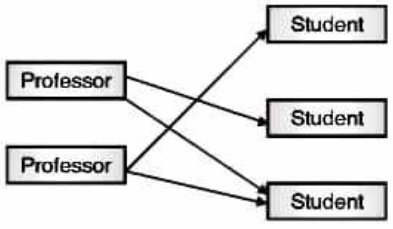


Fig. 3.1.3 : College hierarchy

(c) Example

IDS (Integrated Data Store) is one of the DBMS products based on network models. This was developed by joined efforts of IBM and North American Rockwell known as Information Management System.

(d) Advantages

(i) Simple design : The network model is simple and easy to design and understand.

(ii) Ability to handle many types of relationship

- The network model can handle the one-to-many or many-to-many or other relationships.
- Hence network model manages multi user environment.

(iii) Ease of data access

- In a network model an application can access a root (parent) record and all the member records within a set (child).
- Provides very efficient and high-speed retrieval.

(iv) Data Integrity

In a network model, no member can exist without a parent entity. A user must therefore first define the root record and then the child record.

1. Data Independence

- In network model application programs work independently of the data.
- Any changes made in the data do not affect the application program.

2. Conformance to standards

In a network model, facilitates the administrator's portability by offering data creation by DDL (Data Definition Language) and DML (Data Manipulation Language).

(e) Disadvantages

(i) System complexity

(a) In a network model, one data record is accessed at a time.

(b) This can increase the complexity of system for accessing multiple records at a time.

(ii) Lack of structural independence

Any changes made to the database structure (or data) require the application programs to be modified before it can access data.

3.1.3 Hierarchical Model

Q. Explain Hierarchical model.

(a) Introduction

- This was developed by joined efforts of IBM and North American Rockwell known as Information management system.
- It was the first DBMS model.
- The data is sorted hierarchically, either by top down or bottom up approach of designing.
- This model uses pointers to navigate between stored data.
- This model represents data as a hierarchical tree Hierarchical model

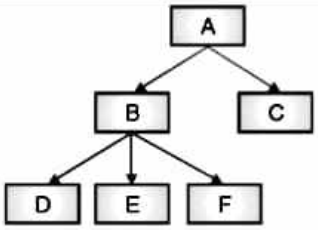


Fig. 3.1.4

(b) Basic building blocks / structure

- Let us consider simple organizational structure as shown in Fig. 3.1.5.

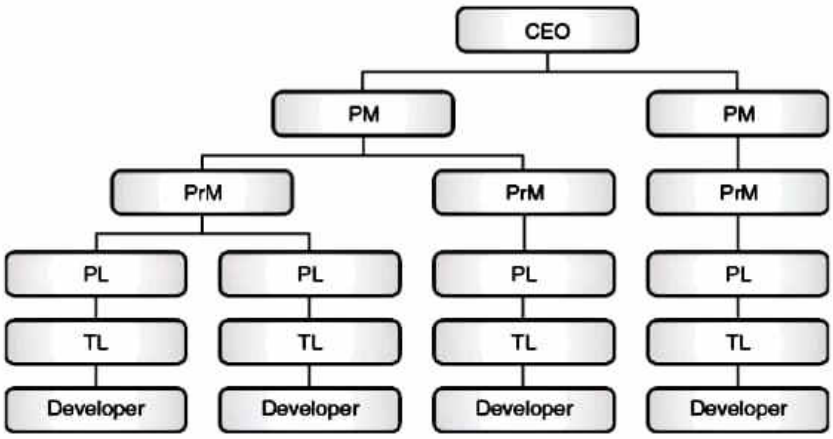


Fig. 3.1.5 : Organizational hierarchy

- CEO is root node having many DU Heads below that managers who manages multiple project leaders who organizes developer as shown below :

  CEO → Program Manager (PM) → Project Manager (PrM) → Project Leader (PL) → Team Leader (TL) → Developer

(c) Business rule

One parent node may have many child nodes, but one child cannot have more than one parent.

(d) Example

One of the popular DBMS based on hierarchical model is Information Management System (IMS) from IBM.

(e) Advantages

(i) Conceptual simplicity

Relationship between various levels is logically very simple. Hence database structure becomes easier to view.

(ii) Database security

Security is given by DBMS system itself it does not depends on whether programmer has given security or not.

(iii) Simple creation, updation and access

- Hierarchical model is simple to construct with help of pointers or similar concepts and very simple to understand also adding and deleting records is easy in tree structure using pointers.
- This file system is faster and easy data retrieval through higher level records in tree structure.

(iv) Database integrity

- There is always Parent Child Association between different levels of records in files.
- Hence child record is attached with the parent record which maintains the integrity.

(v) Data independence

- DBMS itself maintains data independence so it reduces program efforts and its maintenance.
- If one part of database code is changed no need to change other part of database coding.

(vi) Efficiency

This model have's good performance when database contains large amount of data in which one record has many related records like a class contains many students studying in it.

(f) Disadvantages

(i) Complex implementation

Only data independence is not enough for designer and programmers to build database system they need to have knowledge of physical data storage which may be complex.

(ii) Difficult to manage

- Any change in a location of data needs change in all application programs that accesses changed data.
- Data access is restricted by Pointer path.

(iii) Lack of structural independence

- Change in database structure does not affects data access is called as **structural independence.**
- Advantage of data independence may be restricted by **structural independence.**

(iv) Complex application programming

- Programmers must know how physical data is stored in order to access data.
- Even the programmer knows path of data storage.

(v) Limitations in implementation

- Generally 1 : N relationship can be implemented in hierarchical model.

  Example : one teacher teaches too many students.

- It is very difficult to implement M:N relationship in hierarchical model.

  Example : many students can have many books.

- Query optimization is generally not possible or it is possible only up to certain extent.

3.2 Object Oriented Database Models

Q. Explain object model.

(a) Introduction

- The data is stored in the form of objects, which are structures called *classes* that display the data within it.
- The fields are instances of these classes called as objects. This model is used in file management systems.
- The DBMS (Database Management System) developed with help of such model is called as OODBMS (Object Oriented Database Management System).
- Object oriented databases evolved to handle more complex applications such as databases for scientific experiments, geographic information system, engineering design and manufacturing.
- This model represents DB in terms of objects, their attributes and their behaviours.

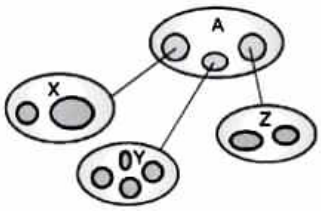


Fig. 3.2.1

(b) Advantages

(i) OO (Object Oriented) features provide a clear modular structure which is good for defining abstract data types where internal implementation details are hidden.

(ii) This model is easy to maintain and modify existing code as we can create new model with small change in existing.

(c) Disadvantages

(i) This model is often provided through object-oriented languages such as C++ and Java.

(ii) Practically very complex and inapplicable many a times.

3.3 Comparison of All Data Models

(MSBTE- W-18)

Q Distinguish between network model and hierarchical model (any 4points). **W-18, 4 Marks**

Q. Compare various data models.

Sr. No.	Parameter	Hierarchical Model	Network Model	Relational Model	Object oriented Model
1.	Data Independence	Yes	Yes	Yes	Yes
2.	Structural Independence	No	No	Yes	Yes
3.	Storage Type	Segment	Record	Relation (Table)	Class
4.	Single row storage	Segment occurrence	Current Record	Row(tuple)	Object – instance of class
5.	Basic storage	Segment field	Record field	Relation Attribute	Object attribute
6.	Storage Identifier	Sequence Field	Record key	Key	Object identifier
7.	Advantages	*Promotes Data sharing, *conceptual simplicity, *Handle simple relationships (1:N) *Flexible data access	*conceptual simplicity, *Handle complex relationships (M:N)	*Tabular view, *Ad-hoc query capability, *improves management and implementation simplicity	*Semantic contents, *Promotes data integrity
8.	Disadvantages	*Complex implementation, *Lack of standards *Limited implementations (No DML)	*Simplicity limits efficiency *Complex navigational system	*Hardware and software required	* Slow development of standards * complex navigation * Slow transactions if overload on system
9.	Examples	IMS (Information Management System)	IDS (Integrated Data Store)	Oracle, DB2, SQL SERVER etc.	Database using C, java etc

Review Questions

Q. 1 What is data model?

Q. 2 Explain Record based logical model.

Q. 3 Explain network database model.

Q. 4 Explain Hierarchical model.

Q. 5 Explain object model.

Q. 6 Compare various data models.

Q. 7 Explain object oriented database model . State its advantages and disadvantage.

❑❑❑

CHAPTER 4

UNIT I

Entity Relationship Model

Syllabus

Data modeling using the E-R Model : Entity Relationship Model, Strong Entity Set, Weak Entity Set, Types of Attributes, E-R Diagrams

4.1 ER Model

- In 1976, Scientist Chas hen developed the **Entity-Relationship (ER) model** which is a high-level conceptual data model.
- ER diagram is the first step of database design to specify the desired components of the database system and the relationships among those components.
- ER model define data elements and relationships among various data elements for a specified system.
- ER Diagrams having components,
  1. Entity
  2. Attributes
  3. Relationships

4.2 Entities

(MSBTE - S-14, W-14, S-16, W-16, S-17)

.Q. State weak and strong entity set.

S-14, W-14, 2 Marks

Q. Define entity. S-16, 2 Marks

Q. Explain strong entity and weak entity set.

W-16, S-17, 4 Marks

Q. What is weak entity set and how it is represented in E-R diagram ?

Q. Describe the following integrity constraints using suitable example: Entity

(1) Introduction

- A basic component of ER model.
- An **entity** is anything in real world with its own independent existence.

  Example : Student, faculty, subject having independent existence.
- An entity may be an object with a physical existence or it may have logical existence.

  Example : Department, section, adult (age>18) may have physical existence or not.
- Each entity has its own properties which describes that entity such properties are known as **attributes.**

(2) Entity set

Entity set is collection of all entities of same type.

(a) Strong entity set

(b) Weak entity set

(a) Strong entity set

Entity type which has its own key attributes by which we can identify specific entity uniquely is called as strong entity Set.

Example

- In case of Employee entity any specific employee can be identified by his Employee_id which is primary key of employee entity.

- In case of student in class each student is identified by unique roll number which is his primary key.

Strong entity type is represented by single rectangle.

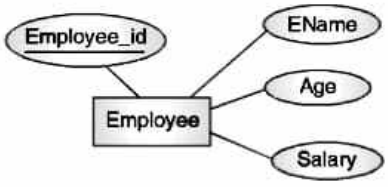


Fig. 4.2.1 : Employee entity

(b) Weak entity Set

- Entity type which cannot form distinct key from their attributes and takes help from corresponding strong entity is called as weak entity Set.
- These types of entities are dependent on strong entity for primary key.
- For some weak entities we assign virtual primary key. Such virtual primary key of weak entity is called as '**discriminator**'.
- Weak entity type is represented by double rectangle.
- **Example :** In case of "Dependent" entity depends on "employee" entity for primary key.

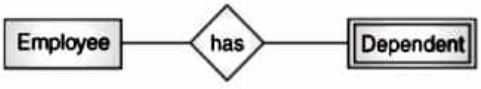


Fig. 4.2.2 : Weak entity "dependent"

4.3 Attributes

(MSBTE - S-16, S-17, W-17)

Q.	Define attribute.	S-16. 1 Mark
Q.	Explain single value and multi-value attribute of E-R model.	S-17. 4 Marks
Q.	Explain any 4 types of attributes.	W-17. 4 Marks

- Various properties that describe an entity are known as **attributes**.
- The attribute value that describes each entity becomes a major part of data stored in database.
- A particular entity will have some value for each of its attributes.

Example : For an employee of ID 30 value of name attribute is 'Jayendra'.

Employee entity may be described by attributes name, age, phone etc.

4.3.1 Types of Attributes

(a) Simple and composite attributes

(i) Simple attributes

Simple attributes which cannot be divided into subparts.

Example : Salary of employee cannot be broken down into some parts.

Fig. 4.3.1(a) : Simple attribute

(ii) Composite attributes

Composite attributes which can be divided into subparts.

Example : Name attribute can be divided into First_Name and Last_Name.

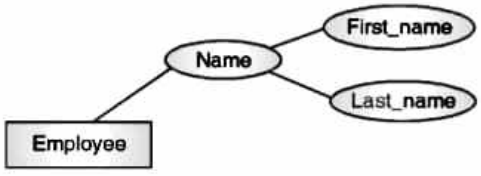


Fig. 4.3.1(b) : Composite attributes

(b) Single valued and multi valued attributes

(i) Single valued attribute

The attribute having **atomic** (only one) value for a particular entity is called as single valued attribute.

Example : Each student has only one roll number.

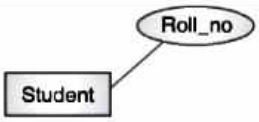


Fig. 4.3.2(a) : Single valued attributes

(ii) Multi valued attributes

The attribute having many values for a particular entity is called as multi-valued attribute.

Example : Each Employee has multiple mobile numbers.

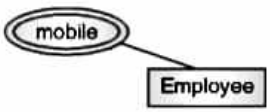


Fig. 4.3.2(b) : Multi valued attributes

(c) Stored and derived attributes

(i) Stored attributes

The simple attributes stored in database are called as **stored attributes.**

Example : 'Date_of_joining' for Employee is a stored attribute.

(ii) Derived attributes

The value of this attributes which can be derived from the value of related stored attribute is called as derived attributes.

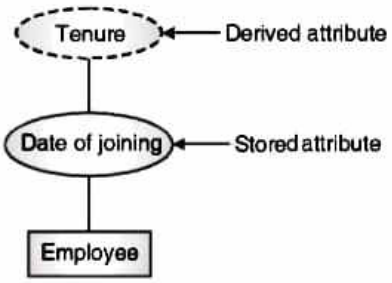


Fig. 4.3.3 : Derived attributes

Example : Employee tenure can be calculated from stored attribute 'Date_of_joining' of employee by subtracting it from today's date.

(d) Null attribute

- This attribute can take NULL value when entity does not have value for it.
- This is a special attribute the value of which is unknown, unassigned, not applicable or missing.

Example :

- The 'Net_Banking_Active_Bin' attribute gives whether particular customer having net banking facility is activated or not activated.
- For bank which does not offer facility of net banking in customer table 'Net_Banking_Active_Bin' attribute is always null till Net banking facility is not activated as this attribute indicates Bank offers net banking facility or does not offers.
- These attribute can be used in future or for unknown, unsigned, missing values of attribute.

(e) Key attributes

This is an attribute of an entity which must have a unique value by which any row can be identified is called as key attribute of entity.

Example : Emp_Id for employee.

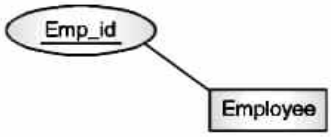


Fig. 4.3.4 : Key attributes

Such key attributes are used to identify a row in the table uniquely.

Type	Notation
Attribute (Simple/Single valued/Stored)	
Key attribute	
Multi valued attribute	
Composite attribute	
Derived attribute	

Fig. 4.3.5 : ER Notation for various types of attributes

4.4 Relationships

Q. What is relationship set? Give various constraints of relationship.

(1) Introduction

- A relationship is an association among one or more than one entity.
- We use the diamond symbol to show relationships.
- It has to be read from left to right or up to down.
- **Example:** Employee works for Department.

Fig. 4.4.1 : ER Diagram

(2) Degree

The degree of relationship is number of participating entity types in a particular relation.

(3) Relationship set

Collection of all relationship of same type is called relationship set.

4.4.1 Constraints on Relationship

(A) Mapping Constraints / Cardinalities

- Number of entities from each side participating in a relationship set.
- Cardinality expresses specific number of entity occurrence of related entity.

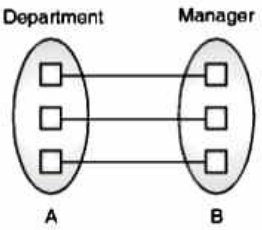
Types of mapping constraints

(i) One to one

- In this type of constraint one tuple in entity is related with only one tuple in other entity.
- That is one row in table is related with only one row in other table.
- A associated with at most one entity in B.
- B associated with at most one entity in A.

Example :

- One department can have only one manager.
- Every row in Department table can be having relationship with only one row in Managers table.



(a) One to one mapping

(b) Representation in ER diagram

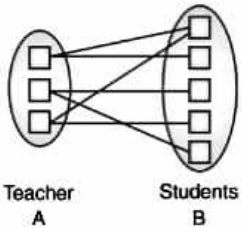
Fig. 4.4.2 : One to one mapping

(ii) One to many

- In this type of constraint one tuple in entity can be related with many tuples in other entity.
- A can be associated with any number of entities in B.
- B can be associated with at most one entity in A.

Example :

- One teacher may teach to many students.
- Every row in Teacher table can have relationship with many rows in Student table.



(a) One to many mapping

(b) Representation in ER diagram

Fig. 4.4.3 : One to many mapping

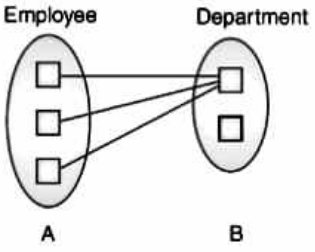
(iii) Many to one

- In this type of constraint many tuple in entity can be related with only one tuple in other entity.
- A can be associated with at most one entity in B.

- B can be associated with any number of entities entity in A.

Example :

- Number of employee works for department.
- Multiple rows in Employees table can be related with only one row in Department table.



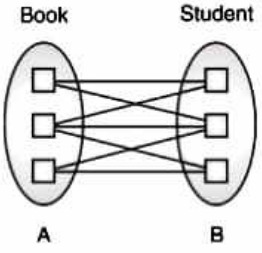
(a) Many to one mapping

(b) Representation in ER diagram

Fig. 4.4.4

(iv) Many to many

- In this type of constraint many tuple in entity can be related with multiple tuples in other entity.
- A can be associated with any number of entities in entity B.
- B can be associated with any number of entities entity in A.



(a) Many to many mapping

(b) Representation in ER diagram

Fig. 4.4.5

Example :

- Books in library issued by students.
- Multiple rows in Book table can be related with many rows in Student table.

(B) Participation constraints

(i) Total participation

- In case of total participation every object in an entity must participate in a relationship.
- The total participation is indicated by a dark line or double line between entity and relationship.

Example : Every department must have a manager.

Fig. 4.4.6 : Total participation

(ii) Partial participation

- In case of partial participation more than one object in an entity may participate in a relationship.
- The total participation is indicated by a single line between entity and relationship.

Example : Employees works for department.

Fig. 4.4.7 : Partial participation

(C) Degree of Relationship

(a) The degree of the relationship type is number of participating entity types.

(b) **Types** (Binary vs Ternary Relationship)

- **Binary relationship type :** A relationship of degree two.
- **Ternary relationship type :** A relationship of degree three.

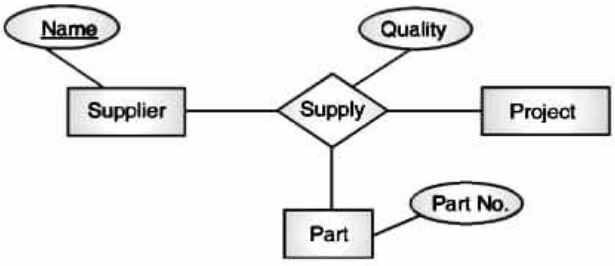


Fig. 4.4.8 : Ternary relationship

4.5 Sample ER Diagrams

Ex. 4.5.1 : A publication may be a **book** or an **article**. Articles are published in **Journals**. **Publication** has title and location. Book having their title and category. Article includes title, Topic and date. Publication is written by **Authors** stores Name, address and mobile number. Publication also belongs to particular **subject** which has their names.

Soln. :

Step 1 : **Identify entities**

1. Publication
2. Book
3. Article
4. Journal
5. Subject
6. Author

Step 2 : **Identify attributes**

1. **Publication (Title, Location)**

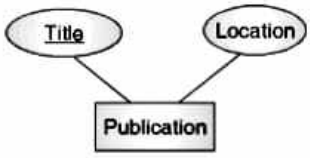


Fig. P. 4.5.1 : Publication entity

2. **Book (Publisher_id, Title, Category)**

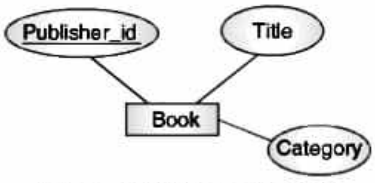


Fig. P. 4.5.1(a) : Book entity

3. **Article (Publisher_id, Title, Date, Topic)**

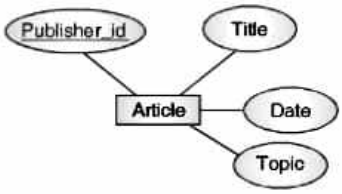


Fig. P. 4.5.1(b) : Article entity

4. **Journal (JID)**

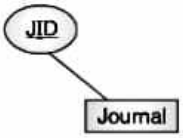


Fig. P. 4.5.1(c) : Journal entity

5. **Subject (SID, Name)**

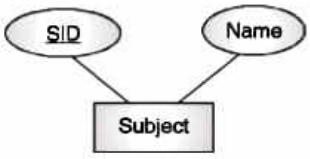


Fig. P. 4.5.1(d) : Subject entity

6. **Author (Auth_id, Name, Address, Mobile)**

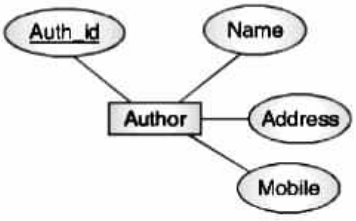


Fig. P. 4.5.1(e) : Author entity

Step 3 : **Identify relationships**

1. **Articles are published in Journal.**

Fig. P. 4.5.1(f) : Relationship between Articles and Journal

2. **Publication is written by Author.**

Fig. P. 4.5.1(g) : Relationship between Publication and Author

3. Publication belongs to a particular subject.

Fig. P. 4.5.1(h) : Relationship between Publication and Subject

Step 4 : **Identify inheritance relations**

Publication can be

BOOK or ARTICLE.

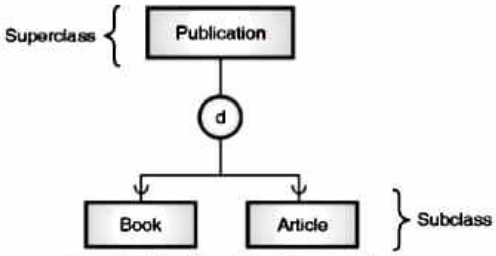


Fig. P. 4.5.1(i) : Inheritance relation

Step 5 : **Merging all above relations we will get final ER model**

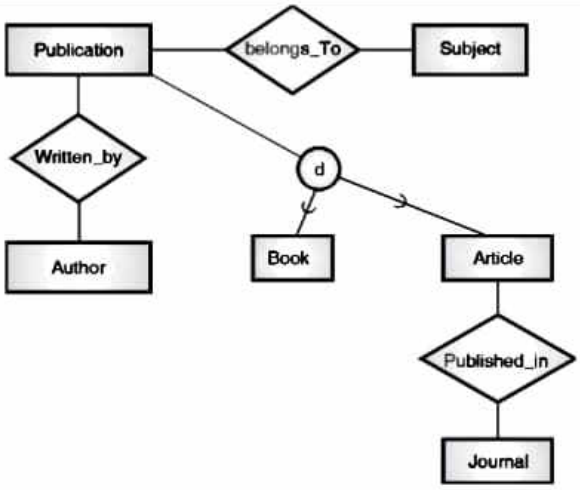


Fig. P. 4.5.1(j) : ER model for publication

Ex. 4.5.2 : Draw ER diagram for car insurance company that has a set of customer each of whose having one or more car. Each car associated with it zero to any number of record accidents.

Soln. :

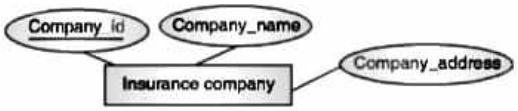
(1) Identify all entities

(a) Insurance company (b) Customer

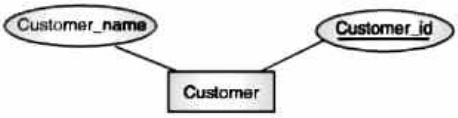
(c) Car (d) Accidents

(2) Identify all attributes

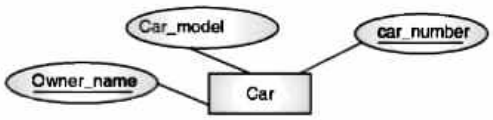
(a) Company entity



(b) Customer entity



(c) Car entity



(d) Accidents

Fig. P. 4.5.2

(3) Identify all relationship

a) Car insurance company has a set of customers

b) Customer owns one or more car

c) Each car associated with zero or any number of accidents

Fig. P. 4.5.2(a)

(4) Construct ER diagram by merging all above relationships

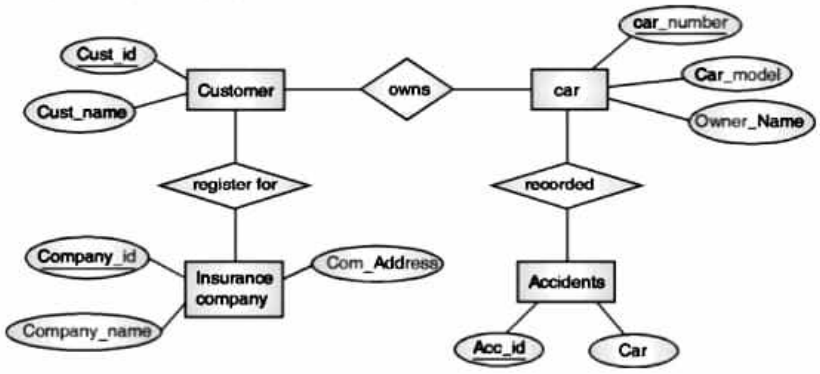


Fig. P. 4.5.2(b) : ER model for insurance company

Ex. 4.5.3 : Construct an ER diagram for a hospital with a set of patients and the set of medical doctors associated with each patient a record of various text and examination conducted.

Soln. :

(1) Identify Entities

(i) Hospital

(ii) Patient

(iii) Doctors

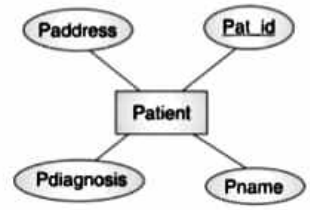
(iv) Medical-record (Record of various test and examination conducted)

(2) Identify Attributes

(i) Hospital (Hosp_id, HName, HAddress, Hcity)

(a) Hospital entity

(ii) Patient (Pat_id, Pname, Pdiagnosis, Padress)



(b) Patient entity

Fig. P. 4.5.3

(iii) Doctor (Doc_id, DName, Qualification, salary)

Fig. P. 4.5.3(c) : Doctor entity

(iv) Medical_Record (Record_id, Date_of examination, Problem)

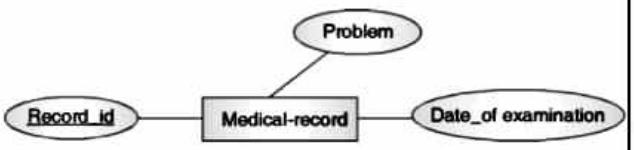


Fig. P. 4.5.3(d) : Medical entity

(3) Identify relationships

(a) Hospital has a set of patients

Fig. P. 4.5.3(e)

(b) Hospital has a set of doctors

Fig. P. 4.5.3(f)

(c) Doctors are associated with each patient

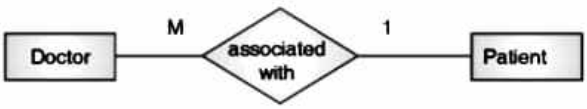


Fig. P. 4.5.3(g)

(d) Each patient has record of various test and examination conducted

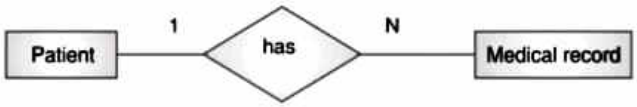


Fig. P. 4.5.3(h)

(4) Construct ER Model from merging all above relationship

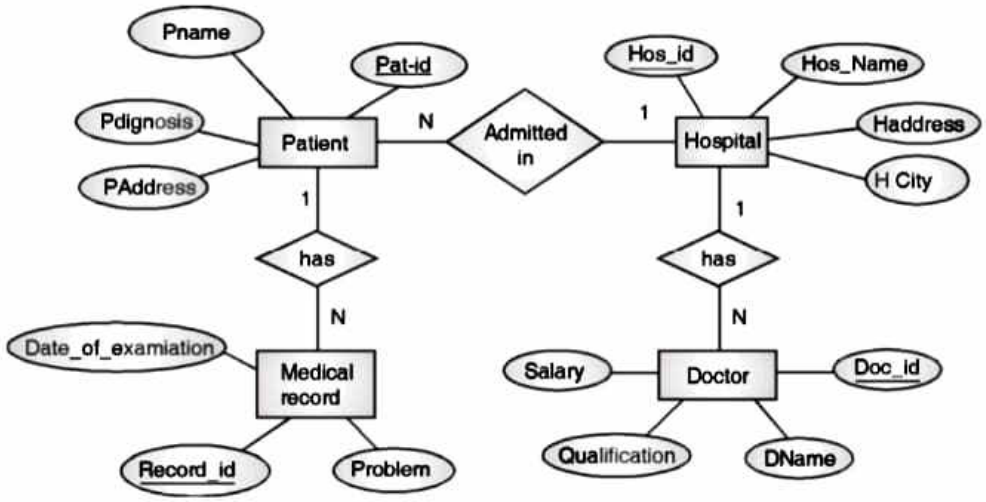


Fig. P. 4.5.3(i)

4.6 Entity Set In Terms of Tables

The mapping of entity relationship diagram to relational model is done using following steps and using following example.

4.6.1 Entity to Tables

Q. Explain how to convert entity into relational table.

(1) Regular entity types

- **Tables :** Regular entity sets can be represented as table in relational model.
- **Columns :** Attributes of entity set can be converted to the columns (attributes) of the tables in relational model.

Example : Regular entity employee mapped as employee table in object model like 'Stud_id', 'Stud_Addr' etc. are shown as table columns.

ER model

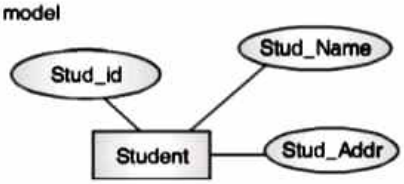


Fig. 4.6.1: Regular entity

Student table

Stud_id	Stud_Name	Stud_Addr
1	Snehal	Mumbai
2	Pratiksha	Mumbai
3	Supriya	Mumbai
4	Tanmay	Goa

(2) Weak entity types

For each weak entity type with owner entity, create a table and include all simple attributes of weak entity type as columns of table, including foreign key attributes as the primary key attribute of the table that correspond to the owner entity type.

Example : Dependents (Weak entity) in Employee (Owner entity).

ER Model

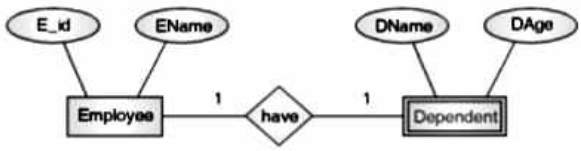


Fig. 4.6.2 : Weak entity

Employee table

E_id	Ename	DName	DAge
1	Sachin	Jyoti	23
2	Suhas	Manju	22
3	Jayendra	Tanya	27

4.6.2 Attributes to Columns of Table

Q. Explain how to convert attribute in ER to relational Table.

(a) Simple attributes

Simple attribute can be directly converted to a column (Attribute) in relational model.

Example :

Employee 'Age' can be directly converted to column.

Fig. 4.6.3 : Simple attribute

Employee table

Eid	Age
1	23
2	24
3	43
4	28

(b) Composite attributes

These attributes need to be stored as set of simple component attributes (Columns) in relational model by avoiding actual attribute ('name' in below example).

Example : In composite attribute 'Name' is converted to three columns in object model by avoiding actual attribute 'Name'.

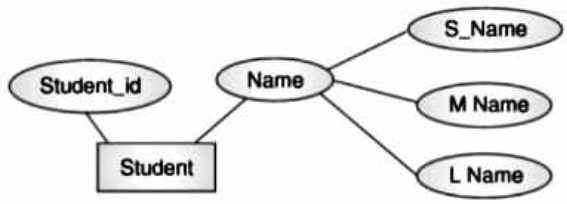


Fig. 4.6.4 : Composite attribute

Student table

Student_id	S_Name	MName	LName
1	Harshad	Rupali	Malar
2	Bipin	Anand	Shinde
3	Aanand	Ganesh	Panchal
4	Tushar	Bipin	Pimple

(c) Multi valued attributes

Multi valued attributes are mapped as a relation which includes combination of the primary key of table and multi valued attribute as a composite primary key as shown in Fig. 4.6.5

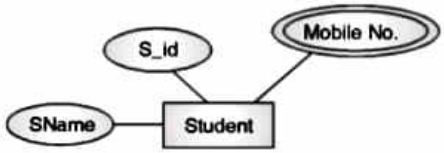


Fig. 4.6.5: Multi valued attribute

Student table

Student table

Sid	SName
1	Jayendra
2	Suhas
3	Sachin

Mobile Table

Sid	Mobile No.
1	9891295492
1	9821959241
2	808091856
3	8095124890
3	9773112456

Mobile table

Sid	Mobile No.
1	9891295492
1	9821959241
2	8080918456
3	8095124890
3	9773112456

(d) Derived attributes

There is no need to store such attribute in relational model. It will be calculated from stored attribute.

(e) Key attributes

Key attribute in ER Model can be directly converted to primary key attribute of relational model.

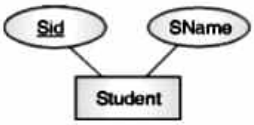


Fig. 4.6.6 : Key attributes

Student table

Sid	SName
1	Deepak
2	Vaibhav
3	Yogita
4	Bency

4.7 Relationships to Tables

Q. Explain how to convert various type of relations in ER to relational Table.

(a) Foreign key approach

If binary relationship type does not possess many attributes then we can map such relation using foreign key.

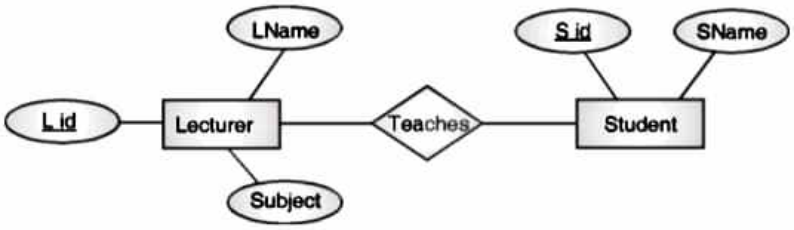


Fig. 4.7.1 : Foreign key approach

Lecturer table and student table

Lecturer table		
Lid	**LName**	**Subject**
1	Omprakash	IP
2	Yogesh	INS
3	Amit	PM

Student table		
Sid	**Sname**	**Lid**
1	Bency	1
2	Deepak	1
3	Yogita	1
4	Snehal	2
5	Pratiksha	2

Lid is foreign key in student table while primary key in lecturer table.

(b) Merged relationship approach

When there is total participation it is possible to merge relation and involved entities as a single relation and then map it to a table.

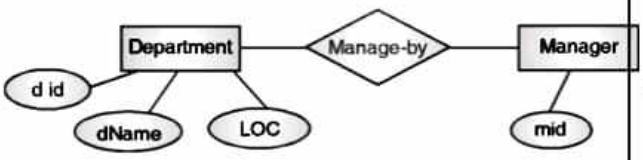


Fig. 4.7.2 : Merged relationship approach

Department table

Department table			
did	**Dname**	**Loc**	**mid**
10	IDF	Mahape	11
20	Mayban	Pune	22
30	Lax	Mumbai	33

(c) Cross reference approach

- A relationship type in EER is mapped to new table in relational model.
- Column of such table is all attributes of relation and primary key attributes of all tables linked to this relation.

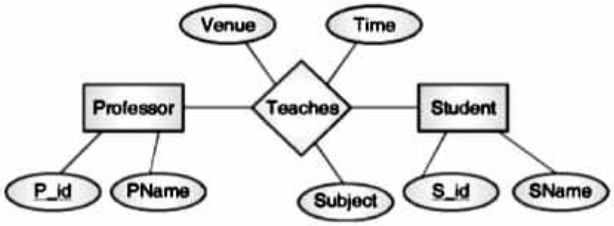


Fig. 4.7.3 : Cross reference approach

Professor table

Professor table	
Pid	**PName**
1	Om
2	Nitin

Student table

Student table	
Sid	**SName**
1	Snehal
2	Tanmay

Teacher relation table

Teacher relation table				
Pid	**Sid**	**Venue**	**Time**	**Subject**
1	1	SFIT	1 pm	ADBMS
1	2	XIE	1 pm	DBMS

4.8 Solved Examples

Ex. 4.8.1 : Draw an E-R diagram and reduce it to relational database model for a university database for scheduling of classrooms for final exams. This database could be modelled using entities as exam (course_name, section_number, room_number, time); course. (name, department, C_number), room (r_number, capacity, building). Entity section is dependent on course.

Soln. :

Step 1 : ER diagram

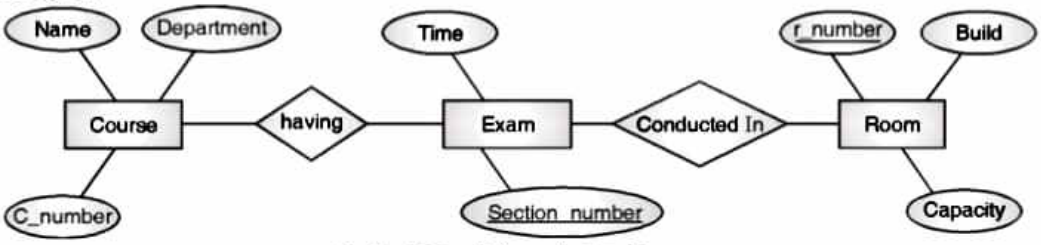


Fig. P. 4.8.1 : University ER diagram

Step 2 : Reducing to Tables

Converting entity to table and attribute to columns

Course (C_number, name, department)

Exam (Section_number, time, C_number,)

Room (R_number, building, capacity, Section_number)

Ex. 4.8.2 : Draw an E-R diagram for a university database consisting of 4 entities,

(i) Student (ii) Department (iii) Class (iv) Faculty and convert it to tables.

A student has a unique id, the student can enroll for multiple classes and has at most one major. Faculty must belong to department and faculty can take multiple classes-Every student will get a grade for the class he/she has enrolled.

Soln. :

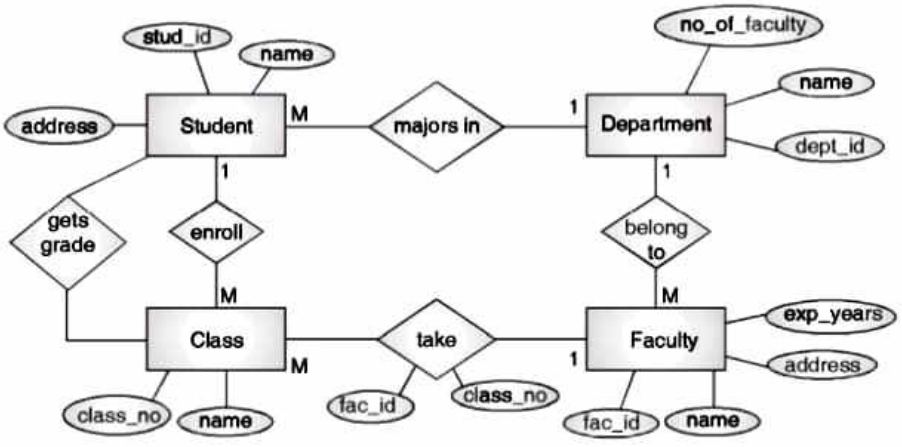


Fig. P. 4.8.2

Table P. 4.8.2

Student	
id	Primary Key
name	
address	
dept_id	Foreign key references to dept_id column of Department table
Faculty	
fac_id	Primary Key
fac_name	
exp_years	
address	
dept_id	Foreign key references to dept_id column of Department table
Class	
class_ no	Primary Key
c_name	
Stud_class	
class_ no	Foreign key references to dept_id column of Department table
stud_ id	Foreign key references to dept_id column of Department table
take_class	
fac_id	Foreign key references to fac_id column of Faculty table
class_ no	Foreign key references to class_ no column of Class table
Department	
dept_id	Primary Key
d_name	
no._of faculty	
Grade	
stud_ id	Foreign key references to dept_id column of Department table
class_ no	Foreign key references to dept_id column of Department table
Grade	

Ex. 4.8.3 : Draw an E-R diagram of library management system considering issue and return, fine calculation facility, also show primary key, weak entity and strong entity. W-18, 4 Marks

Soln. :

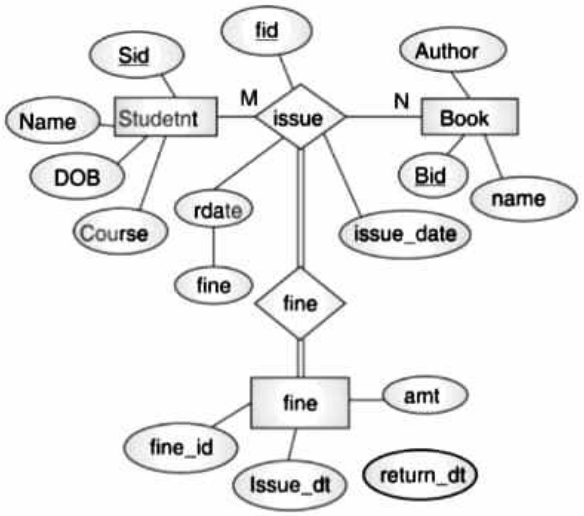


Fig. P. 4.8.3

Review Questions

Q. 1 Explain ER Diagrams and its components.

Q. 2 What is a ER diagram ?

Q. 3 Explain the components of ER diagram.

Q. 4 Write short note on : weak entity set, strong entity set

Q. 5 Define Entity . Explain weak and strong entity.

Q. 6 What are the types of attributes.

Q. 7 Explain constraints on relationship .

Q. 8 Explain how to convert attribute in ER to relational Table.

❑❑❑

Relational Data Model

Syllabus

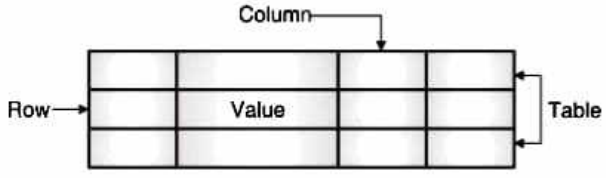
Fundamentals of RDBMS - Record, fields, Data types, tables and database

Concept of RDBMS, E.F Codd's rule for RDBMS, Key concepts – Candidate key, Primary key, Foreign key.

5.1 Relational Data Model

1. Introduction

- The relational model first proposed by E. F. Codd hence he is known as father of relational databases.
- Relational database was an attempt to simplify database structure by making use of tables and columns.
- Tables are known as **"relations"**, columns are known as **"attributes"** and rows (or records) are known as **"tuples"**.
- Relational Data Model Notations



- **Sample Relational Schema**

Student table

Sid	Name	Class	Major

Course table

Cid	Name	Hours

Department table

Did	Name

Marks table

Sid	Cid	Marks	Grade

Fig. 5.1.1 : Student Database

2. Relation / Relational Table

- Relations are a logical structure which is a collection of tables consisting horizontal rows also called as tuples and vertical columns also called as Attributes.
- This concept doesn't represent how the data is stored in the physical memory of computer system.
- Each table in a database has its unique table name.
- **Characteristics of Relation**
  - A table composed of rows and columns.
  - Each table row (tuple) represents a single entity occurrence within the entity set.
  - Each table column represents an attribute and each column has a distinct name called as **attribute** name.
  - Intersection of row and column represents a single data value also called as **Domain**.
  - All values in a same column must conform to the same format of data.
  - Each table must have a single attribute or set of attributes that uniquely identifies each row.

Example

In college database there is student table that contains all information about students.

Student table

Id	Name	Class	Branch	Age	Address	Mobile	Phone

(c) Attributes / Fields

- A column in above table represents data item stored in it, such column in database is called as attribute of a table.
- Every table must have at least one column in it.
- It is not possible to have multiple columns with same column name while, it is possible to have two columns with same column name but in two different tables.
- The SQL standard does not specify any maximum number of columns in a table.

Example : Id, Name, class are attributes of student table.

(d) Tuple / Records

- A single row in relational table which contains all the information about a single entity.
- Each horizontal row of the student table represents a Student tuple.
- A table can have any number of rows in it.

(e) Domain

- Every column in a table has a set of data values that are allowed for that column which is called as **Domain**.
- A domain with possible values should be associated with every attribute.
- In a relational table a domain can have a single value or no (Null) value.
- Domains are used to check that values inserted in database are correct or not and ensures that the comparisons made are correct.

5.2 E. F. Codd's Rule for RDBMS

Dr. E.F. Codd was the creator of the relational data model. This model was published as a two-part article in Computer World (Codd, 1985). It contains a list of 12 rules that determine whether a DBMS is relational and to what extent it is relational. These rules are a very useful measure for evaluating a relational system.

In the article, Codd mentions that according to these rules, there is no database yet that is fully a relational system. He says that rules 6, 9, 10, 11 and 12 are difficult to satisfy. Each of the twelve rules is discussed below. In Codd 1990, Codd extended the 12 rules to 18 to include rules on catalog, data types (domains), authorization etc.

5.2.1 Overview of Codd's Rule

(MSBTE - W-14)

Q. List first four codd rules. **W-14, 4 Marks**

Table 5.2.1

Summary of Codd's 12 Rules		
Sr.No.	**Rule**	**Description**
1.	Information Rule	All available data in system should be represented as a relations or tables. Information needs to be stored as values in tables.
2.	Guaranteed access rule	Each data item must be accessible without ambiguity by providing table name and its primary key of the row also include its column name to be accessed.
3.	Systematic treatment of null values	Null values are not equal to blank space or zero, they are unknown, unassigned values which should be treated properly.
4.	Self-describing database	There should be dynamic online catalog based dictionary on relational model which keep information about tables, data in database.
5.	Comprehensive data sublanguage	The data access language (SQL) must be the only means of accessing data stored in the database.
6.	View updating rule	All views of data are theoretically updatable can be updated using system also.
7.	High level insert, update and delete	This rule states that in a relational database, the query language must be capable of performing

Summary of Codd's 12 Rules		
Sr.No.	**Rule**	**Description**
		manipulations (such as, inserting, updating or deleting data) on sets of rows in a table.
8.	Physical data independence	Any changes made in the way data is physically stored (that is, data stored in file systems) must not affect applications that access data.
9.	Logical data independence	This rule states that changes to the database design should be done in a way without the users being aware of it.
10.	Integrity independence	Data integrity constraints which are definable in the query language must be stored in the database as data in tables that is, in the catalog and not in the application programs.
11.	Distribution independence	In a RDBMS, data can be stored centrally i.e. on a single system or distributed across multiple systems.
12.	Non subversion rule	This rule states that there should be no other access path to the database other than SQL

1. Information rule

a. This rule states that all available data in system should be represented as a relations or tables.

b. Data can be stored and viewed only from tables and not in any other way.

c. **Example :** If you wants to store data of faculties it can be stored as shown in Table 5.2.2.

Table 5.2.2

Faculty Table			
Faculty_code	**Faculty_Name**	**DOB**	**Subject**
100	Yogesh	17/07/64	DSA
101	Amit	24/12/72	MIS
102	Omprakash	03/02/80	PWRC
103	Nitin	28/11/66	DT
104	Mahesh	01/01/86	DT
105	Ashok	28/11/66	DSA

2. Guaranteed access rule

a. Data item must be accessible by providing table name and its primary key column name of the row to be accessed.

b. For accessing data we should make use of Table name, Primary key (column name) and other column names to be accessed.

c. All data items are uniquely identified and accessible via this identity (primary key).

d. Example : If you want to access data of faculty names you must provide

Table name -> Primary key -> Column_name

Faculty -> Faculty_Code ->Faculty_Name

3. Systematic treatment of null values

a. Null values are values which are missing, unknown or unassigned values.

b. We need to handle such data in a consistent manner.

c. Nulls should have no values and should simply be a missing data.

d. Treating Nulls as zero for missing numeric data or as a blank for missing character data is not allowed by this rule.

e. If required vendors or designer may use default values for missing data.

4. Self-describing database

a. For storing user data, a relational database must contain data about itself.

b. There are two types of tables in RDBMS :

(i) **User tables :** Contain data about tables created by any users of system

(ii) **System tables :** Contain data about the database structure and database objects.

c. **Metadata :** The data which describes the database structure is called meta-data. In short

data about data is metadata.

The collection of system tables is called as the system catalogues or data dictionary.

Storing any part of the data dictionary in operating system files would violate this rule. That is database only should store metadata.

5. Comprehensive data sublanguage

a. The data access language (SQL) must be the only means of accessing data stored in the database and it has capacity to perform all required operations.

b. The language must support relational operations and set operations with regard to the following :

(i) Data Definition

(ii) Data Manipulation

(iii) Integrity Definition

(iv) Transaction control

(v) Data control

c. Accessing data files (files that contain the actual data), through any utility other than an SQL interface, violates this rule.

6. View updating rule

a. This rule states that views should allow updates in the underlying.

b. All views are theoretically updatable are updatable by system also.

c But SQL supports only updates of single tables at a time therefore, if you join three tables to create a complex view and try to update that view and then DBMS would fail to translate these updates to the underlying tables, thereby violating this rule.

d. Also, a view not including the column that uniquely identifies (like primary key) each record in a table cannot be updated, thus violating the rule.

Example : If Employee_ID column is not present in view then it is not possible to update such views.

Note

- Views are data object like virtual tables or window of a table which can show you the data that is present in any table.
- Views are unlike a data object table as it contains no data, just statements that returns a table.
- There are situations where users may require only a part of the table information or information from a collection of tables. Such requirements can be met by creating a 'Views'. Sports teacher may require only sort related data this requirement can be satisfied by creating a view.

7. High level insert, update and delete

a. This rule states that in a relational database, the query language must be capable of performing manipulations (such as, inserting, updating or deleting data) on sets of rows in a table.

b. This rule states that multiple row inserts (inserting many rows at once) or bulk insert operations (inserting data based on external files or from database tables) should be allowed in relational databases.

b. A database that supports only single row manipulation at a time cannot be considered as relational.

8. Physical data independence

a. This rule states that in a RDBMS, any changes made in the way data is physically stored (that is, data stored in file systems) must not affect applications that access data.

b. This rule explains the changes done in back end (oracle/SQL Server) must not affect Front end application (Java).

c. If a file supporting a table was moved from one disk to another, or renamed, then this should have no impact on the application.

9. Logical data independence

a. This rule states that changes to the database design should be done in a way without the users being aware of it.

b. The change could be to expand the database (adding a new table) or to reduce it, but the application that refers to the data (logical) must work as before.

c. If a single table were split into two, then a view would have to be provided joining the two tables back together so that there would be no impact on the application.

10. Integrity independence

a. In order to be considered as a relational database, all data integrity, which are definable in the query language referred to in rule 5 must be stored in the database as data in tables, that is, in the catalog and not in the application programs.

b. The data integrity rules that should apply to relational databases are :

(i) **Entity Integrity :** The primary key column cannot have missing values or duplicate value.

(ii) **Referential Integrity** : For every foreign key column value, there must exist a matching primary key column value.

11. Distribution independence

a. In a RDBMS, data can be stored centrally (on a single system) or distributed (across multiple systems).

b. The data in a centralized database should remain logically unaffected if database is distributed across multiple systems.

c. For example, a user should be able to retrieve data from two tables distributed across two terminals, the same way, as they would retrieve them if stored in the single terminal.

12. Non subversion rule

a. This rule states that there should be no other access path to the database, other than SQL.

b. Any other access language may not bypass (or subvert) security or integrity rules, which otherwise would be obeyed by the regular data access language.

5.3 Key Concept

- The column value that uniquely identifies a single record in a table called as Key of table.
- Any key consisting of a single attribute is called a **simple key,** while that consisting of a combination of attributes is called a **composite key**.

5.3.1 Types of Keys

(MSBTE - W-13, S-14, W-14, S-15, W-15, S-17, W-17, W-18)

Q. Describe primary key and candidate key. **W-13, S-14, W-15, S-17, 2 Marks**

Q. Describe following keys :
(i) Primary key
(ii) Foreign key. **W-14, S-15, W-17, 4 Marks**

Q. Define the terms :
(i) Candidate key
(ii) Primary key **W-18, 2 Marks**

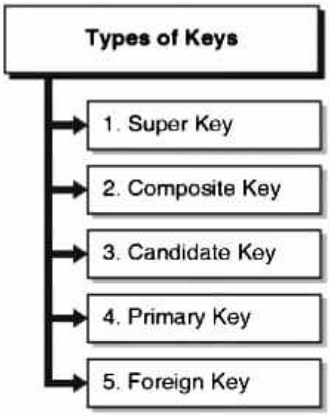


Fig. 5.3.1

(i) Super Key

- An attribute or set of attributes that uniquely identifies a single tuple in entity.
- There can be more than one super keys in single table.

Example

ID is a key of student table. It is possible to have only one student with a one ID (Say only one student **'Mahesh'** with ID = 1)

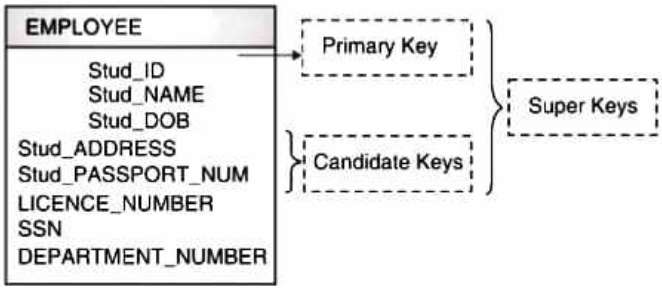


Fig. 5.3.2 : Types of Keys

(ii) Composite Key

Any key with more than one attributes that uniquely identifies a single tuple in entity.

Example

In Fig. 5.3.2, a super key has more than one attribute so, it is a composite key.

(iii) Candidate Key

- A super key with minimum number of attributes is a candidate's key.
- Superkey concept given above may contain unnecessary attributes to key so the concept of a superkey is not sufficient.
- A Candidate Key (CK) is a superkey with the minimal attribute from super key.
- A minimal (Irreducible) superkey is called as candidate's key.
- A superkey that does not contain a subset of attributes that is itself a superkey.
- Minimum attributes of superkey by omitting unnecessary attributes of table which are sufficient for identifying entity (row/record) uniquely are called as **candidate keys**.
- Candidate key is also a potential primary key.

Example

In Fig. 5.3.2, combination of (Stud_Passport_Num, Licence_Number,ssn) acts like a Candidates key.

(iv) Primary Key

- A primary key is a column or group of columns in a table that uniquely identify tuple or record (row) in relational table.
- Primary key cannot be a null value and it must be unique for each tuple in relational table.
- In Employee table Emp_Id is unique and not null column, also referred as Primary Key.

Table 5.3.1

Emp Table		
Emp_Id	**Emp_name**	**Did**
1	Sachin	20
2	Suhas	10
3	Jay	20
4	Om	10

(v) Foreign Key

- A foreign key is a column or group of columns in a table that provides a connection between data of two tables.
- Foreign key acts as a cross-reference between two tables because it refers to the primary key of another table, so it establishes a link between tables.
- The referential integrity constraint is specified between two tables to maintain the consistency among tuples in the two tables.
- The tuple in one relation refers only to an existing tuple in another relation.

Example

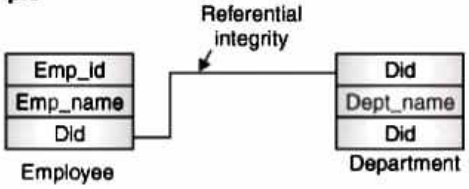


Fig. 5.3.3 : Did as Foreign Key

Table 5.3.2

Emp Table		
Emp_Id	**Emp_name**	**Did**
1	Sachin	20
2	Suhas	10
3	Jay	20
4	Om	10

Department Table	
Did	**Dept_name**
10	HR
20	TIS
30	L&D

- In the above example 'Emp' table has 'Did' as foreign key reference this is called as referential integrity.

Table 5.3.3

Key Type	Definition
Super Key	An attribute or set of attributes that uniquely identifies a single tuple in entity.
Composite Key	Any key with more than one attributes that uniquely identifies a single tuple in entity.
Candidate Key	A super key with minimum number of attributes is a candidate's key. No subset of candidate key van be key.
Primary key	A selected key of strong entity which uniquely identify tuple in entity is a primary key of that entity.
Alternate Key	A Candidate key which is not selected as primary key
Secondary Key	An attribute or set of attributes that used to access a single tuple in entity.

Review Questions

Q. 1 Explain relational database model.

Q. 2 Explain terms primary key and foreign key with example.

Q. 3 Explain column check constraint in SQL.

Q. 4 Explain the reference integrity with example?

Q. 5 Write short note on: Integrity Constraint.

Q. 6 Explain codd's Rule .

Q. 7 What is primary key and foreign key .

Q. 8 Explain various types of keys.

❑❑❑

Normalization

Syllabus

Normalization : Normalization Concepts, Need of Normalization, Types of Normalization – 1NF, 2NF,3NF

6.1 Normalization Process

(MSBTE – W-13, S-14, W-15, S-16, W-17, W-18)

Q. Define normalization.

W-13, S-16, W-17, W-18, 2 Marks

Q. Explain normalization with example.

S-14, 4 Marks

Q. What is meant by database normalization ?

W-15, 2 Marks

(1) Introduction

- Normalization is a step by step decomposition of complex records into simple records.
- Normalization is a process of organizing data in database in more efficient form.
- It results in tables that satisfy some constraints and are represented in a simple manner.
- This process is also called as canonical synthesis.

(2) Definition

Normalization is a process of designing a consistent database by minimizing redundancy and ensuring data integrity through decomposition which is lossless.

6.1.1 Needs of Normalization

(MSBTE -W-13, W-14, S-16, W-17)

Q. Explain two goals of Normalization.

W-13, S-16, W-17, 2 Marks

Q. Explain need of normalization. W-14, 4 Marks

Q. What are the three data anomalies that are likely to occur as a result of data redundancy ?

Q. Can data redundancy be completely eliminated in database approach ? Why or why not ?

1. Ensures Data Integrity

- Data integrity ensures the correctness of data stored within the database.
- It is achieved by imposing integrity constraints.
- An integrity constraint is a rule, which restricts values present in the database.

2. Prevents redundancy in data

- A non-normalized database is more vulnerable to various problems, if it stores data redundantly.
- If data is stored in two locations, but the data is updated in only one of the locations, then that data becomes inconsistent; this is referred to as an "update anomaly".
- A normalized database stores non-primary key data in only one location.

3. To avoid data anomaly

- A non-normalized table can have some inconsistencies which may cause data anomalies.
- A relational database table should avoid all data anomalies.
- The normal forms of relational database decide, whether relational database design is vulnerable to data anomalies.

(a) Update Anomaly

Same information can be present in multiple records of various relations; updates to only one table may result in logical inconsistencies.

Example

- Each record in an "Emp_Salary" table might contain an Emp_ID, Ename, Address, Salary. Thus, a change of address for a particular Employee will potentially need to be applied to multiple tables such as Employee table.
- If all the records are not updated then some tables may leave in an inconsistent state.

(b) Insertion Anomaly

There is a possibility in which certain facts cannot be recorded at all or they are not yet recorded.

Example

- Consider a table, Faculty (Faculty_ID, FName, Subject_Code, Subject, Class).
- We can add the details of any faculty member who teaches for a certain subject in a certain class, but we cannot record the details of a new faculty member who has not yet been assigned to teach any subject or class. So, subject and class column may be empty initially.
- If data is deleted from one table all relevant data must also be deleted or redundant.

(c) Deletion Anomaly

- If data deleted from one table then all relevant data in another related tables must also be deleted, otherwise it will create redundancy problem.
- Deletion of some data from a relation necessitates the deletion of some unrelated data also called as deletion anomaly.

Example

- In the previous example, the table suffers from this type of anomaly. If a faculty member temporarily ceases to be assigned a subject, we must delete the entire record on which that faculty member appears.
- There are formal methods for quantifying "how normalized" a relational database is, and these classifications are called Normal Forms (or NF).

6.2 Functional Dependencies

Q. Explain concept of functional dependency.

(1) Introduction

- The concept of functional dependency is given by E. F. Codd which is also called as normalization process.
- We can say column X is functionally dependent on another column Y. If data value in column X change when data value in another column Y is modified.

Example : let us say, in Employee table when name of employee is changed its ID also need to be changed so, we can say NAME column is depend on employee ID column. i.e. for different name of employee different ID is given.

Employee table

Employee Table		
ID	***Name***	**....**
Mah001	*Mahesh*	....

- Functional Dependency (FD) provides a formal mechanism to express constraints between various attributes of a relation.
- Functional dependency determines the set of values or the attribute based on another attribute.
- Functional data dependencies are restrictions imposed on the data in database.

(2) Goals of Function Dependency

Q. Give various goals of functional dependency.

Avoid data redundancy (or minimize data redundancy)

- Same data should not be repeated at multiple locations in same database.

Enhancing data reliability

- To maintain a quality data in database.

(3) Explanation

- Let Given relation R, attribute Y of relation R is functionally dependent on the attribute X of relation R if and only if each value of X in R has associated with precisely one value of Y.
- In given relation R, attribute Y is functionally dependent on X if for each value of attribute Y there is exactly one value of X (Inverse of this is not always true).

Example

EMP (Emp_Id, Name, Salary)

- In above example, salary of employee is functionally dependent on his Emp_Id, because a particular Emp_ID corresponds to one and only one salary value.
- An attribute may be functionally dependent on single attribute or combination of attributes.

(4) Representation

It is denoted by (→)

$$X \rightarrow Y$$

This form can be written as Y is functionally dependent on X or X determines Y.

Example

Employee name is functionally dependent on Employee code.

E_code → E_Name

E_code	*E_Name*	*Address*
1	*Mahesh*	*Worli*
2	*Mukesh*	*Thane*
3	*Sachin*	*Andheri*
4	*Manish*	*Dombivli*

6.2.1 Types of Functional Dependencies

Q. What are types of functional dependencies?

Q. Define multi valued dependencies. Explain the fourth normal forms algorithm to remove it.

(1) Full functional dependency

Definition

A functional dependency A → B is a full functional dependency if removal of any attributes from A means that the dependency does not hold any more.

Example 1

{Emp_no, Project_no} → Hours i.e. Emp_no → Hours and Project_no → Hours

- In Example 1, Hours are fully functionally dependent on both Emp_no and Project_no.
- The number of hours spent on the project by a particular employee can not be determined with the project number (Project_no) alone. It needs the employee number (Emp_no) as well.

Example 2

If one of the attributes is not present (Null) then relations does not hold true.

Emp_no	Project_no	HOURS
1	A1	12
2	B1	4
3	C1	8
4	D1	12

(2) Partial functional dependency

A partial dependency means that a non key column is depend on some columns in composite primary key of a table.

Example 1

Let us say, in employee table ID and Name of employee together making its composite key. So ID column is partially dependent on name because when name of employee is changed ID also need to be changed.

Employee table

ID	Name	. . . .	DOB
Mah001	Mahesh	. . . .	01/01/1986

Definition

An FD A → B is a partial dependency if there is some attribute X ∈ A (X subset of A), that can be removed from A and the dependency will still hold.

Example 2

{Emp_no, Project_no} → Ename that is, Emp_no → Ename

- In Example 1, Ename is partially dependent on {Emp_no, Project_no}
- Reason being, employee name (ename) can be determined using the employee id (Emp_no) alone even if project_no is removed from the relation.

Note : For a table to be in 2nd Normal form there should be no partial dependencies.

Example 3

Emp_no	EmpName	ProjectNo	HOURS
1	Jayendra	A1	12
2	Yogesh	B1	24

If EmpName is null then also there will not be any effect on ProjectNo or Hours attribute of Employee.

(3) Transitive dependency (Trivial functional dependency)

- This concept is used when there is redundancy in database.
- If changing any non key column (column other then key column) causes change in other non key column in such situations you may have transitive dependency.

Definition

- When one non key attribute is functionally dependent on another non key attribute then such a dependency is called as **transitive or trivial dependency.**
  - Non key attribute → Non key attribute
  - An FD X →Y in a relation R is a transitive dependency, if there is a set of attributes Z that is not a subset of any key of R, and both X → Z and Z → Y holds true.

Example

EMP_DEPT {Eno, Ename, Dnumber, DMgrNo}
Eno → DmgrNo is transitive

Dependency of DmgrNo on key attribute Eno is transitive as DmgrNo is depends on Dnumber which is depend on Eno.

Eno → Dnumber and Dnumber → DmgrNo

Eno	Ename	Dnumber	DMgrNo
1	Mahesh	10	1
2	Mangesh	10	1

(4) Multivalued dependency

Definition

- **Multivalued dependency,** is defined as relationship which accepts the cross product pattern.
- Multivalued dependency defined by X →→ Y is said to hold for a relation R(X,Y,Z) if for a given set of values of X, there is a set of associated values of attribute Y and X values depend only on X values and have no dependence on the set of attributes Z.

Example

- Consider relation between Employee (EName, Address, Car) which can be given as :

  EName →→ Address

  EName →→ Car

- $\alpha \rightarrow\rightarrow \beta$ says relationship between α and β independent of relationship between α and R-β. That means EName →→ Address relationship is independent of EName and (EName, Car) relation. i.e. EName →→ Address is independent of EName →→ Car

EName	Address	Car
Yogesh	Jogeshwary	Maruti 800
Om	Panvel	Omni
Nitin	Vashi	Zen

- We will take one more relation

Product purchased	CustName	Street	City
Bread	Yogesh	AB Nort	Mumbai
Bread	Yogesh	H. C. Road	Goa
Eggs	Nitin	SVP Road	Delhi

- Relation between CustName and Product_purchased is independent of relationship between customer and his address.
- If Yogesh has purchased bread, we want all Yogesh's address associated with that purchase.

CustName → Street, City

6.3 Armstrong's Axioms - Closures of Functional Dependency

Q. Give armstrong axioms.

Q. List the Armstrong's axioms for functional dependencies. What do you understand by soundness and completeness of these axioms ?

- Given that A, B and C are sets of attributes in a relation R; one can derive several properties of functional dependencies.

- Axioms are nothing but rules of inference which provides a simple technique for reasoning about functional dependencies.

(1) Primary rules

a. Subset property (axiom of reflexivity)

If Y is a subset of X as shown in Fig. 6.3.1.

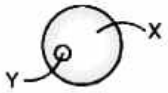


Fig. 6.3.1

Then, $X \rightarrow Y$

Means Y is functionally dependent on X

Or X functionally determines Y and also $X \rightarrow X$ (X functionally dependent on X)

b. Augmentation (axiom of augmentation)

If Relations R (X, Y, Z, W) having functional dependency $X \rightarrow Y$,

Then, $XZ \rightarrow YZ$ is True.

c. Transitivity (axiom of transitivity)

If Relations R (X, Y, Z, W) having functional dependency $X \rightarrow Y$ and $Y \rightarrow Z$, Then, $X \rightarrow Z$ is True.

(2) Secondary rules (based on primary rules)

a. Union :

If Relations R (X, Y, Z, W) having functional dependency $X \rightarrow Y$ and $X \rightarrow Z$, Then $X \rightarrow YZ$ is True.

b. Decomposition

If Relations R (X, Y, Z, W) having functional dependency $X \rightarrow YZ$ and $X \rightarrow Y$, Then $X \rightarrow Z$ is True.

c. Pseudo transitivity

If Relations R (X, Y, Z, W) having functional dependency $X \rightarrow Y$ and $YZ \rightarrow W$,

Then $XZ \rightarrow W$ is True.

(3) Canonical cover

A functional depending set X is canonical or minimal if the set has following properties

1. Each right set of a functional dependency of X contains only one attribute.
2. Each left set of a functional dependency of X is minimal. It means there should not be any extraneous attribute is present in dependency.
3. Reducing any functional dependency will change the content of X.

Note : Extraneous attributes are one which can be removed without changing the closure set of functional dependency.

Ex. 6.3.1 : Consider relation R = (A,B,C,D,E,F) having set of FD's

$A \rightarrow B$ $A \rightarrow C$

$BC \rightarrow D$ $B \rightarrow E$

$BC \rightarrow F$ $AC \rightarrow F$

Calculate some members of Axioms as be below :

(i) $A \rightarrow E$ (ii) $BC \rightarrow DF$

(iii) $AC \rightarrow D$ (iv) $AC \rightarrow DF$

Soln. :

(i) $A \rightarrow E$

As $A \rightarrow B$ and $B \rightarrow E$

So using Transitive rule,

$$\therefore A \rightarrow E$$

(ii) $BC \rightarrow DF$

As $BC \rightarrow D$ ...(i)

$BC \rightarrow F$ ...(ii)

$\therefore$ Using union rules (i) and (ii)

$$\therefore BC \rightarrow DF$$

(iii) $AC \rightarrow D$

$A \rightarrow B$ ...(iii)

As $BC \rightarrow D$ ...(iv)

$\therefore$ Using pseudo transitivity

$\therefore AC \rightarrow D$ ...(v)

(iv) $AC \rightarrow DF$

Using above rule (v)

$AC \rightarrow D$ ...(vi)

$AC \rightarrow F$ ...(vii)

$$\therefore AC \rightarrow DF$$

6.4 Decomposition

Q. What is decomposition ?

1. Introduction

- If a relation is not in the normal form and we wish the relation to be normalised so that some of the

anomalies can be eliminated, it is necessary to decompose the relation in two or more relations.

- The process of decomposition of a relation R into a set of relations R_1, R_2....R_n was based on identifying attributes and using that as a basis of decomposition.

$$R = R_1 \cup R_2 \cup \cdots \cup R_n$$

- This is a process of dividing one table into multiple tables using projection operator.

2. Goals

Q. What are goals of decomposition ?

(a) Decomposition may cause some problems of information loss as we are neglecting some columns by normalizing the relation.

(b) Lossless join decomposition

No information is deleted from a relation while fragmenting in order or reconstructs the original relation from fragmented relations.

(c) Dependency preservation

All functional dependencies result in just one relation.

(d) Boyce-Codd Normal Form (BCNF)

It causes no redundancy.

3. Example

Department_Student Schema = (Dept_ID, Dname, Stud_ID, Sname, Location)

From above relations we can create new Department table and Student by decomposing the Department_Student table,

From Department_Student schema we can create following relations :

Department = $\Pi_{\text{Dept\_ID, Dname, Location}}$ (Department_Student)

Student = $\Pi_{\text{Stud ID, Sname, Dept ID}}$ (Department_Student)

As Dept_ID column is common between above two tables so we can combine tables to regain Department_Student schema.

6.4.1 Desirable Properties of Decomposition

Q. Explain properties of decomposition in detail.

The main properties of decomposition are as follows :

1. Lossless-join decomposition
2. Dependency preservation
3. Lack of redundancy (Repetition of information)

(1) Lossless-join decomposition

- It is clear that decomposition must be lossless so that we do not lose any information from the relation that is decomposed.
- Lossless join decomposition ensures that we can never get the situation where spurious tuple are generated in relation, for every value on the join attributes there will be a unique tuple in one of the relations. For above join to become lossless we need to go for following steps.

Steps

(i) Let R_1 and R_2 form decomposition of relation R as R_1 and R_2 are both sets of attributes from R.

(ii) Decompose the relation schema Department_Student into

Department-schema = (Dept_Id, Dname)

Student-schema = (Stud_id, Sname, Location)

(iii) The attributes in common must be a key for one of the relation for decomposition to be lossless. $R_1 \cap R_2 \neq \Phi$ There must not be null value.

Note : You are joining a primary key and a foreign key of table.

Example

Above relation can be lossless decomposed as follows,

Schema 1 : Department schema contains (Dept_Id, Dname)

$\therefore$ $R_1 \leftarrow \Pi_{\text{Dept\_Id,Dname}}$ (Department_student)

Dept_Id	Dname
10	Development
20	Teaching
30	HR

Schema 2 : Student schema contains (Stud_Id, Dept_Id, Sname, Location)

$\therefore$ $R_2 \leftarrow \Pi_{\text{Stud\_Id, Dept\_Id, Location,Sname}}$ (Department_student)

Stud_Id	Dept_Id	Location	Sname
1	10	Mahim	Sushant
2	20	Vashi	Snehal
3	30	Worli	Pratiksha
4	20	Dadar	Supraja

This is lossless-join decomposition as $R_1 \cap R_2 \neq \Phi$ common column is Dept_Id

(2) Dependency preservation

- Dependency preservation is another important requirement since a dependency is a very important constraint on the database.
- As a result of any database updates, the database should not result in illegal relation being created. Hence, our design should allow us to check updates without natural joins.
- If $X \rightarrow Y$ holds then we know that the two (sets) attributes are closely related or functionally dependent and it would be useful if both attributes in the same relation so that the dependency can be checked easily.
- This can be done by maintaining functional dependency.
- Consider relation R(X, Y, Z, W) that has the following dependencies F,

$$X \rightarrow Y$$

$$X \rightarrow Z$$

If we decompose the above relation into R1 (X, Y) and R2 (Y, Z, W) the dependency $X \rightarrow Z$ is not preserved.

Example

Student Schema = (Stud_Id, Stud_Name, Dept_Id, Dname, Location, Subj_Id, SubjName)

Student-Department schema = Stud_Id $\rightarrow$ Dept_Id, Dname, Location

Student-Subject schema = Stud_Id $\rightarrow$ Subj_Id, SubjName

(3) Repetition of Information

- Decomposition that we have done should not suffer from any repetition of information problem.
- STUDENT and SECTION data are separated into distinct relations. Thus we do not have to repeat STUDENT data for each SECTION.
- If a single SECTION is made into several STUDENTS, we do not have to repeat the SECTION data for each STUDENT.
- IT is desirable not to have any redundancy in database.
- This property may be achieved by normalization process.

6.5 Types of Normalization

6.5.1 First Normal Form (1NF)

(MSBTE - W-14, W-17, W-18)

Q. State and explain 1NF with example. W-14, W-17, W-18, 2/4 Marks

Q. List normalization types. W-18, 1 Mark

Q. What is first normal form ? Explain with example.

(1) Introduction

- Simplest form of normalization, simplifies each attribute in relation.
- This normal form given by E.F. Codd (1970) and the later version by C.J. Date (2003).

(2) Definition

- A relation is in 1NF, if every row contains exactly one value for each attribute.
- 1NF states that attributes included in relation must have atomic (Simple, indivisible) values and all attribute in a tuple must have a single value from the domain of that attribute.
- In short rules for data in 1NF is,
- Table columns should contain atomic data.
- There should not be any repeating group of data.

(3) Example

- Let us understand this with an example.
- Consider a table **'Faculty'** which has information about the faculty, subjects and the number of hours allotted to each subject they teach in class.

Table 6.5.1 : Faculty

Faculty code	Faculty Name	Date of Birth	Subject	Hours
100	Yogesh	17/07/64	DSA	16
			SS	8
			IS	12
101	Amit	24/12/72	MIS	16
			PM	8
			IS	12
102	Omprakash	03/02/80	PWRC	8
			PCOM	8
			IP	16
103	Nitin	28/11/66	DT	10
			PCOM	8
			SS	8
104	Mahesh	01/01/86	DT	10
			ADBMS	8
			PWRC	8

- Table 6.5.1 does not have any atomic values in the 'Subject' column. Hence, it is called un-normalized table. Inserting, updating and deletion would be a problem is such table. Hence it has to be normalized.
- For the Table 6.5.1 to be in first normal form, each row should have atomic values. Hence let us reconstruct the data in the table. A **'Sr. No.'** column is included in the table to uniquely identity each row.

Table 6.5.2 : 1NF

Sr. No.	Faculty code	Faculty Name	Date of Birth	Subject	Hours
1.	100	Yogesh	17/07/64	DSA	16
2.	100	Yogesh	17/07/64	SS	8
3.	100	Yogesh	17/07/64	IS	12
4.	101	Amit	24/12/72	MIS	16
5.	101	Amit	24/12/72	PM	8
6.	101	Amit	24/12/72	IS	12
7.	102	Omprak ash	03/02/80	PWRC	8
8.	102	Omprak ash	03/02/80	PCOM	8
9.	102	Omprak ash	03/02/80	IP	16
10.	103	Nitin	28/11/66	DT	10
11.	103	Nitin	28/11/66	PCOM	8
12.	103	Nitin	28/11/66	SS	8
13.	104	Mahesh	01/01/86	DT	10
14.	104	Mahesh	01/01/86	ADBMS	8
15.	104	Mahesh	01/01/86	PWRC	8

This Table 6.5.2 shows the same data as the Table 6.5.1 but we have eliminated the repeating groups. Hence the table is now said to be in **First Normal Form (1NF)**.

6.5.2 Second Normal Form (2NF)

(MSBTE - W-14, W-17, W-18)

Q. State and explain 2NF with example. **W-14, W-17, W-18, 2/4 Marks**

(1) Introduction

- This normal form makes use of functional dependency and tries to remove problem of redundant data that was introduced by 1NF.
- Therefore, before applying 2NF to a relation, it needs to satisfy 1NF condition.
- This normal form is given by E.F. Codd (1971).

(2) Definition

- A relation is in 2NF, if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key of the relation.
- OR A relation is in 2NF, if it is in 1NF and every non-key attribute is fully functionally dependent on the whole and not just part of primary key of relation.
- In short 2NF means,
- It should be in 1NF.
- There should not be any partial dependency.
- 2NF prohibits partial dependencies.

(3) Steps

(a) Find and remove attributes that are related to only a part of the key or not related to key.

(b) Group the removed attributes in another table.

(c) Assign the new table a key that consists of that part of the old composite key.

(d) If a relation is not in 2NF, it can be further normalized into a number of 2NF relations.

(4) Example

Let us consider the table we obtained after first normalization.

Table 6.5.3

Sr. No.	Faculty code	Faculty name	Date of birth	Subject	Hours
1	100	Yogesh	17/07/64	DSA	16
2	100	Yogesh	17/07/64	SS	8
3	100	Yogesh	17/07/64	IS	12
4	101	Amit	24/12/72	MIS	16
5	101	Amit	24/12/72	PM	8
6	101	Amit	24/12/72	IS	12
7	102	Omprakash	03/02/80	PWRC	8
8	102	Omprakash	03/02/80	PCOM	8
9	102	Omprakash	03/02/80	IP	16
10	103	Nitin	28/11/66	DT	10
11	103	Nitin	28/11/66	PCOM	8
12	103	Nitin	28/11/66	SS	8
13	104	Mahesh	01/01/86	DT	10
14	104	Mahesh	01/01/86	ADBMS	8
15	104	Mahesh	01/01/86	PWRC	8

- While eliminating the repeating groups, we have introduce redundancy into table. Faculty code, Name and Date of birth are repeated since the same faculty is multi skilled.
- To eliminate this, let us split the table into 2 parts; one with the non-repeating groups and the other for repeating groups.

Table 6.5.4

Faculty code	Faculty Name	Date of birth
100	Yogesh	17/07/64
101	Amit	24/12/72
102	Omprakash	03/02/80
103	Nitin	28/11/66
104	Mahesh	01/01/86

Faculty_code → Faculty_name, Date_of_Birth

The other table is those with repeating groups.

Table 6.5.5

Sr. No	Faculty code	Subject	Hours
1	100	DSA	16
2	100	SS	8
3	100	IS	12
4	101	MIS	16
5	101	PM	8
6	101	IS	12
7	102	PWRC	8
8	102	PCOM	8
9	102	IP	16
10	103	DT	10
11	103	PCOM	8
12	103	SS	8
13	104	DT	10
14	104	ADBMS	8
15	104	PWRC	8

- Faculty code is the only key to identify the faculty name and the date of birth. Hence, Faculty code is the primary key in the first table and foreign key in the second table.
- Faculty code is repeated in the Subject table. Hence, we have to take into account the 'SNO' to form a composite key in Subject table. Now, SNO + Faculty code can unique identity each row in this Table 6.5.5.
- Hence, the relation is now in Second Normal form.

(5) Anomalies (Problems)

Q. What are the various anomalies associated with RDBMS ?

(a) Insert Anomoly (Problems while inserting data)

- New record needed to insert in both tables.
- Inserting the records of various Faculties teaching same subject would result the redundancy of hours information.

(b) Update Anomoly (Problems while updating data)

- Updating some record in faculty table may exist in subject table.
- For a subject, the number of hours allotted to a subject is repeated several times. Hence, if the number of hours has to be changed, this change will have to be recorded in every instance of that subject. Any omissions will lead to inconsistencies.

(c) **Delete Anomoly** (Problems while deleting data)

- If a faculty leaves the organization, information regarding hours allotted to the subject is also needed to be deleted from subject table.
- Hence, This Subject table should therefore be further decomposed without any loss of information in 3rd normal form.

6.5.3 Third Normal Form (3NF)

(MSBTE - W-14, S-16, S-17)

Q. Explain 3NF with example.

W-14, S-16, S-17, 4 Marks

(1) Introduction

- This normal form is given by E.F. Codd (1971).
- This normal form used to minimize the transitive redundancy.
- In order to remove the anomalies that arose in Second Normal Form and to remove transitive dependencies, if any, we have to perform third normalization.

(2) Definition

- A relation is in 3NF, if it is in 2NF and no non-key attribute of the relation is transitively dependent on the primary key.
- 3NF prohibits transitive dependencies.
- In short 2NF means,
  1. It should be in 2NF.
  2. There should not be any transitive partial dependency.

(3) Example

Now let us see how to normalize the second table obtained after 2NF.

Table 6.5.6

Sr. No	Faculty code	Subject	Hours
1	100	DSA	16
2	100	SS	8
3	100	IS	12
4	101	MIS	16
5	101	PM	8
6	101	IS	12
7	102	PWRC	8
8	102	PCOM	8
9	102	IP	16
10	103	DT	10
11	103	PCOM	8
12	103	SS	8
13	104	DT	10
14	104	ADBMS	8
15	104	PWRC	8

- In this Table 6.5.6, hours depend on the subject and subject depends on the Faculty code and Sr. No.
- But hours are neither dependent on the faculty code nor the SNO. Hence, there exists a transitive dependency between Sr. No., Subject and Hours.
- If a faculty code is deleted, due to transitive dependency, information regarding the subject and hours allotted to it will be lost.
- For a table to be in 3rd Normal form, transitive dependencies must be eliminated.
- So, we need to decompose the table further to normalize it.

Table 6.5.7

Sr. No	Faculty code	Subject
1	100	DSA
2	100	SS
3	100	IS
4	101	MIS
5	101	PM
6	101	IS
7	102	PWRC
8	102	PCOM
9	102	IP
10	103	DT
11	103	PCOM
12	103	SS
13	104	DT
14	104	ADBMS
15	104	PWRC

Table 6.5.8

Subject	Hours
DSA	16
SS	8
IS	12
MIS	16
PM	8
PWRC	8
PCOM	8
IP	16
DT	10
ADBMS	8

- After decomposing the 'Subject' table we now have 'Fac_Sub' and 'Sub_Hrs' table respectively.

(4) Advantages

(i) Insertion

No redundancy of data for subject and hours while inserting the records.

(ii) Updation

Subject and hours are stored in the separate table.

So updation becomes much easier as there is no repetitiveness of data.

(iii) Deletion

Even if the faculty leaves the organization, the hours allotted to a particular subject can be still retrieved from the Sub_Hrs table.

Note : In most cases, third normal form is the sufficient level of decomposition. But some case requires the design to be further formalized up to the level of 4^{th} as well as 5^{th}. These are based on the concept of Multi Valued Dependency.

6.6 Solved Problems on Normalization Process

Ex. 6.6.1 : Relation R(A, B, C, D, E, F, G, H, I, J). Having following set of FDs, show whether it is in 2NF and 3NF.

AB → C

BD → EF

AD → GH

A → I

H → J

Soln. :

(a) Finding key

- For finding key we must find the super key of above relation.
- We start with minimum possible key

Closure of $\{A\}^+ \rightarrow \{A, I\}$

Closure of $\{A, B\}^+ \rightarrow \{A, B, C, I\}$

Closure of $\{B, D\}^+ \rightarrow \{B, D, E, F\}$

Closure of $\{A, D\}^+ \rightarrow \{A, D, G, H, I, J\}$

Closure of $\{A, B, D\} \rightarrow \{A, B, C, D, E, F, G, H, I, J\}$

- So Super key for above relation is A, B, D.

(b) Decomposition to 2NF

- **2NF** → Every non key attribute must full functionally dependent on primary key.
- As All other attribute can be determined by A, B, D primary key but each attribute not directly dependent on all A, B and D together.
- Example E, F depend only on B, D and not on A. It's a partial dependency.
- So, above relation not in 2NF.

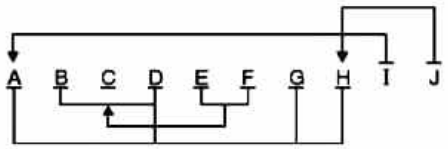


Fig. P. 6.6.1

Ex. 6.6.2 : Relation R (A, B, C, D, E). Having following set of FDs, convert it to 3NF.

A → BD B → C D → E

Soln.:

(a) Finding key

Closure of A : $\{A\}^+$ = {A,B,D,C,E}

A can be the key. No other single attribute can be key.

(b) Decomposition to 2NF

- **2NF** → every non key attribute must full functionally dependent on primary key.
- All attribute can be determined by primary key A and each attribute directly dependent on A.

∴ Relation in 2NF

(c) Decomposition to 3NF

- For relation to be in 3NF there should not be any Transitive dependency

  In above relation A → D and D → E

  ∴ E is transitively dependent on A

  ∴ Transitive dependency exists.

  ∴ Relation not in 3NF

- Composing to 3NF

  R_1 (A, B, C, D)

  R_2 (D, E)

- Now, relation in 3NF

(d) Decomposition to BCNF

- **BCNF** → every determinant must be candidate key.
- In relation R_1 A is Determinant and it is candidate key also.
- In relation R_2 D is Determinant and it is candidate key also.

  ∴ Relation R is in BCNF

Ex. 6.6.3 : We are given Relation R with Attributes A, B, C, D, E, F and the FDs as below, Find and explain which Armstrong's Axioms can be applied here to find closure,

A → BC B → E CD → EF

Soln. :

1. $\{A\}^+ = \{A, B, C, E\}$

Axioms :

(a) Axiom of Reflexivity

$A \rightarrow A$ ...(1)

(b) Axiom of Pseudo transitivity

$A \rightarrow BC$ and $B \rightarrow E$ ...(2)

$\therefore A \rightarrow EC$ ...(3)

(c) Decomposition from Equations (1), (2) and Equation (3)

$A \rightarrow A, A \rightarrow B ; A \rightarrow C ; A \rightarrow E$

$\therefore \{A\}^+ = \{A, B, C, E\}$

2. {A, B} = {A, B, C, E}
3. {A, D} = {A, B, C, D, E, F}

Review Questions

Q. 1 Write a short note :

a. Functional dependency

b. Decomposition

c. Normalization

Q. 2 Write a short note on decomposition.

Q. 3 What is normalization ? Explain 1NF, 2NF, 3NF and BCNF with suitable example.

Q. 4 Describe decomposition. What are the desirable properties of decomposition ?

Q. 5 Define normalization

Q. 6 What is the need of normalization.

Q. 7 We are given Relation R with Attributes A, B, C, D, E, F and the FDs as below, Find and explain which Armstrong's Axioms can be applied here to find Closure.

A → BC B → E C D → EF

Q. 8 Give Armstrong's axioms.

❑❑❑

Structured Query Language

Syllabus

Introduction to Structure Query Language,

Data Types in SQL, Components of SQL : DDL, DML, DCL, DQL

DDL Commands : CREATE, ALTER, DROP, TRUNCATE, DESC, RENAME

Data Integrity Constraint : Types of Data Integrity Constraint : IO Constraint – Primary Key, Foreign Key, Unique Key Constraint , Business Rule Constraint : Null, Not Null and Check Constraint.

DML Commands : INSERT, UPDATE, DELETE

DCL commands : COMMIT, SAVEPOINT, ROLLBACK, GRANT and REVOKE

DQL Commands : SELECT

SQL Operators : Arithmetic Operators, Comparison Operators, Logical Operators, Set Operators, Range Searching Operators – Between Pattern Matching operators - Like

7.1 Overview of SQL

- **SQL (Structured Query Language)** is a computer language aimed to store, manipulate, and retrieve data stored in relational databases.
- It was developed by IBM Research in the mid 70's and standardized by ANSI in 1998.
- The first commercial relational database was released by Relational Software (later called as Oracle).
- SQL is a keyword-based language and each statement begins with a unique keyword.
- SQL syntax is not case sensitive.

7.1.1 Role of SQL

Q. Explain role of SQL with example.

1) SQL is an interactive query language which can be used to retrieve data from database.
2) SQL is a database programming language which can be used along with programming language to access data from database.
3) SQL is a database administration language which can be used to monitor and control data access by various users.
4) SQL can be used as an Internet data access language.

7.1.2 SQL Data types

(MSBTE - W-13)

Q. Explain data types in SQL. W-13, 2 Marks

Built-in Data Types

- The basic data types available with SQL standard are as enlisted below, all data types may not be supported by SQL server or Oracle.
- Data types include numeric, character string, bit string, Boolean and datetime.

1. Numeric data types

This datatype is used to store a number values that can be decimal or floating point values.

(a) Integer number of various size

These types of system are used to store natural numbers which are not having any decimal values.

Example: 111, 23 etc.

As per size of number we can use following types of integers.

(i) INTEGER (p)

(ii) INTEGER or INT

(iii) SMALLINT

(iv) BIGINT

Table 7.1.1

Data type	Abbrivation	Description	Range
INTEGER	INT	Integer numerical with precision 10.	– 2,147,483,648 to 2,147,483,647
SMALL INTEGER	SMALLINT	Integer numerical with precision 5.	– 32768 to 32767
BIG INTEGER	BIGINT	Integer numerical with precision 19.	– 9, 223, 372, 036, 854, 775, 808 to 9, 223, 372, 036, 854, 775, 807

(b) Floating point numbers of various precision

This system used for storing decimal numbers which may be of greater size than integers.

Example : 11.2, 12.3 etc.

As per size of floating number we can use following types of numbers.

(i) FLOAT or REAL

(ii) DOUBLE PRECISION

Table 7.1.2

Data Type	Abbrivation	Description	Range
FLOAT (p)	FLOAT (p)	Approximate numerical with mantissa precision p.	$1 \le p \le 45$ Zero or absolute value 10^{-999} to 10^{+999}
REAL	REAL	Approximate numerical with mantissa precision 7.	Zero or absolute value 10^{-38} to 10^{+38}

(c) Formatted numbers

This system used for storing some special numbers which may be of greater size than integers and floating point numbers.

Example : 1.12342 (Numeric(1,5)),12.234 (Numeric(2,3)) etc.

(i) DECIMAL or DEC(i, j)

(ii) NUMERIC(i, j)

Where i = Precision = Total number of digits after decimal point

j = Scale = Total number of digits after decimal point. (default value is 0)

Table 7.1.3

Data Type	Description	Range
DECIMAL(p, s)	Exact numerical with precision p and scale s.	$1 \le p \le 45$ $0 \le s \le p$
NUMERIC(p, s)	Exact numerical with precision p and scale s.	$1 \le p \le 45$ $0 \le s \le p$

2. Character string data type

– This data type is used to store a character string which is combination of some alpha bates and enclosed in single quotation marks.

Example : 'Mahesh','abc' etc.

(a) Fixed length : CHAR (n), Where n = number of characters

Example : If 'abc' is stored in char (10) will be stored as 'abc '.

(abc padded with 7 blank spaces)

(b) Varying length: VARCHAR (n) Where n = maximum number of characters

Example : If 'abc' is stored in VARCHAR (10) will be stored as 'abc' (no balnk spaces)

Table 7.1.4

Data Type	Description	Range
CHAR(n)	Character string of fixed length n.	$1 \leq n \leq 15000$
VARCHAR(n)	Variable length character string of maximum length n.	$1 \leq n \leq 15000$

3. Date time data type

(a) Date

- The DATE data type has ten positions and its components are YEAR, MONTH and DAY in form YYYY-MM-DD.
- The length is 10.
- Generally not supported by SQL server.(Supported by DB2)
- **Example :** Date '2009-01-01' (as 'YYYY-MM-DD')

(b) Time

- The TIME data type has at least eight positions and its components are HOUR, MINUTES and SECOND in form HH:MM:SS [.sF] where F is the fractional part of the SECOND value.
- Generally not supported by SQL server.
- If a second's precision is not specified, *s* defaults to 0. The length is 8 (or 9 + *s*, if *s* > 0).
- **Example :** Time '11:16:59' (as 'HH:MM:SS')

(c) Timestamp / datetime

- The TIMESTAMP data type includes both date and time fields, plus a minimum of six positions and for decimal fractions of second and optional with TIMEZONE Qualifier.
- Represented using the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND in the format YYYY-MM-DD HH:MM:SS[.sF] where F is the fractional part of the SECOND value.
- If a second precision is not specified, *s* defaults to 6. The length is 26 (or 19, if s = 0 or 20+*s*, if *s* > 0).
- **Example :** TimeStamp '2009:01:01 11:16:59 648302' (as 'YYYY-MM-DD HH:MM:SS TIMEZONE')
- Current TimeStamp : Local date and time without timezone

7.2 Components of SQL: DDL, DML, DCL, TCL

1. Data Definition Language (DDL)

Q. Explain DDL commands with example.

- To create, drop or alter database schema, we use Data Definition Language (DDL).
- DDL statements are used to build and modify the structure of your tables and other objects in the database.
- When you execute a DDL statement, it takes effect immediately, as it is **Autocommited** into database. Hence no rollback operation (Undo) can be performed with these set of commands.
- Database objects are any data structure created in database.

  Example : Table, view, sequence etc.

2. Data Manipulation Language (DML)

Q. Explain DML commands.

- **Data Manipulation Language (DML)** statements are used for manipulating or managing data in database.
- DML commands are not auto-committed like DDL statements.
- It means changes done by DML command can be rolled back. In other words the DML statements do not implicitly commit the current transaction.

- The set of DDL commands are as follows,
  1. INSERT Statement: To add some data to Database table we need to use this command.
  2. DELETE Statement: To add remove data from table we need to use this command.
  3. UPDATE Statement: To add change data added to table we need to use this command.
- DML is set of commands used to,

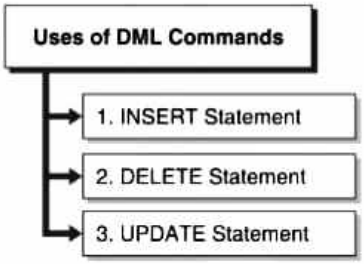


Fig. 7.2.1 : DML commands

3. Data Control Language (DCL)

Q. Write a note on DCL.

- **Data Control Language (DCL)** is used to control various user actions like inserting data, updating data, deleting data or viewing data.
- To perform any operation in the database user needs **privileges** like creating tables, index or views.
- DCL is set of commands used to,
- **Grant** : Gives some privilege to user for performing task on database.
- **Revoke :** Take back permissions given to user.

4. Transaction Control Language (TCL)

- Any SQL query can be executed with two basic operations READ and WRITE on the database tables.
- After executing SQL query, we must specify its final action as commit (save data) or abort (or revert back changes).
- The COMMIT statement ends the operations and makes all changes made to the data permanently, on successful completion.
- ABORT terminates and undo's all the actions done so far.

7.3 Data Definition Language (DDL)

(MSBTE - W-18)

Q. List four DDL commands with syntax. **W-18, 2 Marks**

Q. Explain DDL commands with example.

- To create, drop or alter database schema, we use Data Definition Language (DDL).
- DDL statements are used to build and modify the structure of your tables and other objects in the database.
- When you execute a DDL statement, it takes effect immediately, as it is **Autocommited** into database. Hence no rollback operation (Undo) can be performed with these set of commands.
- Database objects are any data structure created in database.
- **Example :** Table, view, sequence etc.

7.3.1 Viewing Structure of Table - DESC Command

- It is possible to view all details of database and tables from data dictionary.
- If you have not yet selected any database, the result is NULL.
- To find out tables present in currently used database.

```
mysql>SHOW TABLES;
+---------------------------+
| Tables_in_TestDB          |
+---------------------------+
| Test                      |
| Emp                       |
+---------------------------+
```

- It is difficult to remember name of a database or table along with its structure.
- MySQL addresses this problem through several statements that provide information about the databases and tables it supports.
- To view the structure of all tables available in database.

Syntax

```
DESCRIBE TABLE Table_name;
        OR
DESC TABLE Table_name;
```

Example

```
DESCRIBE TABLE test;
```

```
mysql>DESCRIBE test;
+---------+------------ ----+--------+------+---------+---------+
| Field   | Type          | Null | Key | Default | Extra   |
+---------+---------------+------+-----+---------+---------+
| name    | varchar(20) | YES  |     | NULL    |         |
| id      | varchar(10) | YES  |     | NULL    |         |
+---------+---------------+------+-----+---------+---------+
```

7.3.2 Create Table - CREATE Command

(MSBTE - W-16)

Q. Give the syntax and example of CREATE Command. **W-16, 2 Marks**

Q. Explain CREATE command with example.

- To create database object like table, database, view etc. we use Data Definition Language (DDL).
- DDL statements are used to build and modify the structure of your tables and other objects in the database.
- CREATE statement is used to create any database object.

Syntax

```
CREATE TABLE <Table_Name>
(  Column_1 datatype,
   Column_2 datatype,
   . . .
   Column_n datatype
);
```

Example

```
CREATE TABLE Employee
(     Eid varchar2(10),
First_Name char (50),
Last_Name char (50),
Address char (50),
City chars (50),
Country char (25),
Birth_Date date
);
```

- To view the structure of table created in database;

```
DESCRIBE Employee;
```

7.3.3 Alter Table - ALTER Command

(MSBTE - W-14, W-16, W-17)

Q. Explain Alter command. Give syntax of add and modify option. **W-14, W-16, W-17, 4 Marks**

Q. Explain ALTER command with example.

- Once table is created in database, we may require to change structure of database object. This can be done with help of ALTER command.
- The alter table statement may be used as you have seen to specify primary and foreign key constraints, as well as to make other modifications to the table structure.
- Key constraints may also be specified in the CREATE TABLE statement.

Syntax

```
ALTER TABLE <Table_Name>
ADD Column_1 datatype ;
OR
ALTER TABLE <Table_Name>
Modify Column_1 New_datatype;
OR
ALTER TABLE <Table_Name>
DROP Column_1;
```

Example

```
ALTER TABLE Employee
ADD Address Varchar2 (100);
```

To view the changed structure of table

```
DESCRIBE TABLE Employee;
```

7.3.4 Drop Table - DROP Command

(MSBTE - W-16, W-17)

Q. Explain DROP Command with syntax and example. W-16, W-17, 2 Marks

Q. Explain DROP command with example.

- This command can be used to remove database objects from our DBMS.
- DROP command removes data from your database.

Syntax

```
DROP TABLE <Table_Name>
```

Example

If we want to permanently remove the Employee table that we created, we'd use the following command

```
DROP TABLE Employee
```

- Similarly, the command below would be used to remove the entire employees database

```
DROP DATABASE employees
```

7.3.5 Rename Table

(MSBTE - W-13, W-16)

Q. Give the syntax of RENAME command with example. W-13, W-16, 4 Marks

Q. Explain RENAME command with example.

- We can rename any table object at any point of time.

Syntax

```
RENAME TABLE <Table_Name>
To <New_Table_Name>;
```

Example

```
RENAME TABLE Employee
To EMP;
```

7.3.6 Truncate Table

Q. Explain TRUNCATE command with example.

- Use the Truncate statement to delete all the rows from table permanently.
- It works same as "DELETE FROM <table_name>" but, Truncate cannot be rolled back.
- If any delete triggers are defined on the table, then the triggers are not fired on truncate table. It de-allocates memory space. So that the free space can be used by other tables.

Syntax

```
TRUNCATE TABLE EMP;
```

7.4 Data Integrity Constraints

Q. Why are relational database constraints in a database?

- Constraints makes sure that only authorised user will make modifications to database and changes should not lead to loss of data consistency and correctness.
- These concepts makes sure that correctness and completeness of data stored in the database.
- This objective can never be guaranteed, one cannot ensure that every entry made in database is accurate.
- Some example of incorrect data is as follows,
  1. Student taking admission to a branch which is not available in the college.
  2. Employee assigned with non existing department.
  3. Sometimes inconsistency introduced due to system failures.

7.4.1 Types of Data Integrity Constraints

1. Domain Relational Constraints
2. Entity Relational Constraints
3. Referential Relational Constraints

1 Domain Relational Constraint – Business Rule Constraint

Q. Why are domain integrity important in a database?

- Domain constraints allow us to test whether the values inserted into the database are correct or not.
- The CREATE TABLE Command may also include domain constraints which can check integrity of database.
- These domain constraints are the most basic form of integrity constraint.

Types of Constraints

- Null Constraint / Required Data Constraint
- CHECK Constraint / User Defined Constraint
- DEFAULT Constraint

(a) Required Data Constraint / Null Constraint

(MSBTE – W-14, W-15)

Q. How to apply NOT NULL constraint at the time of table creation ? Give syntax.

W-14, 2 Marks

Q. Explain not null constraints by suitable example.

W-15, 4 Marks

- Database may have some attributes mandatory like, user registration must have email address.
- These attributes (columns) in a database are not allowed to contain NULL values or blanks.

Example

In the student table, student must have an associated student name.

```
name char(100) NOT NULL
Father_name char(100) NULL
Sur_name char(100)
```

- Therefore, the NAME column in the STUDENT table is a required data.
- Whereas, the Father_name and Sur_name column in the STUDENT table is a nullable data.
- The DBMS can prevent user from inserting NULL values in any column with the help of such constraints.
- Some attributes (columns) in a database are not allowed to contain NULL value. NULL values are values which are unknown, unassigned or missing attribute values.

Example : In the student database, every student must have an associated student name. Student_name should not be NULL.

```
CREATE TABLE Student
(Name char (25) NOT NULL)
```

(b) Check Constraint

- The **check constraint** is used to ensure that attribute value satisfies specific condition as specified by data requirements.
- Suppose in Student Table, gender of student can be Male or Female only.
- The DBMS can prevent user from entering incorrect or other data in database table.

Example

Table with student entity having gender which can be M or F.

Hence, attribute gender can take only two values either 'M' or 'F'.

```
gender char(1) CHECK (gender in ('M','F')
```

- Use of **check constraint** is to ensure that attribute value satisfies specific user defined condition.
- **Example :** Table with customer entity having name, cid and gender which can be M or F.
- Hence, attribute gender can take only two values either 'M' or 'F'.

```
CREATE TABLE customer
(Name char (25) NOT NULL,
Gender char (1),
CHECK (Gender IN ('M','F')))
```

(c) Default Constraint

- Default keyword is used to add a default specified value, if no attribute value is provided by user.
- It avoids addition of NULL value by inserting default value as specified.

Example

Table with customer entity having name and gender.

If name is not added for customer that will be taken as 'Unknown' if we specify DEFAULT value of NAME column to 'UNKNOWN'

```
NAME varchar(50) DEFAULT 'UNKNOWN'
```

- Default keyword is used to add some value if no attribute value added for tuple.

Example : Table with customer entity having name, cid and gender in which cid is primary key. If name is not added for customer that will be taken as 'Unknown'.

```
Create table customer
(Name char (25) DEFAULT 'UNKNOWN')
```

2. Entity Relational Constraints

Q. Why are entity integrity important in a database?

- Entity constraints allow us to test whether the tuple (entity) inserted into the database are correct or not.
- The **create table** Command may also include Entity constraints which can be the primary key of table.

3. Referential Relational Constraints

- In case of unique constraint, no two tuples can have equal value for same attributes.
- This constraint says that attributes forms candidates key, which allows one Null value which is unique by itself.
- This UNIQUE constraint can be applicable to user defined domain declaration also.

```
EMAIL varchar(30) UNIQUE
```

Example

```
Create table customer
        (   Name char(25),
            Cid char(10),
            Email char(50) UNIQUE)
```

7.4.2 Primary Key Constraint

- A table in a relational database has one column or combination of some columns whose values uniquely identifies a single row in the table. This column or combination of columns is called the **primary key** of the table.
- Primary key attribute is same as unique key constraint with Not NULL constraints.
- The main difference in unique constraint and primary key constraint is that one null value is allowed in unique constraint which can be treated as unique value while nulls are not allowed in primary key constraint.
- For example, each row of the STUDENT table has a unique set of values in its STUDENT_ID column, which uniquely identifies the student represented by that row.
- Duplicate values are not allowed in primary key column, because they can cause problems in distinguishing one entity from another (entity may be an employee).
- The DBMS can prevent user from inserting same data values in a column again and again.

```
STUDENT_ID char(10) PRIMARY KEY
```

Example

Table with customer entity having name, cid and gender in which cid is primary key.

```
Create table customer
(Name char (25),
Cid char (10) PRIMARY KEY)
```

7.4.3 Referential Integrity - Foreign Key

Q. Why are referential integrity important in a database?

Q. Explain the purpose of foreign key.

- A value appearing in one relation (table) for a given set of attributes also appears for another set of attributes in another relation (table). This is called **referential integrity.**
- The referential integrity constraint is specified between two tables to maintain the consistency among tuples in the two tables.
- The tuple in one relation refers only to an existing tuple in another relation.

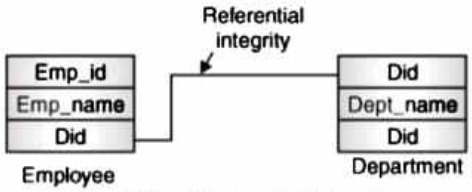


Fig. 7.4.1 : Referential Integrity

Employee and Department table

Emp Table		
Emp_Id	**Emp_name**	**Did**
1	Sachin	20
2	Suhas	10
3	Jay	20
4	Om	10

Department Table	
Did	**Dept_name**
10	HR
20	TIS
30	L&D

- In the above example 'Emp' table has 'Did' as foreign key reference this is called as referential integrity.
- Here we are forcing database to check the 'Did' value key from the 'department' table while inserting any value of 'Emp' table in Did column if there is no value existing in department table of that 'Did' then we cannot insert that value in 'Emp' table. This helps to maintain data consistency.

Example

For above shown relation we can write table as follows,

```
Create table Emp ( Emp_id integer,
 Emp_name varchar (100) not null,
 Did as integer references department(did))
Create table Department  ( Did integer,
Dept_name varchar (100) not null,
                    Primary key (did))
```

7.5 Data Manipulation Language (DML) - Insert, Delete and Update

(MSBTE - W-14, S-15)

Q. List DML commands. **W-14, S-15, 2 Marks**

DML is set of commands used to,

1. Insert data into table
2. Delete data from table
3. Update data of table

7.5.1 INSERT Command

(MSBTE – S-14)

Q. Write syntax of insert command. Demonstrate with suitable example. **S-14, 4 Marks**

Insert statement used to add records to the existing table.

Syntax

```
INSERT INTO<Table_Name>|(Column1, . . . ,
ColumnN)
VALUES (column1. . . , columnN);
```

Example

```
/* To add data in employee table*/

INSERT INTO Employee
VALUES (1001, 'Mahesh');

INSERT INTO Employee
VALUES (NULL, 'Jayendra');

INSERTINTO Employee (Name,Eid)
VALUES ('Sachin' , 1002);

INSERTINTO Employee (Name,Eid)
VALUES ('Suhas' , NULL);
```

To check inserted rows, write following query.

```
/*To Print all data in employee table*/
    SELECT  *
    FROM    Employee;
```

Output

Eid	Name
1001	Mahesh
1002	Sachin
NULL	Jayendra
NULL	Suhas

7.5.2 DELETE Command

(MSBTE - W-15, W-16, W-17)

Q. Describe how to delete the data from table with an example. **W-15, W-16, W-17, 4 Marks**

Delete statement is used to delete some or all records from the existing table.

Syntax

```
DELETE
FROM    <Table_Name>
WHERE   Condition;
```

Example

```
DELETE
FROM    Employee
WHERE   Eid IS NULL;
```

If we omits the WHERE condition then all rows will get deleted.

To check inserted rows write following query.

```
SELECT  *
FROM    Employee;
```

Eid	Name
1001	Mahesh
1002	Sachin

7.5.3 UPDATE Command

Insert statement used to modify the records present in existing table.

Syntax

```
UPDATE  <Table_Name>
SET     column1=value
WHERE   condition;
```

Example

```
UPDATE  Employee
SET     Eid=1003
WHERE   name = 'suhas';
```

If we commit the WHERE condition then all rows will get updated with given value.

To check inserted rows write following query.

```
SELECT  *
FROM    Employee;
```

Output

Eid	Name
1001	Mahesh
1002	Sachin
1003	Suhas
NULL	Jayendra

7.6 DQL Commands

Q. Explain 'SELECT' clause. Give its syntax.

- DRL is set of commands used to retrieve data from database server.
- This language also includes all functions used to manipulate the data in database just for display purpose like aggregate functions.
- This clause is used for selecting various attributes or columns of a table.

Syntax

```
SELECT  {ALL / DISTINCT}
            [ * /Column1,Column2...]
FROM    <Table_Name>
```

7.6.1 SELECT Clause - Select all Columns

- SELECT is basic statement used to retrieve all or some columns of data from table.
- We can select all columns from table by specify * as column name.

Syntax

```
SELECT  *
FROM    <Table_Name>
```

Example

Selecting all columns data in Faculty table

```
SELECT  *
FROM Faculty
```

Output

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Yogesh	17/07/64	DSA	16
100	Yogesh	17/07/64	SS	8
100	Yogesh	17/07/64	IS	12

Faculty_code	Faculty_Name	DOB	Subject	Hours
101	Amit	24/12/72	MIS	16
101	Amit	24/12/72	PM	8
101	Amit	24/12/72	IS	12
102	Omprakash	03/02/80	PWRC	8
102	Omprakash	03/02/80	PCOM	8
102	Omprakash	03/02/80	IP	16
103	Nitin	28/11/66	DT	10
103	Nitin	28/11/66	PCOM	8
103	Nitin	28/11/66	SS	8
104	Mahesh	01/01/86	DT	10
104	Mahesh	01/01/86	ADBMS	8
104	Mahesh	01/01/86	PWRC	8

7.6.2 SELECT Clause - Select Specific Columns

- SELECT statement can be used to retrieve specific columns of data from table to maintain security of data.
- We can select columns from table specifying names of required columns.

Syntax

```
SELECT   Column_1_Name,....
FROM     <Table_Name>
```

Example

Selecting all columns data in Faculty table

```
SELECT  Faculty_Name, Subject
FROM    Faculty
```

Output

Faculty_Name	Subject
Yogesh	DSA
Yogesh	SS
Yogesh	IS
Amit	MIS
Amit	PM
Amit	IS
Omprakash	PWRC
Omprakash	PCOM
Omprakash	IP
Nitin	DT
Nitin	PCOM
Nitin	SS
Mahesh	DT
Mahesh	ADBMS
Mahesh	PWRC

7.6.3 SELECT Clause - Select Unique Records

- SELECT is basic statement used to retrieve all or some columns of data from table as explained in above examples.
- SELECT can have two options as SELECT ALL records in which query result including duplicate records along with unique records or SELECT DISTINCT records to show only unique records.
- SELECT ALL is default and SELECT DISTNCT will search for distinct rows of in table.
- We can either select all columns from table by specify * as column name or we can specify the required name of columns. In any case DISTINCT will look for unique record set for display purpose only.

  (It cannot remove duplicate records from original table.)

Syntax

```
SELECT   {ALL / DISTINCT}
                [ * /Column1, Column2...]
FROM     <Table_Name>
```

Example 1 : To select all faculties from above faculty table.

```
SELECT ALL Faculty_Name
FROM    Faculty ;
```

Output

Faculty_Name
Yogesh
Yogesh
Yogesh
Amit
Amit
Amit
Omprakash
Omprakash
Omprakash
Nitin
Nitin
Nitin
Mahesh
Mahesh
Mahesh

Example 2 : To select different Faculty_names from above faculty table.

```
SELECT DISTINCT Faculty_Name
FROM Faculty ;
```

Output

Faculty_Name
Yogesh
Amit
Omprakash
Nitin
Mahesh

7.7 Ordering Query Results - ORDER BY Clause

Q. Explain 'ORDER BY' clause.

- The ORDER BY keyword is used to sort the result-set by a specified column.
- The ORDER BY keyword sorts the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.

Syntax

```
SELECT column_name
FROM table_name
ORDER BY column_name ASC|DESC
```

Example 1

Table 7.7.1 : Faculty table

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Yogesh	17/07/64	DSA	16
101	Amit	24/12/72	MIS	16
102	Omprakash	03/02/80	PWRC	8
103	Nitin	28/11/66	DT	10
104	Mahesh	01/01/86	DT	10

- Now we want to select all the Faculties from the table above, however, we want to sort the Faculty by their Faculty_Name.
- We use the following SELECT statement :

```
SELECT *
FROM Faculty
ORDER BY Faculty_Name
```

- In second query ORDER BY 2 indicates order by second column of table.

Output

Faculty_code	Faculty_Name	DOB	Subject	Hours
101	Amit	24/12/72	MIS	16
104	Mahesh	01/01/86	DT	10
103	Nitin	28/11/66	DT	10
102	Omprakash	03/02/80	PWRC	8
100	Yogesh	17/07/64	DSA	16

Example 2

- Now we want to select all the Faculties from the table above, however, we want to sort the Faculties descending by their Faculty_Name.

- We use the following SELECT statement:

```
SELECT    *
FROM      Faculty
ORDER BY  Faculty_Name DESC
```

Output

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Yogesh	17/07/64	DSA	16
102	Omprakash	03/02/80	PWRC	8
103	Nitin	28/11/66	DT	10
104	Mahesh	01/01/86	DT	10
101	Amit	24/12/72	MIS	16

7.8 SQL Operators

7.8.1 Filtering Query Results - WHERE Clause

Q. Explain WHERE Clause.

- The ORDER BY keyword is used to sort the result-set by a specified column.
- The ORDER BY keyword sorts the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.

Row Selection – WHERE Clause

- Row Selection operation in SQL query is done using 'WHERE' Clause.
- In WHERE clause we can write row qualifier condition that row must satisfy to print in final result of query.
- If WHERE clause is not written in that case all rows will be printed by default.
- We will ignore WHERE condition to print all available rows in table.

Syntax

```
SELECT    *
FROM      Table_Name
[WHERE condition]
```

Examples

1. Select all departments from department table.

```
SELECT    *
FROM      departments;
```

Output

DEPT_ID	DEPT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500
60	IT	103	1400
80	Sales	149	2500
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting	203	1700

2. Select departments having location id 1700 from department table.

```
SELECT    *
FROM      departments
WHERE     location_id=1700;
```

Output

DEPT_ID	DEPT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
90	Executive	100	1700
110	Accounting	205	1700
190	Contracting	203	1700

Important

- SQL statements are not case sensitive.
- Keywords cannot be abbreviated or split across lines.
- Clauses are usually placed on separate lines.
- Indents are used to enhance readability.

7.8.2 Arithmetic Operators

- To perform various arithmetic operations, we can use these operators.
- A comparison operator is a mathematical symbol used to compare two values in mathematical expression.

- The result of an arithmetic operators can be any numeric value.
- The various comparison operators are enlisted as given in table 7.8.2.

Table 7.8.2

1.	+	Addition
2.	-	Subtraction
3.	*	Multiplication
4.	/	Division
5.	%	Reminder

Examples

1. Find all Faculty not teaching 'DT'

```
SELECT  *
FROM    Faculty
WHERE   Subject <> 'DT'
```

2. Find all Faculty teaching more than 10 hours.

```
SELECT  *
FROM    Faculty
WHERE   Hours > 10
```

7.8.3 Comparison Operators

- To compare attribute value of tuple with arithmetic comparisons.
- A comparison operator is a mathematical symbol used to compare two values in mathematical expression.
- Comparison operators in conditions will be used to evaluate expression. The result of a comparison can be TRUE, FALSE, or UNKNOWN.
- The various comparison operators are enlisted as given in Table 7.8.3.

Table 7.8.3 : Comparison operators

1	=	Equal to
2	<	Less Then
3	<=	Less Then Equal to
4	>	Greater Then
5	>=	Greater Then Equal to
6	<>	Not Equal to

Examples

1. Find all Faculty not teaching 'DT'

```
SELECT  *
FROM    Faculty
WHERE   Subject <> 'DT'
```

2. Find all Faculty teaching more than 10 hours.

```
SELECT  *
FROM    Faculty
WHERE   Hours > 10
```

7.8.4 Logical Operators (AND, OR, NOT)

- The Logical operators will accept expression and return result as true or false to combine one or more true or false values.

Table 7.8.4

Operator	Description
AND	Logical AND compares between two Booleans expressions to returns true when both expressions are true otherwise false
OR	Logical OR compares between two Booleans expressions to return true when one of the expression is true otherwise false
NOT	Not takes a single Boolean expression and changes its value from false to true or from true to false.

1. Find all Faculty not teaching 'DT' and taught hours more than 10.

```
SELECT  *
FROM    Faculty
WHERE   Subject <> 'DT' AND Hours > 10
```

2. Find all Faculty teaching more than 10 hours or taught hours more than 10.

```
SELECT  *
FROM    Faculty
WHERE   Subject <> 'DT' OR Hours > 10
```

3. Find all Faculty not teaching 'DT'.

```
SELECT  *
FROM    Faculty
WHERE   NOT Hours > 10
```

7.9 SET Operations

(MSBTE - W-18)

Q. Explain set operations with examples. **W-18, 4 Marks**

- The results of two queries can be combined using the set operations union, intersection and difference.

```
Query_1 UNION        [ALL] Query_2
Query_1 INTERSECT    [ALL] Query_2
Query_1 EXCEPT       [ALL] Query_2
```

- In order to calculate the union, intersection or difference of two queries, the two queries must be "union compatible", which means that they both return the same number of columns, and that the corresponding columns have compatible data types.

7.9.1 Union Operation

- Union effectively appends the result of Query_2 to the result of Query_1.
- Furthermore, it eliminates all duplicate rows, in the sense of **DISTINCT**, unless **ALL** is specified.

DISTINCT : Duplicate row shown only once.

ALL : Duplicate row shown only once.

Syntax

```
SELECT *
FROM   stud ;
```

Output

Id	Name	Semester	Diploma	Gender
1	Fred	2	Bio	M
3	Tom	1	Bio	M
4	John	3	Phy	M
2	Lisa	2	Bio	F

(4 Rows Selected)

Syntax

```
SELECT *
FROM   stud_extern ;
```

Output

Id	Name	Semester	Diploma	Gender
3	Tom	1	Bio	M
4	John	3	Phy	M
5	Simon	4	Inf	F

(3 Rows Selected)

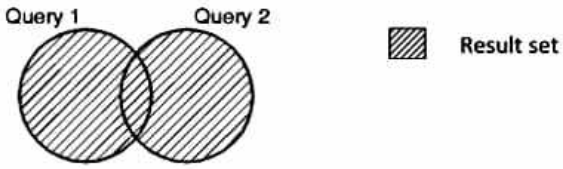


Fig. 7.9.1 : Union operation

Syntax

```
SELECT *
FROM   stud
UNION
SELECT *
FROM   stud_extern ;
```

Output

Id	Name	Semester	Diploma	Gender
1	Fred	2	Bio	M
2	Lisa	2	bio	F
3	Tom	1	Bio	M
4	John	3	Phy	M
5	Simon	4	Inf	F

(5 Rows Selected)

7.9.2 Intersect Operation

Returns all rows that are both in the result of Query_1 and in the result of Query_2. Duplicate rows are eliminated unless **ALL** is specified.

Example

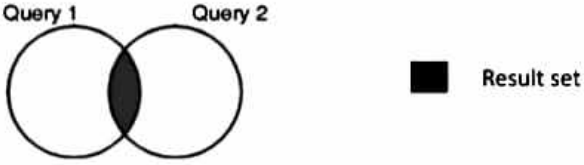


Fig. 7.9.2 : Intersect operation

Syntax

```
SELECT  *
FROM    stud
INTERSECT
SELECT  *
FROM    stud_extern;
```

Output

Id	Name	Semester	Diploma	Gender
3	Tom	1	Bio	M
4	John	3	Phy	M

(2 Rows Selected)

7.9.3 Except Operation

- Returns all rows that are in the result of query1 but not in the result of query2.
- Again, duplicates are eliminated unless **ALL** is specified.

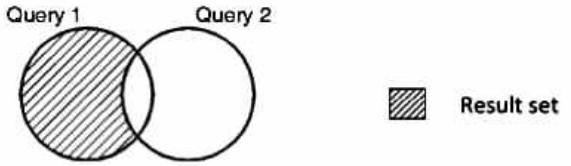


Fig. 7.9.3

Syntax

```
SELECT  *
FROM    stud
EXCEPT
SELECT  *
FROM    stud_extern ;
```

Output

Id	Name	Semester	Diploma	Gender
1	Fred	2	Bio	M
2	Lisa	2	bio	F

(2 Rows Selected)

7.9.4 Nesting SET Operations

- It is possible to use one Set operations within other set operation called as nesting operation
- It is possible to nest and chained like below

Syntax

```
Query1 UNION query2 UNION query3
```

Example

```
(Query1 UNION query2) UNION query3
```

7.10 Range Searching Operators - Between

(MSBTE - W-13)

Q. Explain range searching operator, "BETWEEN" **W-13, 2 Marks**

- BETWEEN statement used to find out all tuple whose attribute value is ranging between given range of values.
- This clause also includes the boundary values.

Examples

1. **Find all Teachers taken hours 10 to 20.**

```
SELECT  *
FROM    Faculty
WHERE   Hours BETWEEN 10 AND 20
```

2. **Find all Teachers not taken hours between 10 to 20.**

```
SELECT  *
FROM    Faculty
WHERE   Hours NOT BETWEEN 10 AND 20
```

7.11 Pattern Matching Operation - Like

(MSBTE – W-13)

Q. Explain pattern matching operator - "LIKE". **W-13, 2 Marks**

- Pattern matching statements are used to find out particular string in attribute value.
- LIKE is keyword that is used in the WHERE clause for pattern matching.
- Determines whether a specific character string matches a specified pattern.

– SQL provides two wildcard characters for use with LIKE.

Percentage sign ('%')

- Percentage sign means any string of zero or more characters.
- Percent sign (%) is used to define the start and ending or your string.

Underscore sign ('_')

- Under Sore sign means a single character.
- Under Sore sign is used to determine letter at particular position.

Syntax

```
SELECT      Column_name
FROM        Table_name
WHERE       column_name LIKE {PATTERN}
```

{PATTERN} often consists of wildcards.

Example

Consider the "Faculty" table (Table 7.11.1).

Table 7.11.1

Faculty_code	Faculty_Name
100	Yogesh
102	Omprakash
103	Nitin
104	Mahesh
101	Amit

Pattern 1 : Determine a string which starts with given a letter.

Find all faculties whose name starts with 'A'.

Syntax

```
SELECT      *
FROM        Faculty
WHERE       Faculty_Name LIKE 'A%'
```

Output

Faculty_code	Faculty_Name
101	Amit

Find all faculties whose name starts with 'Om'.

Syntax

```
SELECT      *
FROM        Faculty
WHERE       Faculty_Name LIKE 'Om%'
```

Output

Faculty_code	Faculty_Name
102	Omprakash

Pattern 2 : Determine a string which End with a given letter.

Find all faculties whose name ends with 'h'.

Syntax

```
SELECT      *
FROM        Faculty
WHERE       Faculty_Name LIKE '%h'
```

Output

Faculty_code	Faculty_Name
100	Yogesh
102	Omprakash
104	Mahesh

Pattern 3 : Determine a string contains given a letter.

Find all faculties whose name contains letter 'a'.

Syntax

```
SELECT      *
FROM        Faculty
WHERE       Faculty_Name LIKE '%a%'
```

Output

Faculty_code	Faculty_Name
102	Omprakash
104	Mahesh
101	Amit

Pattern 4 : Determine a string in which given pattern is combination of above forms.

Find all faculties whose names second letter is 'a' contains letter 'e' and name ending with letter 'h'.

Syntax

```
SELECT  *
FROM    Faculty
WHERE   Faculty_Name LIKE '_a%e%h'
```

Output

Faculty_code	Faculty_Name
104	Mahesh

Pattern 5 : Determine a String in which given a letter is present at particular position.

Find all faculties whose names second letter is 'a'.

Syntax

```
SELECT  *
FROM    Faculty
WHERE   Faculty_Name LIKE '_a%'
```

Output

Faculty_code	Faculty_Name
104	Mahesh

Find all faculties whose names second last letter is's'.

Syntax

```
SELECT  *
FROM    Faculty
WHERE   Faculty_Name LIKE '%s_'
```

Output

Faculty_code	Faculty_Name
100	Yogesh
102	Omprakash
104	Mahesh

Pattern 6 : Determine a string not contains given a letter.

Find all faculties whose name do not contains 'a'.

Syntax

```
SELECT  *
FROM    Faculty
WHERE   Faculty_Name NOT LIKE '%a%'
```

Output

Faculty_code	Faculty_Name
100	Yogesh
103	Nitin

Use of escape character in pattern matching :

- Escape characters can be used for some queries like if we want to look for '%' sign in a given column.
- Use the **escape** clause to specify an escape character in the **like** clause.
- An escape character must be a single character string.
- Any character in the server's default character set can be used. (_ , %)
- Specifying more than one escape character raises a SQLSTATE error condition

```
LIKE "%XX_%" escape "XX"
```

Example

Determine a String contains '%' sign. (Use of escape character)

Consider following symbol table

S_code	Symbol	Name
1	%	Percentage
2	*	Astrik

Find details of symbol '%'.

Syntax

```
SELECT  *
FROM    Symbol
WHERE   Symbol LIKE '%\%%' ESCAPE '\'
```

Output

S_code	Symbol	Name
1	%	Percentage
2	*	Astrik

In above case if you want to search for '%' sign then, we will go for above pattern. Above pattern will ignores '/' symbol (as specified in ESCAPE clause) and only finds string with symbol '%'.

7.12 Limit Clause – Restricting Rows in Result

- The LIMIT clause is used to constrain the number of rows returned by the SELECT statement.
- LIMIT takes one or two numeric arguments, which must both be non-negative integer constants.
- This clause operates just before printing result, it means WHERE and ORDER BY clauses operates before LIMIT clause.

Syntax

```
SELECT      *
FROM    Table_Name
[LIMIT number]
```

Examples

Select top 3 departments from department table.

Syntax

```
SELECT    *
FROM    departments
Order by dept_id
LIMIT    3;
```

Output

DEPT_ID	DEPT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
50	Shipping	124	1500

Select 2nd and 3rd departments from department table.

Syntax

```
SELECT    *
FROM    departments
LIMIT 1, 2; /* Retrieve row number 2 and 3 */
```

Output

DEPT_ID	DEPT_NAME	MANAGER_ID	LOCATION_ID
20	Marketing	201	1800
50	Shipping	124	1500

7.13 Transaction Control Language (TCL) – COMMIT, ROLLBACK, SAVEPOINT

(MSBTE – S-14, W-14, S-15,S-16, S-17)

Q. Describe savepoint and rollback commands with example. **S-14, W-14, 4 Marks**

Q. How to use commit, savepoint, rollback commands. **S-15, S-16, S-17, 4 Marks**

Q. Describe commit and rollback with syntax and example. **W-18, 4 Marks**

- Any SQL query can be executed with two basic operations on the database objects :
  1. Read
  2. Write
- After executing SQL query, we must specify its final action as commit (save data) or abort (or revert back changes).
- The COMMIT statement ends the operations and makes all changes made to the data permanent, on successful completion.
- ABORT terminates and undoes all the actions done so far.

7.13.1 Commit Transaction

- A query that is successful and has encountered no errors is committed by issuing commit. That is, all changes to the database are made permanent and become visible to other users of the database.
- The syntax is as follows :

```
COMMIT [WORK]
```

- The keyword WORK is not required, though it might be added for clarity; a simple COMMIT is usually all that is required.
- Microsoft SQL Server 2000 does support the SQL99 standard syntax - in addition to its own. The Microsoft syntax allows for committing named transaction whereas the standard one does not.

```
COMMIT [TRANSACTION] [<transaction name>]
```

- As you can see, only COMMIT is required, everything else is optional and the keywords can be shortened (i.e., TRAN instead of TRANSACTION). Alternatively COMMIT WORK can be used.

7.13.2 Rollback Transaction

- A query that is unsuccessful and has encountered some type of error should be rolled back. That is, all changes to the database are undone and the database remains unchanged by the transaction.
- Transaction-processing systems ensure database integrity by recording intermediate states of the database as it is modified, then using these records to restore the database to a known state if a transaction cannot be committed.

Syntax

```
ROLLBACK
```

7.13.3 Savepoint Transaction

- Transaction-processing systems ensure database integrity by recording intermediate states of the database as it is modified, by using SAVEPOINT at regular intervals so as to whenever transaction failure happens it can be partially rolled back instead of rolling back entire transaction like in above case
- The following example illustrates the partial ROLLBACK statement.

```
DECLARE
Var_sal Emp.sal%TYPE;
BEGIN
SELECT sal INTO var_sal
FROM emp
where empno = 7844;
DBMS_OUTPUT.PUT_LINE ('Salary : '||var_sal||
'(Original)');
Update emp set sal = sal*1.1;
-- Increment salary by 10% (sal + 0.1*sal)
SELECT sal INTO var_sal
FROM emp
where empno = 7844;
DBMS_OUTPUT.PUT_LINE('Salary : '||var_sal||'(Before
Savepoint A)');
SAVEPOINT A;
Update emp set sal = sal*0.8;
-- Decrement salary by 20% (sal - 0.2*sal)
SELECT sal INTO var_sal
FROM emp
where empno = 7844;
DBMS_OUTPUT.PUT_LINE('Salary : '||var_sal||'(Savepoint A)
');
SAVEPOINT B;
Update emp set sal = sal*1.3;
-- Increment salary by 30%
SELECT sal INTO var_sal
FROM emp
where empno = 7844;
DBMS_OUTPUT.PUT_LINE('Salary : '||var_sal||'(Savepoint B)
');
ROLLBACK to savepoint B;
SELECT sal INTO var_sal
FROM emp
where empno = 7844;
DBMS_OUTPUT.PUT_LINE('Salary : '||var_sal||'(Rollback B)
');
END;
```

Output

```
SQL> /
Salary : 1500(Original)
Salary : 1650(Before Savepoint A)
Salary : 1320(Savepoint A)
Salary : 1716(Savepoint B)
Salary : 1320(Rollback B)
```

- Whenever last ROLLBACK is executed, Entire database will not be re stored to its original consistent state but it will restore to state before SAVEPOINT B.
- So, this can be used for rollback database partially.

7.13.4 Commit and Rollback Operation

Q. How to perform commit and rollback operation give example to support your answer.

1. The COMMIT Command :

- Changes made to the database by INSERT, UPDATE and DELETE commands are temporary until explicitly committed. This is performed by the COMMIT command. The syntax for COMMIT command is as follows :

  COMMIT;

- **Example :** Consider the CUSTOMERS table having the following records :

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	2000.00
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000.00
4	Chaitali	27	Mumbai	6500.00

- Following is the example which would delete records from the table having age = 25 and then COMMIT the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
WHERE AGE = 25;
SQL> COMMIT;
```

- As a result, two rows from the table would be deleted and SELECT statement would produce the following result :

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	2000.00
3	kaushik	23	Kota	2000.00
4	Chaitali	27	Mumbai	6500.00

2. The ROLLBACK Command

1. The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database.
2. The ROLLBACK command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued..The syntax for ROLLBACK command is as follows :

   ROLLBACK;

Example

Consider the CUSTOMERS table having the following records :

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	2000.00
2	Khilan	25	Delhi	1500
3	Kaushik	23	Kota	2000.00
4	Chaitali	27	Mumbai	6500.00

Following is the example, which would delete records from the table having age = 25 and then ROLLBACK the changes in the database.

```
SQL> DELETE FROM CUSTOMERS
WHERE AGE = 25;
SQL> ROLLBACK;
```

As a result, delete operation would not impact the table and SELECT statement would produce the

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Pune	2000.00
2	Khilan	25	Delhi	1500
3	kaushik	23	Kota	2000.00
4	Chaitali	27	Mumbai	6500.00

7.14 Data Control Language (DCL)

Q. Write a note on DCL.

- **Data Control Language (DCL)** is used to control various user actions like inserting data, updating data, deleting data or viewing data.
- To perform any operation in the database user needs **privileges** like creating tables, index or views.
- DCL is set of commands used to,
  - **Grant :** Gives some privilege to user for performing task on database.
  - **Revoke :** Take back permissions given to user.

7.14.1 Grant Command

Q. Write syntax for GRANT privileges?

- A system privilege is the right to perform a particular action or to perform an action on any schema objects of a particular type.
- An authorized user may pass on this authorization to other users. This process is called as granting of privileges

- Generally GRANT statement is used by owner of table or view to give other users access permissions.
- In SQL user accounts must present in system before we can grant privileges to him.

Syntax

```
Grant <ALL | privilege list>
ON <relation name or view name>
TO <user | role list | PUBLIC >,
[WITH GRANT OPTION]
```

Privilege list	Meaning
ALTER	Table and views
CREATE	Table and views
DROP	Table and views
DELETE	Tables and views
INSERT	Tables and views
SELECT	Tables and views
UPDATE	Tables and views
ALL	Tables and views

WITH GRANT OPTION

Is used to allow user to grant privileges (which are granted to him) to other users.

Example

- Consider an example for granting update authorization to the *Emp_Salary* relation of the company database. Assume that, initially that the DBA grants update authorization on *Emp_Salary* to other users U1, U2 and U3, who may in turn pass on this authorization to other users. This passing of authorization from one user to other users is called **authorization graph.**
- The following **grant** statement grants user U1, U2 and U3 the **select** privilege on *Emp_Salary* relation :

```
GRANT SELECT, INSERT
ON mydb.*
TO 'mahesh'@'somehost';
```

7.14.2 Revoking Command

Q. Write a short note on Revoking of privileges.

- We can reject the privileges given to particular user with help of revoke statement.
- To revoke an authorization, we use the **revoke** statement.

Syntax

```
REVOKE <ALL | privilege list>
ON <relation name or view name>
FROM <user | role list | PUBLIC>
[RESTRICT/ CASCADE]
```

- **CASCADE** : Will revoke all privileges along with all dependent grant privileges
- **RESTRICT** : This will not revoke all related grants only removes that GRANT only.
- **Examples**

The revocation of privileges from user or role may cause other user or roles also have to leave that privilege. This behaviour is called cascading of the revoke.

To remove select privilege from users U1, U2 and U3.

```
REVOKE select
ON mydb.mytbl
FROM 'mahesh'@'somehost';
```

7.15 Solved Problems

Ex. 7.15.1 : Consider table students_academics_ details with the following columns.

Rollno, Student_name, Class, Division, Faculty, Marks_subject1, Marks_subject2, Marks_subject3.

Fire following queries

1. Find average marks in each subject faculty wise.
2. Find the number of students in each division of each class.
3. Find sum of all the m1, m2, m3 subject marks.

4. Display the class and the number of students where no > 3
5. Find minimum, maximum and average marks of m1, m2, m3 subject.

Soln. :

1. Find average marks in each subject faculty wise.

```
SELECT Faculty,avg(Marks_subjectl),
avg(Marks_subject2), avg(Marks_subject3)
FROM students_academics_details
GROUP BY Faculty
```

2. Find the number of students in each division of each class.

```
SELECT class,division, count(distinct Roll_no)
FROM students_academics_details
GROUP BY class,division;
```

3. Find sum of all the ml, m2, m3 subject marks.

```
SELECT sum(Marks_subjectl), sum(Marks_subject2),
sum(Marks_subject3)
FROM students_academics_details
```

4. Display the class and the number of students where no > 3

```
SELECT class, count(Roll_no)
FROM Students_academics_details
GROUP BY class
Having count(Roll_no)>3;
```

5. Find minimum, maximum and average marks of m1, m2, m3 subject.

```
SELECT min(Marks_subjectl), max(Marks_subjectl),
avg(Marks_subjectl),
min(Marks_subject2), max(Marks_subject2),
avg(Marks_subject2),
min(Marks_subject3), max(Marks_subject3),
avg(Marks_subject3)
FROM students_academics_details
GROUP BY subject;
```

Ex. 7.15.2 : Consider the following table :

Id	name	Work_date	Daily_typing_pages
1	John	2007-01-24	250
2	Ram	2007-05-27	220
3	Jack	2007-05-06	170
3	Jack	2007-04-06	100
4	Jill	2007-04-06	220
5	Zara	2007-06-06	300
5	Zara	2007-02-06	350

Answer following queries :

a) Find the count for a name Zara

b) calculate total of all the dialy typing _pages

c) sum up all the records related to a single person

d) calculate square root of all the dialy_typing_pages

e) fetch year from database

Soln. :

(a) Find the count for a name Zara

```
SELECT Count(*)
FROM Employee
WHERE name LIKE 'Zara'
```

(b) Calculate total of all the dialy typing _pages

```
SELECT Sum(daily_typing_pages)
FROM Employee
```

(c) Sum up all the records related to a single person

```
SELECT
id,name,daily_typing_pages,SUM(daily_typing_pages)
FROM Employee GROUP BY id;
```

(d) Calculate square root of all the dialy_typing_pages

```
SELECT name ,sqrt(daily_typing_pages)
FROM Employee
```

(e) Fetch year from database

```
EXTRACT(YEAR FROM work_date)
FROM Employee
```

Ex.7.15.3 : Consider the following database Employee(emp_id, emp_name, emp_city, emp_addr, emp_dept, join_date)

i) Display the emp_id, of employee who live in city 'Pune' or'Nagpur'.

ii) Change employee name, 'Aayush' to'Aayan'.

iii) Display the total number of employee whose dept is 50. W-18. 6 Marks

Soln.:

```
(i)  Select emp_id;
     From Employee
     Where City in ('Pune', 'Nagpur')
(ii) Update employee
     Set Name emp_name
     Where Aayan
(iii) Select count(*)
     From employee
     Where emp_dept = 50
```

Review Questions

Q. 1 What are the role of SQL with example.

Q. 2 Explain various data types used in SQL.

Q. 3 How to create table employee (Eid, Ename, age, dateofbirth, address).

Q. 4 Write short note on : DDL and DML.

Q. 5 What is SQL? Explain the following structures of SQL queries with appropriate example.

(i) Select clause

(ii) Where clause

Q. 6 Explain DDL commands with example.

Q. 7 Explain DML commands with example.

Q. 8 Explain TCL commands with example.

Q. 9 Explain DCL commands with example.

Q. 10 Write syntax for REVOKE with suitable example.

Q. 11 Write syntax for GRANT privileges.

Q. 12 Write a short note on Revoking- of privileges.

❑❑❑

CHAPTER 8

UNIT III

Interactive SQL

Syllabus

In-built Functions : String, Arithmetic,

Date and Time, Aggregate Functions

8.1 Inbuilt Functions in MySQL

(MSBTE - W-13, S-14, S-16, W-16, S-17, W-17, W-18)

Q. Explain any four string functions with examples. W-13, S-14, S-16, W-16, S-17, W-17, W-18, 4 Marks

Q. Explain various inbuilt functions in MySQL.

Function accepts multiple inputs arguments and after processing it returns only one output result value.

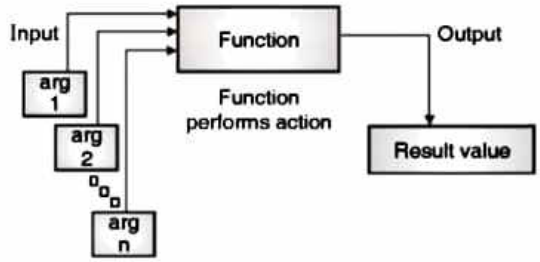


Fig. 8.1.1 : Concept of Functions

Single Row SQL Built In Functions

STRING Functions		MATH Functions		Date
Case Manipulation	Character Manipulation	Manipulation	Computation	Functions
LCASE / LOWER	CONCATE	ABS	MOD	NOW
UCASE / UPPER	INSTR	CEIL	POW	ADDDATE
-	LEFT	FLOOR	SQRT	DATEDIFF

STRING Functions		MATH Functions		Date
Case Manipulation	Character Manipulation	Manipulation	Computation	Functions
-	RIGHT	ROUND	-	DAY
-	MID/SUBSTR	TRUNCATE	-	MONTH
-	LENGTH	-	-	YEAR
-	REPLACE	-	-	HOUR
-	STRCMP	-	-	MIN
-	TRIM	-	-	SEC
-	LTRIM	-	-	REVERSE
-	RTRIM	-	-	-
-	REVERSE	-	-	-

8.2 String Functions

Q. Write a short note on: String function.

- MySQL string functions are used to manipulate string data and derive some information and analysis from the tables.
- String functions can be used with update commands to change records or to update multiple tables.
- These functions can be applied to column having data type CHAR, VARCHAR and VARCHAR2 to change its type.

8.2.1 Case Manipulation Functions

Q. Write a note on case manipulation functions.

- These functions can be used to show display as per required format.
- To change case of any column having data type CHAR.

Functions	Result
LOWER ('PLSQL') LCASE ('PLSQL')	plsql
UPPER ('PLSQL') UCASE ('PLSQL')	PLSQL

Examples :

Syntax 1

```
SELECT Lower ('NAME') "Lower case";\
```

Output

Lower case
name

Syntax 2

```
SELECT Lower (ename) "Lower case name"
FROM Emp;
```

Output

Lower case name
Mahesh
Bhavna
Reshma
Simi

Syntax 3

```
SELECT Upper (ename) "Lower Case name"
FROM Emp;
```

Output

Lower Case name
MAHESH
BHAVNA
RESHMA
SIMI

8.2.2 Character Manipulation Functions

- The function will produce string data as per required manipulation or string operations.
- To apply some operations to particular column of character type

```
SELECT ('Hello', 'World') "Test Column"
```

Output

Test Column
HelloWorld

Some more Operations are as follows,

Function	Result	Description
SUBSTRING('HelloWorld',1,5) MID('HelloWorld',1,5)	Hello	Cuts the string from starting position given to last positions. SUBSTRING(*string*,<start>,<end>)
INSTR('HelloWorld', 'W')	6	Finds location of given character
LENGTH('HelloWorld')	10	Counts the length of string
REVERSE('HelloWorld')	dlroWolleH	Prints given string in reverse order.
LEFT('HelloWorld',5)	Hello	Trims the string from left side by number of characters required.
RIGHT('HelloWorld',5)	World	Trims the string from right side by number of characters required.
STUFF('HelloWorld', 2, 3, '_')	Hello_World	Adds the character give between the positions mentioned in statement.

Function	Result	Description
RTRIM('HelloWorld ')	HelloWorld	Removes all trailing white spaces.
LTRIM(' HelloWorld')	HelloWorld	Removes all leading white spaces.
TRIM(' HelloWorld ')	HelloWorld	Removes all white spaces.
CONCATE('Hello', 'World')	HelloWorld	Connects given strings together
REPLACE('Helo World','l','ll')		
REVERSE('ello')	olle	Finds the reverse of string
STRCMP('Hello', 'Hello') STRCMP('Hello', 'Myworld') STRCMP('Hello', 'Word')	0 -1 1	Compares two strings output is, = 0 if both strings are equal = -1 if first string smaller than second =1 otherwise

Examples

```
mysql> SELECT REPLACE('Helo World', 'l', 'll');
+-------------------------------------------------+
| REPLACE('Helo World', 'l', 'll');               |
+-------------------------------------------------+
| Hello World                                     |
+-------------------------------------------------+
1 row in set (0.00 sec)
mysql> SELECT RIGHT('Hello World', 5);
+-------------------------------------------------+
|    RIGHT('Hello World', 4)                      |
+-------------------------------------------------+
| World                                           |
+-------------------------------------------------+
1 row in set (0.00 sec)
mysql> SELECT SUBSTRING('Hello World' FROM 7);
+-------------------------------------------------+
| SUBSTRING ('Hello World' FROM 7)                |
+-------------------------------------------------+
| World                                           |
+-------------------------------------------------+
1 row in set (0.00 sec)
```

8.3 Arithmetic Functions

Q. Explain SQL arithmetic functions: 1. ABS 2. TRUNCATE.

These functions can be applied to columns with data type **number or numeric type.**

1. ROUND

- Rounds value to specified decimal
- Increasing value of previous digit by 1 if last digit is equal to or below 5.
- If last digit is below value 5, don't change value of previous digit.

Function	Result
ROUND(45.926, 2)	45.93
ROUND (45.924, 2)	45.92

```
mysql> SELECT ROUND(45.84545,2);
    -> 45.85
```

2. CEILING

- Truncates value to specified decimal without changing previous digits.
- CEILING(X) function will returns the smallest integer value not less than X.

Function	Result
CEILING(45.8)	46
CEILING (45.1)	46

```
mysql> SELECT CEILING(45.8);
    -> 46
```

3. FLOOR

- Returns remainder of division.

– FLOOR(X) function will returns the largest integer value not greater than X.

Function	Result
FLOOR(45.8)	45
FLOOR (45.1)	45

```
mysql> SELECT FLOOR(1.22);
  -> 1
```

4. ABS

Returns the absolute value of number.

Function	Result
ABS(22)	22
ABS (-3)	3

```
mysql> SELECT ABS(-5);
  -> 5
```

5. TRUNCATE

– Truncates value to specified decimal by ignoring decimal values.

– Returns the number X, truncated to D decimal places as specified.

– If D is 0, the result will not contain any decimal point.

– D can be negative to cause D digits left of the decimal point of the value X to become zero.

Function	Result
TRUNCATE(45.8145,2)	45.81
TRUNCATE(45.8145,-1)	40

```
mysql> SELECT TRUNCATE(1.22665,2);
  -> 1.22
```

6. MOD (N, M) or N%M

Function will returns the remainder of N divided by number M.

Function	Result
MOD(4,2)	0
MOD(45,10)	5

```
mysql> SELECT MOD(4589,100);
  -> 89
```

7. POW (N, M)

Returns the value of N raised to the power of M.

Function	Result
POW (4,2)	16
POW(2,-2)	0.25

```
mysql> SELECT POW(2,2);
  -> 4
```

8. SQRT (N)

Returns the square root of a non-negative number N.

Function	Result
SQRT(4)	2
SQRT (-16)	Null

```
mysql> SELECT SQRT(16);
  -> 4
```

8.4 Date Time Functions

(MSBTE - W-15, W-16, S-17)

Q. Explain any four date function with example. W-15, W-16, S-17, 4 Marks

Q. Write about SQL Date Time functions.

– These functions can be applied to columns with data type Date and Time.

– We can perform some data and time operations on such columns.

– To apply some operations to particular column of date type.

NOW()

This function will returns today's date and time.

```
mysql> SELECT Now();
```

Today
2016-05-29 05:25:51

To apply some operations are as given below,

Function	Result
NOW()	2016-05-29 05:25:51
ADDDATE(NOW(),1)	2016-05-30 05:25:51
DATEDIFF (NOW (),NOW ())	0
DAY(NOW ())	29
MONTH(NOW ())	5
YEAR(NOW ())	2016
HOUR(NOW ())	5
MIN(NOW ())	25
SEC(NOW ())	51

```
mysql> SELECT NOW() "TODAY";
+---------------------+
| TODAY               |
+---------------------+
| 2016-05-29 05:25:51 |
+---------------------+
1 row in set (0.00 sec)
mysql> SELECT YEAR(NOW()) "TODAY";
+-------+
| TODAY |
+-------+
| 2016  |
+-------+
1 row in set (0.00 sec)
```

8.5 Aggregate Functions

(MSBTE - W-18)

Q. What is mean by aggregate function ?

Q. Enlist four aggregate functions. **W-18, 2 Marks**

1. Introduction

- Sometimes for decision making we need summarize data from table like average, sum, minimum etc.
- SQL provides various aggregate functions which can summarize data of given table.The function operates on the table data produces a single output.
- Such queries are generally used for producing reports and summary forms in an application.

2. Types of aggregate functions

a) **COUNT ([DISTINCT] C) :** The number of (unique) values in the column C.

b) **SUM ([DISTINCT] C) :** The sum of all (unique) values in the column C.

c) **AVG ([DISTINCT] C) :** The average of all (unique) values in the column C.

d) **MIN (C) :** The minimum value in the column C.

e) **MAX (C) :** The maximum value in the column C.

3. Example

Table 8.5.1: Exam_Marks

Sid	SName	Marks
1	Mahesh	90
2	Suhas	80
3	Jyendra	89
4	Sachin	99
5	Vishal	88
6	Payal	90

8.5.1 Types of Aggregate Functions

(MSBTE - W-13, S-14)

Q. Explain any four Aggregate functions with example. **W-13, S-14, 4 Marks**

a) COUNT ()

- This function is used to calculate number of rows (or records) in a table selected by query.
- COUNT returns the number of rows in the table when the column value is not NULL.
- Column in the query must be numeric.

For Examples :

Example 1 : Find total number of students in Table 8.5.1

Syntax

```
SELECT Count(Sid) as Count
FROM   Exam_Marks
```

Output

COUNT
6

Let us consider, Table 8.5.1 modified as,

(Some duplicate rows are present in Table 8.5.2).

Table 8.5.2

Sid	SName	Marks
1	Mahesh	90
1	Mahesh	90
2	Suhas	80
3	Jyendra	89
3	Jyendra	89
4	Sachin	99
5	Vishal	88
6	Payal	90

In this case Query 1 results to wrong result as below,

Syntax

```
SELECT  Count(Sid) as Count
FROM    Exam_Marks
```

Output

COUNT
8

So we can modify query as given below.

Example 2 : Find total number of different students in Table 8.5.2.

Syntax

```
SELECT  Count(Distinct Sid) as Count
FROM    Exam_Marks
```

Output

COUNT
6

(b) SUM ()

- This function is used to calculate sum of column values in a table selected by query.
- Column in the query must be numeric.
- Value of the sum must be within the range of that data type.

For Example : Find total of marks scored by all students.

Syntax

```
SELECT  SUM(Marks) as Sum
FROM    Exam_Marks
```

Output

SUM
446

(c) AVG ()

- This function is used to calculate Average of column values in a table selected by query.
- This function first calculates sum of column and then divide by total number of rows.
- AVG returns the average of all the values in the specified column.
- Column in the query must be numeric.

Example

Find average marks of students.

Syntax

```
SELECT  AVG(Marks) as AVG
FROM    Exam_Marks
```

Output

AVG
89.33

(d) MIN()

- This function is used to find minimum value out of column values in a table selected by query.
- Column in the query need not be numeric data type.

Example

Find minimum marks scored by students.

Syntax

```
SELECT  MIN(Marks) as Min
FROM    Exam_Marks
```

Output

MIN
80

(e) MAX()

- This function is used to find maximum value out of column values in a table selected by query.
- Column in the query need not be numeric data type.

Example

Find maximum marks scored by students.

Syntax

```
SELECT  MAX(Marks) as Max
FROM    Exam_Marks
```

Output

MAX
80

8.5.2 Summary of Aggregate Functions

Q. Compare various Aggregate functions.

Function	Description
AVG ([DISTINCT | ALL] n)	Average value of n, ignoring null values.
COUNT ({ * | [DISTINCT | ALL] expr))	Number of rows, where *expr* evaluates to something other than null.
MAX ([DISTINCT | ALL] expr)	Maximum value of *expr*, ignoring null values.
MIN ([DISTINCT | ALL] expr)	Minimum value of *expr*, ignoring null values.
SUM ([DISTINCT | ALL] n)	Sum value of n, ignoring null values.

Review Questions

Q. 1 Explain various inbuilt functions in SQL.

Q. 2 Short note on String functions.

Q. 3 Explain Arithmetic function :

1. ABS
2. TRUNCATE

Q. 4 Explain data time function.

Q. 5 Short note on aggregate function with example.

Q. 6 Explain TRUNCATE function.

Q. 7 Explain ABS function.

❑❑❑

CHAPTER 9

UNIT III

Advanced SQL

Syllabus

Queries Using Group By, Having and Order by Clause, Joins – Inner and Outer Joins, Sub Queries.

9.1 Ordering using Order By Clause

(MSBTE -W-13, W-14, S-15, W-17)

Q. Explain order by clause with suitable example. **W-13, W-14, S-15, W-17, 2 Marks**

(1) Introduction

- The ORDER BY keyword is used to sort the result-set by a specified column.
- The ORDER BY keyword sorts the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.

(2) Syntax

```
SELECT      column_name
FROM        table_name
ORDER BY    column_name ASC/DESC
```

(3) Example

Table 9.1.1 : Faculty table

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Yogesh	17/07/64	DSA	16
101	Amit	24/12/72	MIS	16
102	Omprakash	03/02/80	PWRC	8
103	Nitin	28/11/66	DT	10
104	Mahesh	01/01/86	DT	10

- Now we want to select all the Faculties from the Table 9.1.1, however, we want to sort the Faculty by their Faculty_Name.
- We use the following SELECT statement :

```
SELECT      *
FROM        Faculty
ORDER BY    Faculty_Name
```

- In second query ORDER BY 2 indicates order by second column of table.

Output

The result-set will look like this.

Faculty_code	Faculty_Name	DOB	Subject	Hours
101	Amit	24/12/72	MIS	16
104	Mahesh	01/01/86	DT	10
103	Nitin	28/11/66	DT	10
102	Omprakash	03/02/80	PWRC	8
100	Yogesh	17/07/64	DSA	16

(4) Example

- Now we want to select all the Faculties from the Table 9.1.1, however, we want to sort the Faculties descending by their Faculty_Name.
- We use the following SELECT statement :

```
SELECT      *
FROM        Faculty
ORDER BY    Faculty_Name DESC
```

Output

The result-set will look like

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Yogesh	17/07/64	DSA	16
102	Omprakash	03/02/80	PWRC	8
103	Nitin	28/11/66	DT	10
104	Mahesh	01/01/86	DT	10
101	Amit	24/12/72	MIS	16

9.2 GROUP BY Clause - Grouping Query Results

(MSBTE – W-13, W-14, S-15, S-16, S-17, W-17)

Q. Explain group by clause. **W-13, 2 Marks**

Q. Explain group by clause with suitable example. **W-14, S-15, S-16, W-17, 4 Marks**

Q. Define group by clause. **S-17, 2 Marks**

(1) Introduction

- Related rows can be grouped together by **GROUP BY** clause based on distinct values that exist for specified columns.
- A GROUP BY clause creates a set of data, containing several sets of records grouped together based on some condition.
- The GROUP BY statement is used with the SQL aggregate functions to group the retrieved data by one or more columns or expression.
- The ORDER BY keyword is used to sort the result-set by a specified column.
- The ORDER BY keyword sorts the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.
- The GROUP BY column does not have to be in the SELECT clause.
- A Grouping Query groups rows based on common values in a set of grouping columns. Rows with the same values for the grouping columns are placed in distinct groups. Each group is treated as a single row in the query result.

(2) Syntax

```
SELECT      column_name
FROM        table_name
GROUP BY    column_name
```

(3) Example

The **Student_Dept** table is as follows :

Table 9.2.1

Student_Dept		
Sid	**SName**	**Dept**
1	Mahesh	IT
2	Anu	IT
3	Suhas	CE
4	Jyendra	CE
5	Umang	EXTC
6	Amruta	EXTC
7	Rahul	EXTC
8	Hiral	IT

Retrieve number of students in various departments.

Syntax

```
SELECT      Dept, count (Sid)"Total Student"
FROM        Student_Dept
GROUP BY    Dept;
```

Output

Dept	Total Student
IT	3
EXTC	2
CE	3

You can use the group function in the ORDER BY clause.

Syntax

```
SELECT      Dept, count (*) "Total Student"
FROM        Student_Dept
GROUP BY    Dept;
```

Output

Dept	Total Student
CE	3
EXTC	2
IT	3

9.3 HAVING Clause – Filtering grouped Query Results

9.3.1 Apply Conditions with GROUP BY

(MSBTE - W-13, S-16, S-17)

Q. Explain having clause with example. **W-13, S-16, S-17, 4 Marks**

(1) WHERE clause

- A Grouping Query can also group rows based on common values in a set of grouping columns and with specified condition.
- The WHERE clause specifies the rows to be retrieved. Since there is no WHERE clause, all rows are retrieved by default.
- Retrieve customer information whose purchase amount exceeds grouped by Customer id.

Syntax

```
SELECT      *
FROM        Customer_Loan
WHERE       Amount<4000
GROUP BY    Cust_id ;
```

Output

Cust_id	Loan_no	Amount
103	2010	2555.00
103	2015	2000.00
104	2056	3050.00

(2) HAVING clause

- HAVING is conditional clause which checks data for specific search condition.
- A HAVING clause is like a WHERE clause, but applicable only to groups as a whole (that is, to the rows in the result set representing groups), whereas the WHERE clause applies to individual rows.
- A query can contain both a WHERE clause and a HAVING clause.

Syntax

```
SELECT      *
FROM        Customer_Loan
GROUP BY    Cust_id
HAVING      Cust_id >103 ;
```

Output

Cust_id	Loan_no	Amount
104	2056	3050.00

Difference in operation of WHERE and GROUP BY clause

Execution hierarchy

- The WHERE clause is applied first to the individual rows in the result set then rows that meet the conditions in the WHERE clause are grouped using group by clause.
- The HAVING clause is then applied to the rows in the result set that is produced by grouping.

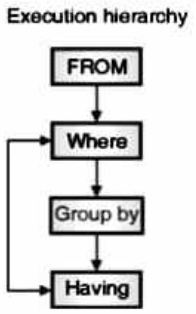


Fig. 9.3.1

- Only the groups that meet the HAVING conditions appear in the query output. You can apply a HAVING clause only to columns that also appear in the GROUP BY clause or in an aggregate function.
- The **Student_Dept** table is as follows :

Sid	SName	Dept
1	Mahesh	IT
2	Anu	IT

Sid	SName	Dept
3	Suhas	CE
4	Jyendra	CE
5	Umang	EXTC
6	Amruta	EXTC
7	Rahul	EXTC
8	Hiral	IT

Retrieve departments HAVING more than two students in it.

Syntax

```
SELECT      Dept, count (Sid) "Total Student"
FROM        Student_Dept
GROUP BY    Dept
HAVING      count (Sid)>2;
```

Output

Dept	Total Student
IT	3
EXTC	2
CE	3

9.4 GROUP BY and Aggregate Function

(1) Introduction

- **GROUP BY** clauses is very useful in combination with aggregate functions.
- Aggregate functions (like **SUM**, **AVG**) return the aggregate of all column values every time they are called.
- Using **GROUP BY** it is possible to find the function value for each individual group of column values.

(2) Syntax

```
SELECT      column_name1,
            Aggregate_Function (Expression)
FROM        table
WHERE       row_Qualifier_Condition
GROUP BY    column_name
HAVING      group_Qualifier_Condition;
```

(3) Example

Retrieve total loan_amount for all the loans taken by each Customer of a bank.

Syntax

```
SELECT      Cust_id, SUM (Amount)
FROM        Customer_Loan
GROUP BY    Cust_id ;
```

Output

Cust_id	Amount
101	8755.00
103	4555.00
104	3050.00

(4) Nested group functions

- Group functions can be nested to a depth of two.
- **Example:** Display the maximum of average salary offered in all department.

Syntax

```
SELECT      MAX (AVG (salary))
FROM        employees
GROUP BY    department_id
```

Output

MAX(AVG(salary))
60000

9.5 SQL Joins Concept

(MSBTE - W-18)

Q. Explain joins in SQL with examples. **W-18, 4 Marks**

- The relational operations can merge columns from two different tables to form new join table.

A	B	C

D	E

A	B	C	D	E

Fig. 9.5.1 : Joining operation

- We can join multiple tables with help of some join condition (Equality or Inequality) or without any join condition (cross join).

9.5.1 Cartesian product / Cross join

Q. What is Cartesian product?

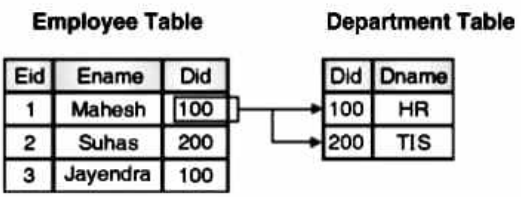
- A cross join performs relational product or Cartesian product of two tables specified in query.
- In DBMS, CROSS join occur due to WHERE condition is missing in query or some invalid operations in where clause leads to undesired results or CROSS join.
- A cartesian product is formed when :
  - A join condition is omitted.
  - A join condition is invalid.
  - All rows in the first table are joined to all rows in the second table.
- To avoid a cartesian product, always include a valid join condition in a WHERE clause.

Syntax

```
SELECT      Column_List
FROM        Table1
CROSS JOIN  Table2
```

Example

- This query automatically selects common column and join tables based on that column. Hence, no need to specify name of common column explicitly,
- But joining table must have common column with same name otherwise error will occur.



Syntax

```
SELECT      e.Eid [Eid],
            e.Ename [Ename],
            e.Did [Did],
            d.Did [Did],
            d.Dname [DName]
FROM        Employee e
CROSS JOIN  Department d
```

Output

Eid	Ename	Did	Did	Dname
1	Mahesh	100	100	HR
1	Mahesh	100	200	TIS
2	Suhas	200	100	HR
2	Suhas	200	200	TIS
3	Jayendra	100	100	HR
3	Jayendra	100	200	TIS

9.5.2 Inner join

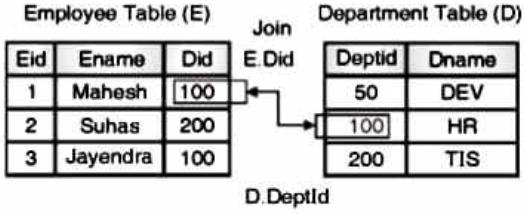
Q. What is inner join?

- In many cases, tables are joined according to search conditions that find only the rows with matching values; this type of join is known as an inner equijoin.
- Inner join joins two table when there is at least one match between two tables.
- Inner join works as simple joins.

Syntax

```
SELECT      Column_List
FROM        Table1
INNER JOIN  Table2
ON          (JOIN_Condition)
OR
SELECT      Column_List
FROM        Table1
INNER JOIN  Table2
USING       (Common_Column_Name)
```

Example



Syntax

```
SELECT      E.Eid [Eid], E.Did [Did],
            D.Dname [DName]
FROM        Employee E
INNER JOIN  Department D
ON          E.Did = D.DeptId
```

Output

Eid	Did	DName
1	100	HR
2	200	TIS
3	100	HR

Case 1 : Natural join

Q. What is natural join? Give its syntax.

- A natural join returns all rows by matching values in common columns having same name and data type of columns and that column should be present in both tables.
- Natural join eliminates duplicate columns present in JOIN table. Therefore common column will be printed only once in resultant table.
- In this type of join, joining tables must have at least one common column between them which is also has same column name.
- **Working :** In this type of join processor looks for a column name which is present in both table than matches data types if match is found than these tables will be joined using common column name.

Syntax

```
SELECT          Column_List
FROM            Table1
NATURAL JOIN    Table2
```

Example

This query automatically selects common column and join tables based on that column. Hence, no need to specify name of common column explicitly, but joining table must have common column with same name <and same data type> otherwise error will occur.

Employee Table

Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	100

Department Table

Did	Dname
100	HR
200	TIS

Syntax

```
SELECT          E.Eid [Eid], D.Did [Did], D.Dname [Dname]
FROM            Employee E
NATURAL JOIN    Department D
```

Output

Eid	Did	Dname
1	100	HR
2	200	TIS
3	100	HR

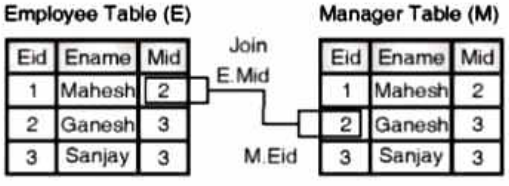
Case 2 : Self join

Q. What is self join ?

- The tables being joined in a query do not need to be distinct; you can join any table to itself as long as you give each table reference a different name using table alias.
- Self-joins are useful for discovering relationships between different columns of data in the same table.
- In this type of join query we must use rename operation or table alias.

Example

This query joins the Employee table to itself over the Eid (Employee ID) and Mid (Manager Id) column, using the table aliases E and M to distinguish the table references.



(Copy of employee table)

Syntax

```
SELECT    E.Ename[Employee],M.Ename [Manager]
FROM      Employee E
INNER JOIN Employee M
ON        E.Mid= M.Eid
```

Output

Employee	Manager
Mahesh	Ganesh
Ganesh	Sanjay
Sanjay	Sanjay

Case 3 : Non Equi-JOIN

Q. Explain non-equijoin with example.

- No equi-join make use of operators like range operator (BETWEEN) or limit operator (IN) etc.
- Such joins are not used frequently.
- The join query in which equality condition is not used then such joins are called as **Non Equal JOINs.**
- **Example :** To join department and Employee table using joining condition that 'Did' column of employee table should match with 'Did' column of department table.

Employee Table	
Ename	**Salary**
Mahesh	45000
Suhas	41000
Jayendra	32000
Smriti	10000
Grishalda	20000
Shubhra	30000

Job_Grades Table		
Grade_level	**Lowest_Sal**	**Highest_Sal**
A	1	9999
B	10000	19999
C	20000	29999
D	30000	39999
E	40000	49999
F	50000	59999

Syntax

```
SELECT    E.Ename, E.Salary, J.Grade_level
FROM      employees E, Job grades J
WHERE     E.Salary BETWEEN J.Lowest_Sal AND
          ; J.Highest_Sal ;
```

Output

Ename	Salary	Grade_level
Mahesh	45000	E
Suhas	41000	E
Jayendra	32000	D
Smriti	10000	B
Grishalda	20000	C
Shubhra	30000	D

9.5.3 Outer join

Q. What is outer join ?

Q. Explain external join. How is it different from natural join ?

- In an inner join or in case of a simple join, the resultant table contains only the combinations of rows that satisfy the join conditions. Rows that do not satisfy the join conditions are discarded. Outer join, joins two table although there is no match between two joining tables.
- An outer join makes one of the tables dominant. Such table is called the outer table and other table is called as subordinate table.
- In an outer join, the resultant table contains the combinations of rows from dominant table that satisfy the join conditions and also rows that do not having matching rows in the subordinate table.
- The rows from the dominant table that do not have matching rows in the subordinate table contain NULL values in the columns selected from the subordinate table.
- The syntax for performing an outer join in SQL is DBMS dependent.

Definition

The Join that can be used to see rows of table that do not meet the join condition along with rows that satisfies join condition is called as **outer join.**

Following two types of syntax are used generally.

syntax

```
SELECT  Column_List
FROM    Table1
[LEFT/RIGHT/FULL] OUTER JOIN Table2
ON      (JOIN_Condition)
```

Types of Outer Join

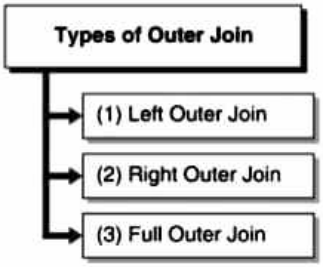
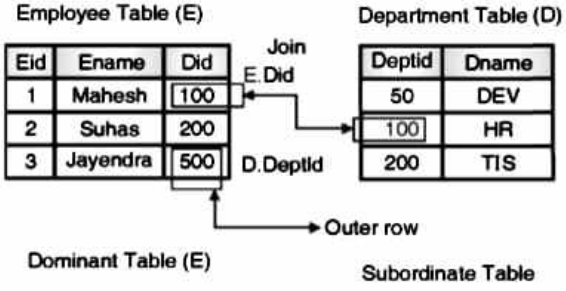


Fig. 9.5.2 : Types of outer join

1. Left outer join

- In the syntax of a left outer join, the dominant table of the outer join appears to the left of the keyword 'outer join'.
- A left outer join returns all of the rows for which the join condition is true and, in addition, returns all other rows from the dominant table and displays the corresponding values from the subordinate table as NULL.

Example



Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Deptid	Dname
50	DEV
100	HR
200	TIS

Syntax

```
SELECT    E.Eid [Eid],
          E.Did [Did],
          D.Dname [Dname]
FROM      Employee E
          LEFT OUTER JOIN Department D
ON        E.Did = D.DeptId
```

Output

Eid	Did	Dname
1	100	HR
2	200	TIS
3	500	NULL

- The above table returns rows in employees table those having or not having any department. The query below will returns only rows in which employees do not have any department. In this query, the database server applies the filter in the WHERE clause after it performs the outer join on Did column of the Employee and Department tables.

Syntax

```
SELECT    E.Eid [Eid],
          E.Did [Did],
          D.Dname [DName]
FROM      Employee E
          LEFT OUTER JOIN Department D
ON        E.Did = D. DeptId
WHERE     D.DeptId IS NULL
```

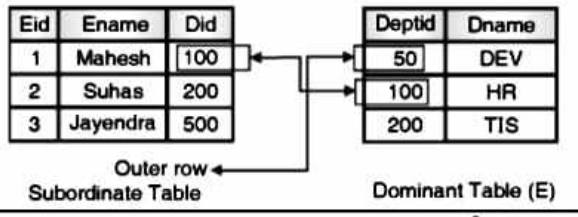
Output

Eid	Did	DName
3	500	NULL

2. Right outer join

- In the syntax of a right outer join, the dominant table of the outer join appears to the right of the keyword 'outer join'.
- A right outer join returns all of the rows for which the join condition is true and also includes all other rows from the dominant table and displays the corresponding values from the subordinate table as NULL.

Example



Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Deptid	Dname
50	DEV
100	HR
200	TIS

Syntax

```
SELECT      E.Eid [Eid],  E.Did [Did],
            D.Dname [Dname]
FROM        Employee E
RIGHT OUTER JOIN Department D
ON          E.Did = D. DeptId
```

Output

Eid	Did	Dname
1	100	HR
2	200	TIS
NULL	50	DEV

- Above table returns all rows from the dominant table Department and as necessary, displays the corresponding values from the subordinate table Employee as NULL.

3. Full outer join

- In full outer join both the table acts as dominant tables alternatively.
- A full outer join returns all of the rows for which the join condition is true and also includes,
  (a) All other rows from the right table and displays the corresponding values from the left table as NULL.
  (b) All other rows from the left table and displays the corresponding values from the right table as NULL.

Example

Employee Table (E)

Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Department Table (D)

Deptid	Dname
50	DEV
100	HR
200	TIS

Outer row ← Outer row →

Dominant Table Dominant Table

Syntax

```
SELECT      E.Eid [Eid],
            E.Did [Did],
            D.Dname [Dname]
FROM        Employee E
FULL OUTER JOIN
            Department D
ON          E.Did = D. DeptId
```

Output

Eid	Did	Dname
1	100	HR
2	200	TIS
3	500	NULL
NULL	50	DEV

9.6 Nested Sub Queries

Q. Explain concept of sub query.

Q. Describe various operators for multiple sub query.

Q. What is nested sub query ? Explain ANY ALL operators with example.

- A subquery is a "query within a query".
- Subquery is query appear within **WHERE** or **HAVING** clause of another query.

```
SELECT      ....
FROM        ....
WHERE       .... (<,>,<=,>=,<>)
                                  ( Sub Query )
HAVING      .... (IN/ANY/ALL)
                                  ( Sub Query )
```

Fig. 9.6.1: Subquery

- Outer query is called as **main query** and inner query which is written in (where or having clause) main query is called **subquery**.
- **Subquery in the WHERE clause :** The result of the subquery (inner query) is used to select some rows from main query.
- **Subquery in the HAVING clause :** The result of the subquery (inner query) is used to select some groups from main query.

- Sub queries can be nested within other sub queries.

Syntax

```
SELECT   select_list
FROM     table
WHERE    expr    operator    (SELECT    select_list
                             FROM  table)
```

- Expr_operator can be of two types like,
  (a) Single row operator (<,>,<=,>=,<>)
  (b) Multiple row operator (IN,ANY,ALL)

9.6.1 Independent Subquery

- A subquery is one which returns only one row to main query.
- A subquery which uses single row operators in above syntax is called as single row subquery.

Example

Find all employees whose salary less than Smriti's salary

Syntax

```
SELECT   ENAME, Salary
FROM     EMPLOYEE
WHERE    Salary <    (SELECT   Salary
                     FROM     EMPLOYEE
                     WHERE    ENAME    = 'Smriti' )
```

50,000 Smriti's salary

Output

Ename	Salary
Mahesh	60000
Jay	51000

List the books whose average sale of books published by TechKnowledge is higher than overall average sale. (Use of subquery in having)

```
SELECT      NAME, AVG (Sale)                  // Outer Query
FROM        SALES, BOOKS
WHERE       Publication = 'TechMax'
AND         SALES.Bid = Books.Bid
GROUP BY    book_name
HAVING      AVG (Sale) > (
                                              // Sub query
Average sale SELECT  AVG (Sale)     overall average
Of books published   FROM  SALES        sale
By Techknowledge Publication
```

Output

Name	AVG(Sale)
DBMS	500
TCS	400

As only one row (Value) is returned by inner query (Subquery) so these types of query is called as Single row subquery.

9.6.2 Multiple Row Subquery

- If subquery returns more than one row then that type of query known as multiple row sub query.
- A subquery which uses multiple row operators in above syntax is called as multiple row subqueries.
- The quantified tests (i.e. **ANY** and **ALL**) uses any one of the simple comparison operators to compare a test value to all of the values returned by a subquery, checking to see whether the comparison holds for some or all of the values.

Table 9.6.1

Sr. No.	Operator	Meaning	Name
1.	**IN**	Equal to any member in the list	**Set Membership Test**
2.	**ANY**	Compare value to each value returned by the subquery	**Quantified**
3.	**ALL**	Compare value to every value returned by the subquery	**Tests**
4.	**EXISTS**	Checks weather subquery returns a value or not.	**Existence Tests**

1. IN

- The subquery forms the set membership test (IN) matches a test value to the set of values returned by a subquery.
- This multiple row operator used to check that a given tuple in main query satisfy at least one values (out of set of values) returned by a subquery.
- Inversely we can use NOT IN condition to check test value is not a member of result set of inner subquery.

Example

Table 9.6.2

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	MAN	5800
141	Rajs	CLERK	3500
142	Davies	CLERK	3100
143	Matos	CLERK	2600
144	Vargas	CLERK	2500
178	Mark	MAN	6800

Find out all employees having salary not equal to any of the manager's salary he should not be manager.

Syntax

```
SELECT   employee_id, last_name, job_id, salary
FROM     employees
WHERE    salary NOT IN
         (SELECT   Salary
          FROM     employee
          WHERE    job_id = 'Man')
AND      job_id <> 'Man' ← employee should not be manager
```

(5800,6800)
Manager's salary

This query selects all rows which are not having value 5800 or 6800 which is salary of employee having job_id 'MAN'.

Output

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	CLERK	3500
142	Davies	CLERK	3100
143	Matos	CLERK	2600
144	Vargas	CLERK	2500

2. ANY

- This multiple row operator used to check that a given tuple in main query satisfies given condition for at least one values (out of set of values) returned by a subquery.
- Checking to see whether the comparison holds true for some values of subquery.

Example

Find out all employees having salary less than any of the managers he should not be manager.

Syntax

```
SELECT  employee_id, last_name, job_id, salary
FROM    employees
WHERE   salary < ANY
        (SELECT  Salary
         FROM    employee
         WHERE   job_id = 'Man')
AND     job_id <> 'Man ';  - - Employee should not be manager
```

(5800,6800)
Manager's salary

Salary of given employee in main query should be greater either 5800 or 6800 (i.e. salary of managers) which is salary of employee having job_id 'MAN'.

Output

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
124	Mourgos	MAN	5800
141	Rajs	CLERK	3500
142	Davies	CLERK	3100
143	Matos	CLERK	2600
144	Vargas	CLERK	2500

3. ALL

- This multiple row operator used to check that a given tuple in main query satisfies given condition for all values (out of set of values) returned by a subquery.
- This quantified tests use one of the simple comparison operators to compare a test value to all of the values returned by a subquery, checking to see whether the comparison holds for all values.

Example

Find out all employees having salary less than all managers he should not be manager.

Output

```
SELECT      employee_id, last_name, job_id, salary
FROM        employees
WHERE       salary < ALL
                (SELECT   Salary
                 FROM     employee
                 WHERE    job_id = 'Man')
AND         job_id <> 'Man';
```

(5800,6800) Manager's salary

Salary of given employee in main query should be greater both 5800 and 6800 (i.e. salary of managers) which is salary of employee having job_id 'MAN'.

Output

EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
141	Rajs	CLERK	3500
142	Davies	CLERK	3100
143	Matos	CLERK	2600
144	Vargas	CLERK	2500

4. EXISTS clause

- The existence test (**EXISTS**) checks whether a subquery returns any values or returns nothing.
- **EXISTS** clause is used for testing whether a subquery has any tuples in there result set or is it empty.
- The **EXISTS** clause results **True** value if the subquery is nonempty else it returns in **False** value.

Example

EMPLOYEE_ID	LAST_NAME	MANAGER_ID	SALARY
141	Rajs	142	3500
142	Davies	143	3100
143	Matos	144	2600
144	Vargas	144	2500

Print all employee details if employee 141 is present in employee table.

Syntax

```
SELECT *
FROM Emp
WHERE EXISTS ( SELECT Eid
               FROM   Emp
               WHERE  Eid=141)
               WHERE  Eid=141)
                        // Literal used to check existence of
                           Employee ID 141
FROM Emp employee id '141'
```

5. NOT EXISTS clause

- This test is complementary test to above existence test checks whether a subquery does not return any values which satisfy search criteria.
- **NOT EXISTS** clause is used for testing whether a subquery does not have any tuples in there result set.
- The **NOT EXISTS** clause results in **True** if the subquery is empty.

Example

EMPLOYEE_ID	LAST_NAME	MANAGER_ID	SALARY
141	Rajs	142	3500
142	Davies	143	3100
143	Matos	144	2600
144	Vargas	144	2500

Print all employee details if employee 141 is not present in employee table.

Syntax

```
SELECT *
FROM Emp
WHERE NOT EXISTS ( SELECT Eid
                   FROM   Emp
                   WHERE  Eid=141)
                   WHERE  Eid=141)  // Literal used
                   to check existence of Employee ID
                   141
FROM Emp employee id '141'
```

9.7 Correlated Sub Queries

(1) Introduction

- In case of simple subquery, first subquery will be solved then using result of subquery, outer query will be solved
- Simple subquery is evaluated once for each query.
- If the previous request is changed slightly, the subquery in the HAVING clause becomes a correlated subquery.
- Correlated subquery is evaluated once for each resultant row in query.
- In case of correlated subquery, first row of main query is found then for this result of main query we solve subquery.
- Correlated sub queries processes data in row-by-row pattern. Each subquery is executed once for every row of the outer query.
- In case of nested queries, we can refer to the table in the FROM clause of the outer query in the inner query using correlated sub-queries.
- We can use a correlated subquery in the HAVING clause.

(2) Syntax

```
SELECT column1, column2, ...
FROM   table1 [ref]
WHERE column1 operator ( SELECT  colum1, column2
                         FROM    table2
                         WHERE expr1 = ref.expr2);
```

The subquery references a column from a table in the main query.

(3) Example

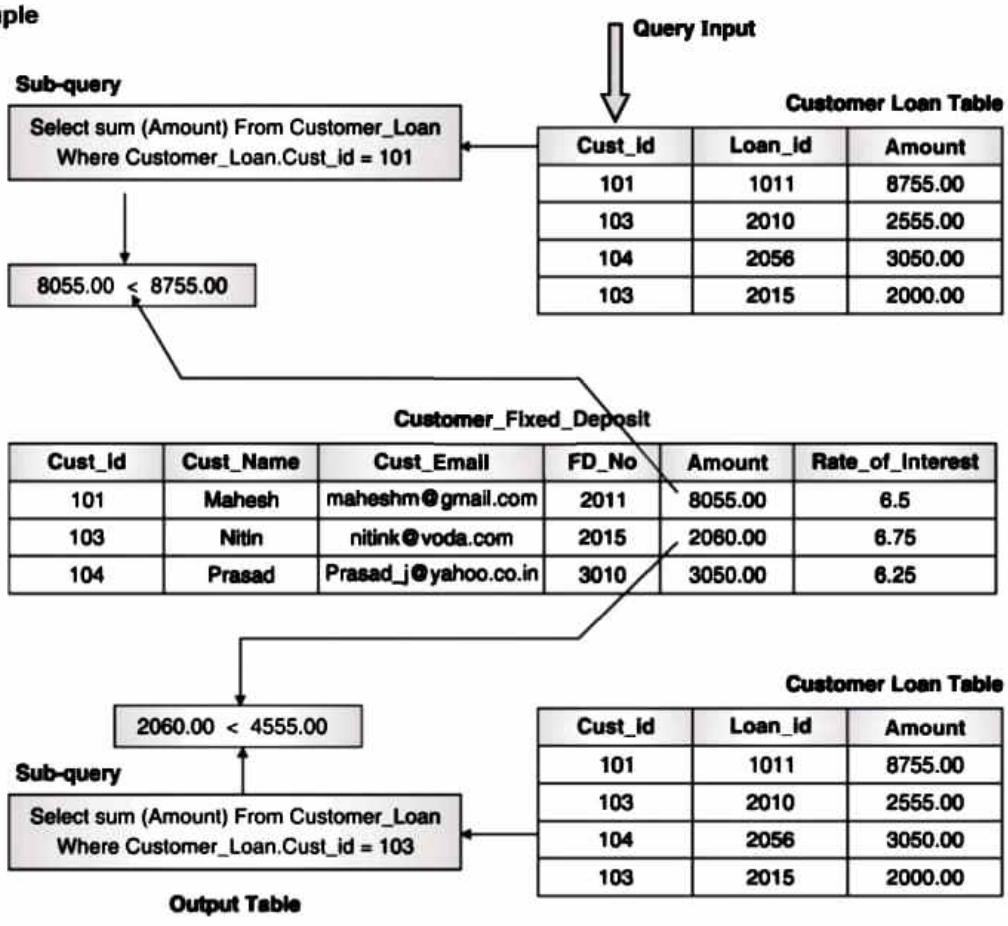


Fig. 9.7.1

The inner query is executed separately for each row of the outer query. i.e. in correlated sub-queries, SQL performs a sub-query over and over again – once for each row of the main query.

Example

Find customers with fixed deposit less than the sum of all their loans.

```
SELECT   Cust_id, Cust_Name
FROM     Customer_Fixed_Deposit [CFD]
WHERE    Amount < ( SELECT     sum(Amount)
                    FROM       Customer_Loan [CL]
                    WHERE      [CL].Cust_id = [CFD].Cust_id )
```

Find all employees who earn less than the average salary in own department.

```
SELECT Faculty_Id, department
FROM   employees outer
WHERE  salary < ( SELECT AVG(salary)
                  FROM   employees
                  WHERE department = outer.department)
```

Find out the employees who have at least one person reporting to them.

```
SELECT      employee_id, last_name
FROM        employees e
WHERE       EXISTS (SELECT   'X'
                    FROM     employee
                    WHERE    manager_id = e.employee_id);
```

TRUE - Print row
FALSE - Do not print that row

- Inner query checks whether the values of 'manager_id' is matched with the any value in 'employee_id' column of employee table (of outer query).
- To check there is any employee working for respective manager.
- If result set of inner query is nonempty depicts that manager having some employees working for him. So, print result on screen.
- The EXISTS operator ensures that the search should not continue further to optimize performance of query.
- To find out the employees who have no one which is reporting to them.

```
SELECT          employee_id, last_name
FROM            employees e
WHERE NOT EXISTS (SELECT 'X'
                  FROM   employees
                  WHERE manager_id = e.employee_id)
```

- Inner query checks values of 'manager_id' is matched with the any value in 'employee_id' column of employee table (of outer query).
- To check there is any employee working for respective manager.
- If result set of inner query is empty depicts that manager having no employees working for him. So, print result on screen.

EMPLOYEE_ID	LAST_NAME	MANAGER_ID	SALARY
141	Raja	142	3500

9.8 Solved Examples

Ex. 9.8.1 : Consider the following relations for database that keeps track of student enrolment in courses and books issued for each course.

STUDENT (Ssn, Name, Subject, DOB)

COURCE (Course_id, Name, Dept)

ENROLL (Ssn, Course_id, Semester, Grade)

BOOK_ISSUED (Course_id, Semester, ISBN)

TEXT (ISBN, Title, Publisher, Author)

Write any 5 SQL Queries.

(1) Write a query to select all courses available in institute.

(2) Find all student details registered for course id 10

(3) Find various book titles and authors for semester higher than 3

(4) Find all students belongs to IT Department (without join)

(5) Find total number of student s enrolled in IT Department.

Soln.:

(1) Writing a query to select all courses available in institute.

```
SELECT  *
FROM    COURSE
```

(2) Finding all student details registered for course id 10.

```
SELECT      S.Ssns.Name
FROM        STUDENT S
Natural JOIN
            ENROLL E
WHERE       E.Course_id = 10
```

(3) Finding various book titles and authors for semester higher than 3.

```
SELECT      B.ISBN, B.Title, B.Author
FROM        BOOK_ISSUED B
NATURAL JOIN
            TEXT T
WHERE       B.Semester > 3
```

(4) Finding all students belongs to IT Department (without join)

```
SELECT      SSN, CName
FROM        STUDENT
WHERE       SSN IN ( SELECT SSN
                     FROM  ENROLL
                     WHERE       Course_id IN
                     (SELECT     Course_id
                     FROM        Course
                     WHERE       Dept = 'IT'))
```

(5) Finding total number of student s enrolled in IT Department.

```
SELECT    COUNT(E.Ssn) Total_Students
FROM      ENROLLE JOIN ( SELECT     Course_id
                         FROM       Course
                         WHERE      Dept = 'IT');
ON        E.Course_id = I.Course_id
```

Ex. 9.8.2 : For the following given database ? Write SQL queries

person (driver_id #, name, address)

car (license, model, year)

accident (report_no, date, location)

owns (driver_id #, license)

participated (driver_id, car, report_number, damage_amount)

(i) Find the total number of people who owned cars that were involved in accident in 1995.

(ii) Find the number of accidents in which the cars belonging to "Sunil K" were involved.

(iii) Update the damage amount for car with licence number "Mum2022" in the accident with report number "AR2197" to Rs. 5000.

Soln. :

(i) We can solve query using subquiry approach we will pick up required data from each table based on

selection criteria or by joining all involved table as follows :

```
SELECT  Count (0.driver_id)
FROM    Person P
        INNER   JOIN Owns 0
        ON P.driver_id = 0.driver_id
        INNER   JOIN participated Pr
        ON P. driver_id = Pr.driver_id
        INNER JOIN Accident A
        ON A. Report_no = Pr.report_no
WHERE
        A.date BETWEEN To_date ('01-01-1995')
        AND To_date ('31-12-1995');
```

(ii)
```
SELECT  Count (report_number)
FROM    participated
WHERE   driver_id IN (SELECT driver_id
                      FROM Person
                      WHERE name='sunil k.')
```

(iii)
```
Update participated
SET damage_amount = 500
WHERE report_number . LIKE 'AR2197'
```

Note : As there is problem with give schema we cannot include car linence number into participated table comparison.

Ex. 9.8.3 : Given the following relational schema.

Division (div #, div-name, director)

Department (dept #, dept-name, location, div #)

Employee (emp#, emp-name, salary, address, dept #)

State the following queries in SQL :

(i) Get the employee name, dept-name and division name for all employees whose salary is above 20,000/-

(ii) List the dept-name and employee names in that dept, for all department whose location is "Mumbai".

Soln. :

(i) Getting the employee name, dept-name and division name for all employees whose salary is above 20,000/-

There are 3 tables involved in this query so we need to join 3 tables i.e. employee, Department and division.

```
SELECT      e.emp_name, d.dept_name, dv.div_name
FROM        EMPLOYEE e
INNER JOIN  Department d
ON          e.dept = d.dept
INNER JOIN  Division dv
ON          d.div = dv.div
WHERE       e.salary > 20000
```

(ii) Listing the dept-name and employee names in that dept, for all department whose location is "Mumbai".

There are only 2 tables are involved in query.

```
Employee → Department
SELECT      e.emp_name, d.dept_name
FROM        Employee e
INNER JOIN  deparment d
ON          E.dept = d.dept
WHERE       d.loction = 'Mumbai'
```

Ex. 9.8.4 : For the following given database, write SQL queries :

person (driver_id #, name, address)

car (license, model, year)

accident (reportCno, date, location)

owns (driver id #, license)

participated (driverid, car, report_number, damage_amount)

(i) Find the total number of people who owned cars that were involved in an accident in 2007.

(ii) Find the number of accidents in which the cars belonging to "Ajay" were involved.

(iii) Find the number of accidents that were reported in Mumbai region in the year 2004.

Soln. :

(i) Finding the total number of people who owned cars that were involved in an accident in 2007.

```
SELECT  count(*)
FROM    person
WHERE   driver_id IN (SELECT driver_id
                      FROM participated
        WHERE car IN (SELECT reportCno
                      FROM accident
                WHERE date between '01-01-2007' AND
                '31-12-2007')
```

(ii) Finding the number of accidents in which the cars belonging to "Ajay" were involved.

```
Select count(*)
FROM accident
WHERE reportCno IN (SELECT report_number
                    FROM participated
                    WHERE car IN (SELECT driver_id
                    FROM person
                    WHERE name = 'Ajay')
```

(iii) Finding the number of accidents that were reported in Mumbai region in the year 2004.

```
SELECT  count(*)
FROM    accident
WHERE   reportCno IN (SELECT report_number
                      FROM participated
                      WHERE car IN (SELECT reportCno
                      FROM accident
                      WHERE location = 'Mumbai'
                      AND date between'01-01-2004'
                      AND '31-12-2004')
```

Ex. 9.8.5 : For the given database, write SQL queries.

Employee (Eid. Name, Street, City)

Works (Eid. Cid. salary)

Manager (Eid, Manager_Name)

Company(Cid, Company_name, city)

(i) Modify the database so that 'Jack' now lives in 'Newyork'.

(ii) Find all employees in the database who live in the same cities as the company for which they work.

Soln. :

(i) Modifying the database so that 'Jack' now lives in 'Newyork'.

```
UPDATE    employee
SET       city = 'Newyork'
WHERE     name = 'Jack'
```

(ii) Finding all employees in the database who live in the same cities as the company for which they work.

```
SELECT  eid,
        city,
        name
FROM    employee
WHERE   city IN
(SELECT city
FROM    company
WHERE   eid = employee·eid)
```

Review Questions

Q. 1 Explain Order by clause with example.

Q. 2 Explain Group by clause with example.

Q. 3 What is having clause ? Explain with example.

Q. 4 Short note on aggregation .

Q. 5 What is outer join ?

Q. 6 Explain external join. How is it different from natural join ?

Q. 7 Explain self join with suitable Example.

Q. 8 What are JOINS ? What are different types of JOINS give example of each.

Q. 9 What is nested sub query? Explain ANY ALL operators with example.

Q. 10 What is correlated sub query? Explain with example.

□□□

CHAPTER 10

UNIT III

Views

Syllabus

Views : Concept of View, The create view command, updating views, views and joins, views and sub-queries, Dropping views.

10.1 Introduction of Views

(MSBTE- W-13, S-14, S-15, W-15, S-16, W-16)

Q. What is view? Write the syntax of create view. W-13, S-14, S-15, S-16, W-16, 2 Marks

Q. Explain views with example. W-15, 4 Marks

Q. What is views ?

1. Definition

- A view is defined as a database object that allows us to create a virtual table in the database whose contents are defined by a query or taken from one or more tables.
- View is defined to hide complexity of query from user.

2. Base table

The table on which view is defined is called as Base table.

3. View - as a window of entire table

Instead of showing entire table to a user we can show a glimpsed of table to the user which is required for him

Example : Consider a student table contains following columns,

STUDENT (Stud_Id, Stud_Name, Std, Div, Addr, Sports, Fees, Cultural_Activity)

Now for a sports teacher requires only sports related data of students so we can create view called as Stud_Sports_View for teacher as below which will only depicts sports data of student to sports teacher.

Stud_Sports_View (Stud_Id, Stud_Name, Sports)

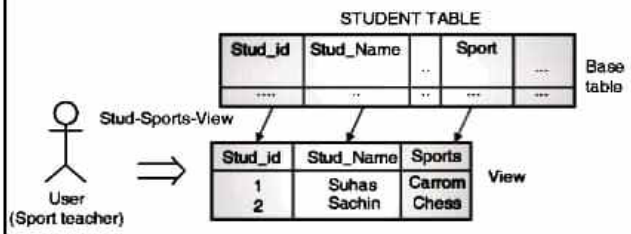


Fig. 10.1.1 : Overview of view

4. Types of views

Q. What are the types of views ?

(a) Simple view
(b) Complex View

(a) Simple view

- The views which are based on only one table called as Simple view.
- Allow to perform DML (Data Manipulation Language) operations with some restrictions.
- Query defining simple view cannot have any join or grouping condition.

(b) Complex view

- The views which are based on more than one table called as complex view.
- Do not allow DML operations to be performed.
- Query defining complex view can have join or grouping condition.

5. Working of views

- When we call view in SQL query it refers to database and finds definition of views which is already stored in database.
- Then the DBMS convert this call of view into equal request on the base tables of the view and carries out the operations written in view definition and returns result set to query from which view is called.

10.2 Creating a Views

(MSBTE- W-13, S-14, S-15, S-16, W-16, W-18)

Q. Write the syntax of create view.

W-13, S-14, S-15, S-16, W-16, 2 Marks

Q. Consider the following schema Depositor (ACC_no, Name, PAN, Balance) Create a view on Depositor having attributes (ACC_no, PAN) where balance is greater than100000.

W- 18, 6 Marks

1. Introduction

- To create a view a sub query must be embedded within the Create View statement.
- The Create statement assigns a name to the view and also gives the query which defines the view.
- To create the view one should have privileges to access all of the base tables on which view is defined.
- Also user must have create view permissions from DBA to create a view in database.
- The create view can change name of column in view as per requirements.

2. Syntax

```
CREATE [OR REPLACE VIEW <view name>
AS
SUB QUERY
[WITH CHECK OPTION]
```

- **OR REPLACE :** Change the definition of a view without dropping (ALTER VIEW)
- **VIEW NAME :** Is name given to a view.
- **SUB QUERY :** The query which retrieves the columns of the table that query must have.
- **WITH CHECK OPTION :** This is type of check constraint, which specifies that only those rows which are selected by view can be inserted updated or deleted.

3. Example

- In the college database, we may want to let a Head of Departments see only the FACULTY rows for own department.

```
CREATE VIEW IT_Faculty
AS
SELECT      *
FROM        FACULTY
WHERE       Faculty_Dept='IT';
```

- View call - Selecting data from above created view

```
SELECT      *
FROM        IT_Faculty ;
```

- If sports teacher of a college wants to see sports information of all students in 'IT' branch.

```
-- View Definition
    CREATE VIEW IT_Sports_Vu
    AS
    SELECT      STUD_ID,
                STUD_Name,
                Sports_Info
    FROM        STUDENT
    WHERE        Branch = 'IT';
-- View Call
    SELECT      *
    FROM        IT_Sports_Vu;
```

- Find summary information of faculty salaries in 'IT' Department.

```
-- View Definition
    CREATE VIEW IT_Faculty_Salary_Summary_Vu
                (
                Total_Faculties,
                Minimum_Salary,
                Maximum_Salary,
                Average_Salary,
                Total_Salary
                )
    AS
    SELECT COUNT(Faculty_ID),
                MIN(Salary),
                MAX(Salary),
                AVG(Salary),
```

```
        SUM(Salary)
    FROM    FACULTY
    GROUP BY Department
    HAVING  Department = 'IT';
-- View Call
    SELECT  *
    FROM    IT_Faculty_Salary_Summary_Vu;
```

The above query will give you,

- Total number of faculties in IT department,
- Minimum salary of faculties in IT department,
- Maximum salary of faculties in IT department,
- Average salary of faculties in IT department,
- Total salary of faculties in IT department.

4. Example

- Consider the following schema Depositor (ACC_no, Name, PAN, Balance)

```
CREATE VIEW depo_Vu
As
SELECT ACC_NO, PAN
FROM depositor
WHERE balance > 1000000
```

10.3 Dropping Views

(MSBTE - S-17)

Q. With the help of example, explain DROP VIEW command DROP query is used to delete a view. **S-17. 2 Marks**

Q. How to drop view ? Give Syntax.

1. Introduction

- To drop a view we use DROP VIEW statement.
- The DROP VIEW statement requires a name to the view.
- To DROP the view one should have must have privileges to from DBA to DROP a view in database.
- The DROP view dose not affects base table or any column of base table.

2. Syntax

```
DROP VIEW <View_name> [RESTRICT|CASCADE]
```

- **RESTRICT :** Delete view only if their in no other view depends on this view.
- **CASCADE :** Delete view along with all dependent view on original view.

3. Example

Remove a view 'IT_Faculty_Salary_Summary_Vu'

```
DROP VIEW       IT_Faculty_Salary_Summary_Vu
CASCADE
 OR
 DROP VIEW      IT_Faculty_Salary_Summary_Vu;
RESTRICT
```

10.4 Modifying a Views

Q. Give syntax for altering views.

- There are some situations in which some modifications are required in view definition.
- The CREATE OR REPLACE statement in view syntax is used to modify view.
- This statement is used by SQL to overwrite the old view definition with new definition without raising any error like existing view with same name.

Syntax

```
CREATE OR REPLACE VIEW <View_name>
AS
SUB QUERY
[WITH CHECK OPTION [CONSTRAINT
; <constraint_name>] ]
[WITH READ ONLY] ;
```

Example

Consider a view defined above 'IT_Faculty' and change it to select all IT faculties having salary above Rs. 25000.

```
CREATE OR REPLACE VIEW IT_Faculty
AS
SELECT      *
FROM        FACULTY
WHERE       Faculty_Dept='IT'
AND         salary > 25000;
```

10.5 Renaming Views

Q. Explain how rename views.

- There are some situations in which some modifications are required in the name of view.

- The RENAME statement in view syntax is used to rename view.
- This statement is used by SQL to overwrite the old view name with new name.

Syntax

```
RENAME TABLE tbl_name
to new_tbl_name [ tbl_name2 to new_tbl_name2]
```

Example

Change name of view old_table.

```
RENAME TABLE old_table to new_table;
```

10.6 Advantages of Views

Q. What are advantages of views ?

(a) Security

- View can restrict user from accessing all data.
- In case of view only data that is given in view is accessible to user. So all data of base table is not accessible to user which will give you security of information.
- For example sports teacher can see data related to sports only and view preventing him from manipulating data pertaining to fees of students.

(b) Hides complexity

- The view may be result of very complex query. Hence instead of writing such complicated query again and again we can store such result to a view and access it whenever we want to access.
- So by writing query we can hide the complexity of original query.

(c) Dynamic nature

- View definition remains unaffected although there is any change in structure of a table.
- This dynamic nature does not hold true in case if base table is dropped or the column selected by view is altered.
- **Example :** If view is made on two tables, selecting two columns from first table and two columns from second table if we add one more column to first table does not cause any change on view.

(d) No direct access to Data Dictionary

- This act like functionality of safeguard to data stored in the data dictionary.
- By this way user cannot change data dictionary to damage database.
- Views can helps to make data in data dictionary easily comprehensible and helpful.

(e) Data integrity

If data is accessed through a view, the DBMS can automatically check the data to check for specified integrity constraints.

10.7 Disadvantages of Views

Q. What are disadvantages of views ?

(a) Performance

- DBMS translates queries of view to queries on base table.
- Sometimes a simple query may take longer time to run. If view is defined by complex multi table query.
- As the complexity of query is hidden by view hence, users are not aware of how much complicated task the query is actually performing.

(b) View management

- The view should be created as per standard then it will be simplifies the job of DBA.
- This happens generally when views are reference other views.
- We need to keep all information of all views in such case so as to it will become very difficult to manage views.

(c) Update restrictions

- When a user tries to update a view, the DBMS must translate this query into an update on rows of the underlying base tables.
- Update is possible for simple views.
- Complex views cannot be updated as they are read-only type of views.

Review Questions

Q. 1 What is views ?

Q. 2 What are the types of views ?

Q. 3 Explain syntax for creating views.

Q. 4 How to drop view ? Give Syntax.

Q. 5 Give syntax for altering views.

Q. 6 Explain how rename views.

❑❑❑

CHAPTER 11

UNIT III

Sequences

Syllabus

Sequences : Creating sequences, altering and dropping a sequence,

Indexes: Index Types, Creating of an Index - simple, unique and Composite Index, Dropping Indexes.

Synonyms : Creating Synonyms, Dropping Synonyms

11.1 Introduction to Database Objects

1. Database is consisting of many database objects like table, view etc.

Object	Description
Table	Used for storage Composed of multiple Rows and Columns
Views	Used to show subset of data from one or more tables
Sequence	Used For Generating primary key values

2. Many applications require the use of unique numbers as primary key values.
3. You can either build code into the application to handle this requirement or use a sequence to generate unique numbers.
4. If you want to improve the performance of some queries, you should consider creating an index. You can also use indexes to enforce uniqueness on a column or a collection of columns.

11.2 Introduction of Sequences

(MSBTE- S-13, W-13, S-14, W-14, W-15, S-16, W-17)

Q. What is sequence ? **S-13, 4 Marks**

Q. What are sequences ? Why it is used ? **W-13, S-14, W-14, S-16, 4 Marks**

Q. Explain sequences with example. **W-15, 4 Marks**

Q. Define sequences. **W-17, 4 Marks**

1. **Sequence :** A sequence is a user created database object that can be shared by multiple users to generate unique integers.
2. A typical usage for sequences is to create a primary key value, which must be unique for each row.
3. Sequences are stored and generated independently. Therefore, the same sequence can be used for generating primary key of multiple tables.
4. A sequence is a **sharable object;** it can be shared between multiple users.
5. You can either build code into the application to generate primary key or use a sequence to generate unique numbers in database so, sequences can replaces application code.
6. Speeds up the efficiency of accessing sequence values when cached in memory.
7. A sequence automatically generates unique numbers by using an internal Oracle routine.
8. This can be a time-saving object because it can reduce the amount of application code needed to write a sequence-generating routine.

9. Sequence numbers are stored and generated independently of tables. Therefore, the same sequence can be used for multiple tables.

11.2.1 Creating Sequences

(MSBTE - W-13, S-14, W-14, S-16, W-17)

Q. Create sequence for student table.

W-13, S-14, W-14, S-16, W-17, 4 Marks

To create a sequence in database we use Database Definition Language (DDL) Statement i.e. CREATE SEQUENCE.

Syntax

```
CREATE SEQUENCE <sequence>
[INCREMENT BY n]
[START WITH n]
[{MAXVALUE n | NOMAXVALUE}]
[{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

Above syntax can be explained as follows,

Syntax	Description
<Sequence>	– It is the name of the a sequence generator – Sequence name is prefixed with 'Seq' (for better readability) – E.g. Seq_StudID
INCREMENT BY n	– Value of *n* specifies the step value between sequence numbers – *Default* value of *n* is 1.
START WITH n	– Value of *n* specifies the first sequence number to be generated from sequence – *Default* value of *n* is 1.
MAXVALUE n	– Value of *n* specifies the maximum value that sequence can generate
NOMAXVALUE	– Value of *n* specifies a maximum value of 10^{27} for an ascending sequence and –1 for a descending sequence – This is the default option.
MINVALUE n	– Value of *n* Specifies the minimum sequence value
NOMINVALUE	– Specifies a minimum value of 1 for an ascending sequence and (10^{26}) for a descending sequence – This is the default option.
CYCLE| NOCYCLE	– Specifies whether the sequence continues to generate values after reaching its maximum or minimum value – Default option is the NOCYCLE.
CACHE n| NOCACHE	– Value of *n* specifies how many values the Oracle server pre allocates and keeps in memory – *Default* value of *n* is 20. (i.e. caches 20 values.)

- Start value must be between value of MINVALUE and MAXVALUE range.
- Cycle Option may duplicate same values so may gives error for primary key.

Example

Create a sequence named STUD_STUDID_SEQ to be used for the primary key of the STUDENT table. The sequence must start with 100 and every key value should be multiple of 5 it must not exceed 9999 and a key value must not be repeated.

```
CREATE SEQUENCE stud_studid_seq
    INCREMENT BY 5
    START WITH 100
    MAXVALUE 9999
    NOCACHE
    NOCYCLE;
Sequence created.
```

The example above creates a sequence named STUD_STUDID_SEQ to be used for the STUDENT_ID column of the STUDENT table. The sequence starts at 100, does not allow caching, and does not cycle.

Generally we do not use the CYCLE option if the sequence is used to generate primary key values, as it may conflict definition of primary key.(Primary key is Unique)

Create a sequence named STUD_ROLLNO_SEQ to be used for the allotting roll numbers to STUDENT table.

```
CREATE SEQUENCE Stud_Rollno_Seq
        INCREMENT BY 1
        START WITH 1
        MAXVALUE 80
        NOCACHE;
Sequence created.
```

The example above creates a sequence named STUD_ROLLNO_SEQ to be used for the STUDENT_Roll_No column of the STUDENT table. The sequence starts at 1, does allow cycle as each class contains 80 students so it restarts with 1 for next class students.

Ex. 11.2.1 : To generate list of numbers as 1,3,5,...,999.

Soln. :

```
CREATE SEQUENCE Stud_Rollno_Seq
        INCREMENT BY 1
        START WITH 1
        MAXVALUE 80
        NOCACHE;
Sequence created.
```

11.2.2 Data Dictionary for Sequences

- Once we have created any sequence entry will be made to the system data directory of that database.
- Data Dictionaries maintain data about sequences,
  1. **USER_OBJECTS** - Contains information about all database objects in database.
  2. **USER_SEQUENCES** - Contains information about only sequences in database.
- Verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT   *
FROM     user_Objects
WHERE    Object_Type='SEQUENCES';
```

- Above query will return the details of all sequences like when it is created object_id, status etc.
- Verify your sequence values in the USER_SEQUENCES data dictionary table.

```
SELECT   sequence_name,
         min_value,
         max_value,
         increment_by,
         last_number
FROM     user_sequences;
```

- Above query will return the details of all sequences about it own properties accepted at the time of creation.
- The LAST_NUMBER column in above table displays the next available sequence number if NOCACHE option is specified.

11.2.3 Referencing Sequence

Once the sequence is created, it generates sequential numbers for inserting in your tables.

For referring sequences we use two pseudo columns.

(a) NEXTVAL

- Gives value of next consecutive number in a sequence.
- This is active clause whenever we call **sequence_name.NEXTVAL** for a sequence it will generate next number in a sequence and once number is generated it cannot be taken back.
- You must qualify NEXTVAL with the sequence name
- It is recommended not to use NEXTVAL in select query; we use this clause at the time of inserting a row in a specified table.

Example

Insert a new department named "Support" in location ID 2500.

```
INSERT INTO students (Stud_id, student_name, class_id)
VALUES (Stud_Rollno_Seq.NEXTVAL, 'Suhas', 25);
1 row created.
```

(b) CURRVAL

- Gives number which is currently generated by a sequence.
- NEXTVAL must be utilised to generate a sequence number in the current user's session before CURRVAL is referenced.
- You must use CURRVAL along with the sequence name.

– When *sequence*.CURRVAL is referenced, the last value used by user is returned.

Example

Find the current value for the DEPT_DEPTID_SEQ sequence.

```
SELECT  dept_deptid_seq.CURRVAL
FROM    dual;
```

(c) Places to use NEXTVAL and CURRVAL

– The VALUES clause of an INSERT statement can contain these clause

– The SET clause of an UPDATE statement can contain these clause

11.2.4 Altering a Sequence

1. Introduction

– If we want to change any of parameter defined in CREATE statement then we will go for ALTER statement to change definition of defined sequence.

– With Help of ALTER Statement we can change the increment value, maximum value, minimum value, cycle option or cache option.

– **Example** : If you reach the maximum value for your sequence, no additional values from the sequence may be generated and you will receive an error indicating that the sequence exceeds the MAXVALUE assigned to it. In such case we need to use ALTER statement to change parameters of sequences.

2. Syntax

```
ALTER SEQUENCE sequence
[INCREMENT BY n]
[{MAXVALUE n | NOMAXVALUE}]
{MINVALUE n | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{CACHE n | NOCACHE}];
```

Syntax	Description
<Sequence>	It is the name of the a sequence to be altered
INCREMENT BY n	Value of *n* specifies the new step value between sequence numbers
MAXVALUE n	Value of *n* specifies the new maximum value that sequence can generate
NOMAXVALUE	Specifies there is no maximum value for new sequence.
MINVALUE n	Value of *n* Specifies the new minimum sequence value
NOMINVALUE	Specifies there is no minimum value for new sequence.
CYCLE | NOCYCLE	Specifies whether the new sequence continues to generate values after reaching its maximum or minimum value
CACHE n| NOCACHE	Value of *n* specifies how many values the Oracle server pre allocates and keeps in memory.

3. Note

– We cannot alter START WITH value as sequence is already created / started.

– If you want to change this value you must drop sequence and create ew sequence with new value.

– You must be the owner of sequence (i.e. sequence must be created by you) or have the ALTER privilege for the sequence in order to change it.

– Only sequence numbers generated after altering sequence are affected by the ALTER SEQUENCE statement.

– The START WITH option cannot be altered using ALTER SEQUENCE. In such cases the sequence must be dropped and re-created in order to restart the sequence at a different number.

4. Example

```
ALTER SEQUENCE Stud_Rollno_Seq
        INCREMENT BY 2
        MAXVALUE 999
```

```
NOCACHE
NOCYCLE;
Sequence altered.
```

- For example, a new MAXVALUE that is less than the current sequence number cannot be given. As it will raise some error.

```
ALTER SEQUENCE dept_deptid_seq
                INCREMENT BY 20
                MAXVALUE 90
                NOCACHE
                NOCYCLE;
ERROR at line 1: ORA-04009: MAXVALUE cannot be made to
be less than the current value
```

11.2.5 Deleting or Dropping a Sequence

- To remove a sequence from the data dictionary and from database we use the DROP SEQUENCE statement.
- You must be the owner of the sequence or have the DROP ANY SEQUENCE privilege to remove it.
- Once removed, the sequence can no longer be referenced. But the primary key values already inserted in table will not be changed or deleted after sequence dropped.

Syntax

DROP SEQUENCE *sequence*;

Example

Ex. 11.2.2 : Create a sequence

Sequence name is Seq_1, Start with1,increment by1, minimumvalue1, maximum value 20.

Use a seq_1 to insert the values into table Student (ID Number (10), Name char(20));

Change the seq_1 max value 20 to 50.

Drop the sequence. **W-18, 6 Marks**

Soln.:

```
(i)   Create Sequence Seq-1
      Start with 1
      Increment by 1
      Minvalue 1
      Max value 20;
(ii)  Insert into Student
      (ID, Name);
(iii) Alter Sequence Seq_1
      Maxvalue 50;
(iv)  Drop Sequence Seq_1;
```

11.3 SQL Indexes

(MSBTE-W-13, W-14, W-15, S-16, S-17, W-17)

Q. Explain index **W-13, W-14, W-15, S-16, W-17, 4 Marks**

Q. Define Index. **S-17, 2 Marks**

1. Introduction

- An index can be created in a table to access data in database more quickly and efficiently.
- MySQL uses indexes to quickly search rows with specific column values as specified in query. Without an index, MySQL engine need to scan the entire table to locate the relevant rows. As table size increases the search speed will be reduced further.
- A database index is a data structure which will improves the speed of operations in a table. Indexes can be created using one or more columns

2. Need of Indexing

- Sometime while fetching data one or more columns in table are more repeating frequently in a WHERE clause of query then indexes on such columns can improve performance and speed of data retrieval.
- The index is generally required on column contains a wide range of values or a large number of null values.

3. Working of Index

- An index will make use of some data structure like B-Tree which will improves the speed of data retrieval on a table, it may include some additional write operations and storage for maintaining it.
- When we create any column of a table with a primary key or unique key, MySQL will automatically create an index named as PRIMARY.
- This type of index is also called the clustered index.

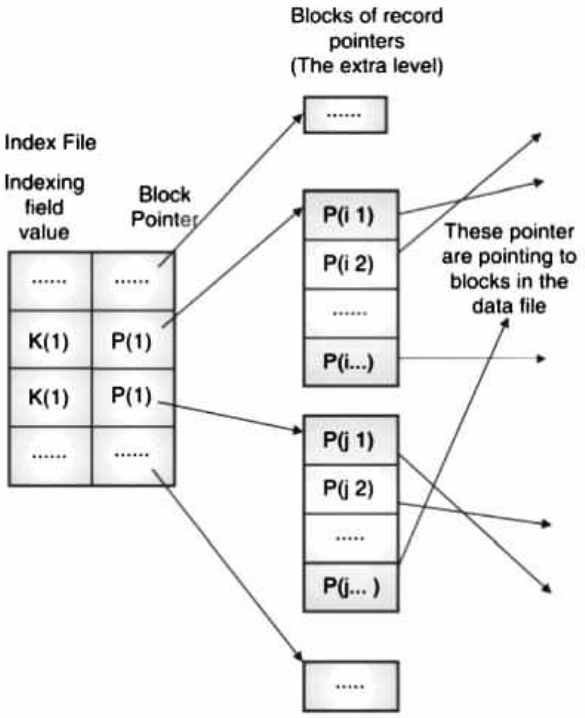


Fig. 11.3.1

- The index creation must consider all columns used in SQL queries and create one or more indexes on such columns.
- Practically, indexes are type of table, which keep key field and a pointer to each record into the table.
- The INSERT and UPDATE statements will take more time on tables due to updates required in index information which creates extra burden on system, whereas the SELECT statements will become fast on such tables.
- The PRIMARY index is stored together with the data in the same table. The clustered index will maintain the order of rows in the table.
- The indexes other than the PRIMARY index are called secondary indexes or also known as **non-clustered indexes.**

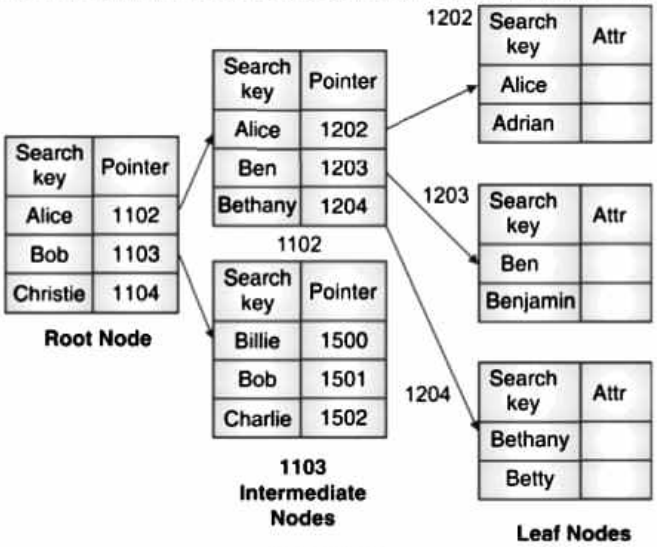


Fig. 11.3.2

- The indexing concept may not be useful if table is very small or there is no condition is applied on column in query also is table is updated very frequently.

4. Performance enhancement by indexing

- The query optimizer may use indexes to locate data at faster speed and without having to scan every row in a table for a given query.
- The index may not be useful if table is very small or there is no condition is applied on column in query also table is updated frequently.

11.3.1 Simple Index

Types of simple index are as follows,

1. Automatic

A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a CREATE TABLE statement.

2. Manual

Users can create non unique indexes on columns to speed up access to the rows.

Syntax (simple Index)

```
CREATE INDEX index
ON table (column[, column]...)
```

Example of Simple Index

- Create index on LAST_NAME column in the EMPLOYEES table.

```
CREATE INDEX emp_last_name_idx
ON employees(last_name);
Index created.
```

- To create a simple index at the time of table creation, we can use given syntax,

```
CREATE TABLE Employee
(       Name INT PRIMARY KEY,
        ID INT NOT NULL,
        DEPT INT NOT NULL,
        GROUP VARCHAR(10),
        INDEX (DEPT, GROUP)
    );
Table created.
```

- Create Function based index on LAST_NAME column in the EMPLOYEES table.

```
CREATE INDEX emp_last_name_idx
ON  employees (UPPER(last_name))
    Index created.
```

11.3.2 Composite Index

- A composite index is an index created on multiple columns of a table.
- MySQL will permit to create a composite index which includes up to 16 columns.
- A composite index is also known as a multiple-column index.
- The order of index columns is very important and it should be arranged in descending order as per their importance.
- Create index on Department name and age column in the EMPLOYEES table.

```
CREATE INDEX emp_Dept_Age_idx
ON employees (dept, age);
Index created.
```

- To create a composite index at the time of table creation, we can use given syntax,

```
CREATE TABLE table_name
(
c1 data_type PRIMARY KEY,
c2 data_type,
c3 data_type,
c4 data_type,
 INDEX index_name (c2, c3, c4)
);
```

11.3.3 Unique Index

- To enforce the uniqueness value of one or more columns, you often use the PRIMARY KEY constraint. However, each table can have only one primary key.
- Hence, if you want to have a more than one column or a set of columns with unique values, you cannot use the primary key constraint.
- Luckily, MySQL provides another kind of index called UNIQUE index that allows you to enforce the uniqueness of values in one or more columns. Unlike the PRIMARY KEY index, you can have more than one UNIQUE index per table.
- To create a unique index at the time of table creation, we can use given syntax,

```
CREATE UNIQUE INDEX index_name
ON table_name(index_column_1,index_column_2,...);
```

- To create a unique index at the time of table creation, we can use below given syntax,

```
CREATE TABLE table_name
(
c1 data_type UNIQUE,
c2 data_type,
c3 data_type,
c4 data_type,
UNIQUE KEY U1(c2, c3, c4)
);
```

- Example to create a unique index at on employee table creation, we can use below given syntax,

```
CREATE TABLE IF NOT EXISTS employee (
id INT AUTO_INCREMENT PRIMARY KEY,
first_name VARCHAR(50) NOT NULL,
last_name VARCHAR(50) NOT NULL,
mobile VARCHAR(10) NOT NULL,
email VARCHAR(100) NOT NULL,
UNIQUE KEY unique_email (email)
);
```

Viewing Table Indexes

- To view the MySQL internally performed this query, you and the EXPLAIN clause at the beginning of the SELECT statement.

```
SELECT  *
FROM    Employees
WHERE   jobTitle = 'Sales'
```

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶	1	SIMPLE	employees	NULL	ref	jobTitle	jobTitle	52	const	17	100.00	NULL

- To view the MySQL internally performed this query, you and the EXPLAIN clause at the beginning of the SELECT statement.

```
SHOW INDEXES FROM employees;
```

11.3.4 Removing Index

- Drop above created index.

```
DROP INDEX upper_last_name_idx;
Index dropped.
```

11.4 Synonyms

- A synonym is an alternative name assigned for database objects such as tables, views, sequences, stored procedures etc.
- A synonym will make easy to access database objects owned by other users.
- Synonyms are used when you are granting access to an object from another schema,

Syntax

```
CREATE      [OR REPLACE] [PUBLIC]
SYNONYM     synonym_name
  FOR       [schema .] object_name;
```

OR REPLACE

Allows you to create the synonym if it already exists

OBJECT_NAME

The name of the object for which you are creating the synonym.

It can be one of the following :

(i) Table

(ii) View

(iii) Stored procedure

(iv) Function

(v) Package

(vi) CREATE PUBLIC SYNONYM suppliers

(vii) FOR app.suppliers;

11.4.1 Creating Synonyms

A synonym is an alternative name assigned for database objects which can be created as given below.

Syntax

Create synonym for Suppliers_Product_Table.

```
CREATE [PUBLIC|Private]
SYNONYM <NewTableName>
FOR <Old_TableName>;
```

Example

Create synonym for Suppliers_Product_Table.

```
CREATE PUBLIC SYNONYM S1
FOR app.Suppliers_Product_Table;
```

Using synonyms

The synonym can be called as,

```
SELECT    *
FROM      S1;
```

11.4.2 Removing Synonyms

A synonym can be removed from database so as table cannot be referred by name of synonym.

Example

Removing synonym for Suppliers_Product_Table.

```
DROP      PUBLIC SYNONYM S1;
```

Review Questions

Q. 1 Explain what do mean by sequence.

Q. 2 How to create a sequence for a student table.

Q. 3 How to alter a sequence.

Q. 4 Explain how to modify sequence and what are its limitations.

Q. 5 Explain deleting or dropping a sequence with syntax.

Q. 6 Short note on indexes.

Q. 7 What are the types of indexes.

Q. 8 Explain CREATE and DROP synonyms.

❑❑❑

CHAPTER 12

UNIT IV

Introduction of PL/SQL

Syllabus

Introduction of PL/SQL, Advantages of PL/SQL, The PL/SQL Block Structure, PL/SQL execution environment, PL/SQL data Types, Variables, Constants.

12.1 Introduction to PL/SQL

- PL/SQL is Oracle Corporation's procedural language which also provides extension to SQL language for accessing data in relational database.
- Many Oracle tools have their own PL/SQL engine which is acting like a separate engine in the Oracle Server.
- PL/SQL programs are written inside a blocks and program can be divided into smaller logical blocks.
- PL/SQL Block is made up of three sections

  Declarative (optional) section

  Executable (required) section

  Exception handling (optional) section
- Only BEGIN and END keywords are required. You can declare variables locally to the block that uses them. Error conditions (known as *exceptions*) can be handled specifically in the block to which they apply.

12.1.1 Block Structure

(MSBTE - W-18)

Q. Draw the block structure of PL/SQL. **W-18, 2 Marks**

- We can store and change values of various variables and identifiers within a PL/SQL block.
- We need to place a semicolon (;) at the end of every SQL or PL/SQL statement.
- We use a slash (/) to run the anonymous PL/SQL block.
- END statement also requires a semicolon to terminate the statement.

DECLARE -- Optional section

We can declare variables, cursors or user-defined exceptions

BEGIN - Mandatory section

SQL statements OR

PL/SQL statements

EXCEPTION – Optional section

Actions to perform when errors occur

END; -- Mandatory section

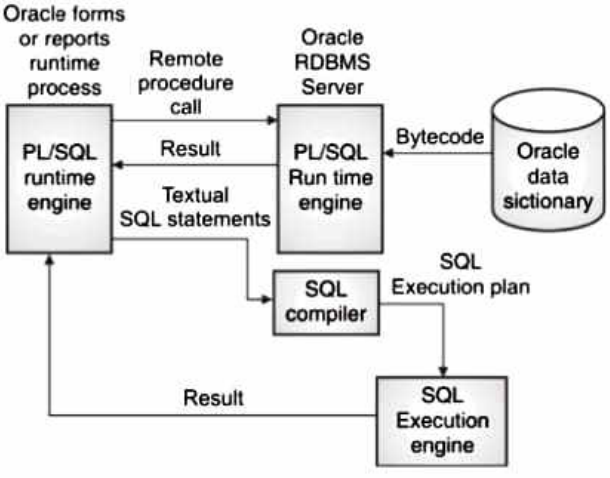


Fig. 12.1.1

12.1.2 PL/SQL Execution Environment

- The PL/SQL engine resides in the mysql Oracle engine, the Oracle engine can process entire PL/SQL blocks.
- These PL/SQL blocks are sent to oracle engine, where procedural statements are executed and SQL statements are sent to the SQL engine in the Oracle. Since the PL/SQL engine resides in the Oracle engine, this will improve speed of operation.
- The Oracle engine will be called only once for execution of any number of SQL statements, if these SQL statements are bundled inside a PL/SQL block.
- Since the oracle engine is called only once for each block, the speed of SQL statement execution is very fast as compared to the Oracle engine called for each SQL sentence.

12.1.3 Sample PL/SQL Code

Simple PL/SQL Block to increment salary of all employees.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET serveroutput ON;
-- the following is an optional declarative section
DECLARE
  v_salary NUMBER(6);
-- the following is the executable section, from BEGIN to END
BEGIN
  UPDATE Emp
  SET Sal = Sal + 0.1*Sal;
-- the following displays output from the PL/SQL section
DBMS_OUTPUT.PUT_LINE('The salary of all employees are Updated ');
END;
```

Output

```
SQL> /
The salary of all employees are Updated
PL/SQL procedure successfully completed.
```

12.1.4 Advantages of PL/SQL

(MSBTE - W-18)

Q. List advantages of PL/SQL. W-18. 2 Marks

- Datatypes designed for SQL can also be used in PL/SQL.
- PL/SQL language offers modern software features like data encapsulation, exception handling, information hiding and object orientation.

1. Procedural Language Capability

- PL/SQL consists of procedural language constructs such as conditional statements (if, if-else) and looping structure like (Basic, While and FOR loops).
- With help of PL/SQL, we can use SQL statements along with PL/SQL control statements to process the data.

2. Block Structures of Code

- PL/SQL consists of blocks structure, which can be nested within each other.
- Each block forms a unit of a task.
- PL/SQL Blocks can be stored in the database and reused whenever needed.

3. Better Performance

- PL/SQL incorporates the data manipulation and SQL query statements to be included in procedural units of code.
- PL/SQL engine can process multiple SQL statements simultaneously as a single block, thereby reducing time and network traffic.

4. Error Handling

- PL/SQL handles exceptions very effectively during the execution of a PL/SQL program.
- Once an exception is caught, specific actions can be taken depending upon relavent catch block. Also, it can be displayed to the user with a message.

12.2 Adding Comments in PL/SQL Block

1) Introduction

- We used to comment code to document each phase of development and to help in debugging PL/SQL programs.

- Comments are strictly informational and do not enforce any conditions or behavior on behavioral logic or data. Well-placed comments are extremely valuable for code readability and future code maintenance.

2) Types of Comments

(a) Single line comments

We can write a single line comment in the PL/SQL code with help of two dashes (--) if the comment is on a single line.

```
BEGIN
--This is PL SQL Comment
END;
```

(b) Multiline comments

We can write a single line comment in the PL/SQL code with help of the symbols /* and */

```
BEGIN
/* This is Multiline
   PL SQL Comment */
END;
```

12.3 Variables and Constants

1) Variables

- Variables are working like a placeholders or memory locations which determine values that you have stored in it.
- You can declare constants and variables in the DECLARE section of any PL/SQL block or any subprogram like procedure or functions.
- Declarations of any variable allocate storage for a value also specifies its datatype and gives a name that you can reference to your variable.
- Declarations can also assign an initial value and some constraint to variables.

2) Constants

- Constants in PL/SQL are name-value pairs that are visible across the programming block.
- Using such constants you can define the value once and then refer to it from the multiple places, instead of redefine the value everywhere it is needed.

3) Naming rules for variables and constants

- Variable name should not start with a digit.
- Variable name should not start with underscore (_).
- Variable name should not contain special characters in it.
- Variable name should not be an oracle reserved word or key word.
- Length of variable name should not exceed 30 characters.

4) Syntax

```
identifier [CONSTANT] DATA_TYPE [NOT NULL | NULL]
[DEFAULT / := Expr]
```

Parameter Name	Description
Identifier	Name we want to assign for a variable
Data_Type	Type of data value you want to store in variable
NOT NULL/ NULL	Null value constraints specify that either value of variable is mandatory or optional
DEFAULT / :=	This clause is used to specify default or initial value of variable.':=' is an assignment operator in PL/SQL.
Expr	Initial value of a variable

5) Example

```
DECLARE
    v_dob        DATE;
    v_branchno   NUMBER(5) NOT NULL := 50;
    v_address    VARCHAR2(50) := 'Unknown';
    v_fees       CONSTANT NUMBER(10,2) := 50000.00;
END;
```

12.4 PL/SQL Output

1) Introduction

- We can declare a host variable and reference it in a PL/SQL block and then display its contents using the PRINT command.

- There is other simple option for displaying information from a PL/SQL block is DBMS_OUTPUT.PUT_LINE.
- DBMS_OUTPUT is an Oracle-supplied package, and PUT_LINE is a procedure within that package.

2) Server output

- The package must be enabled in your session of PL/SQL.
- To make use of package DBMS_OUTPUT.PUT_LINE we need to set server output variable on.
- SET SERVEROUTPUT ON ;

3) Printing text String

To print text string we use single quotation.

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Output Tested ');
END;
```

Output

```
SQL> /
Output Tested
PL/SQL procedure successfully completed.
```

4) Printing PL/SQL Variables

To print variable no need to write single quotation.

```
DECLARE
Var_number Number :=1;
BEGIN
DBMS_OUTPUT.PUT_LINE(Var_number);
END;
```

Output

```
SQL> /
1
PL/SQL procedure successfully completed.
```

5) Appending String and Variables

To print text string, we use single quotation not for integer or number value.

```
DECLARE
Var_number Number :=1;
BEGIN
DBMS_OUTPUT.PUT_LINE('Number Accepted = ' ||
Var_number);
END;
```

Output

```
SQL> /
Number Accepted = 1
PL/SQL procedure successfully completed.
```

12.5 PL/SQL Inputs using Substitution Variable

1) Introduction

We use substitution variable for accepting input using PL/SQL. Substitution variable is prefixed with '&'.

2) Working

- Whenever PL/SQL detects statement contains '&' you are prompted to enter a value for the substitution variable named in the SQL statement.
- Once you enter a value and click enter for Execution
- The results are displayed in the output area of your SQL session.

3) Substitution variables used in

WHERE conditions
ORDER BY clauses
Column expressions
Table names
Entire SELECT statements

4) Accepting text String

To accept text string we use single quotation.

```
DECLARE
Var_N VARCHAR(50) := '&Name';
BEGIN
DBMS_OUTPUT.PUT_LINE(' Accepted Name = '||Var_N);
END;
```

Output

```
SQL> /
Enter value for Name: VisionIT
Accepted Name = VisionIT
PL/SQL procedure successfully completed.
```

5) Accepting number variables

To print variable no need to write single quotation.

```
DECLARE
Var_Number number := &Number;
BEGIN
DBMS_OUTPUT.PUT_LINE(' Accepted Number = '||Var_Number);
END;
```

Output

```
SQL> /
Enter value for number : 12
Accepted Number = 12
PL/SQL procedure successfully completed.
```

Program 12.5.1 : Assuming sales table consisting of columns zone, prodid and quantity. Write a PL/SQL code to accept zone and product id from the user to display total sale of specified product and zone with appropriate labels.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET serveroutput ON;
-- the following is an optional declarative section
DECLARE
   v_zone sales.zone%TYPE:='&Zone';
   v_prodid sales.prodid%TYPE:='&Product_Id';
   v_quantity sales.quantity%TYPE;
   v_TotalSales sales.quantity%TYPE;
-- the following is the executable section, from BEGIN to END
BEGIN
  SELECT SUM(quantity)
  INTO v_TotalSales
  FROM Sales
  WHERE prodid = v_prodid
  AND zone = v_zone;
-- the following displays output from the PL/SQL section
DBMS_OUTPUT.PUT_LINE('# Result ');
DBMS_OUTPUT.PUT_LINE('Product ID : '||v_prodid);
DBMS_OUTPUT.PUT_LINE('Zone : '||v_zone);
DBMS_OUTPUT.PUT_LINE('Total Sales : '||NVL(v_TotalSales,0));
END;
```

Output

```
SQL> /
Enter value for zone: S
Enter value for product_id: 1
# Result
Product ID : 1
Zone : S
Total Sales : 16
PL/SQL procedure successfully completed.
```

Program 12.5.2 : Write PL/SQL block to remove all rows of specified zone (consider above mentioned sales table).

```
-- the following parameter is required for printing DBMS_OUTPUT
SET serveroutput ON;
-- the following is an optional declarative section
DECLARE
   v_zone sales.zone%TYPE:='&Zone';
-- the following is the executable section, from BEGIN to END
BEGIN
  DELETE
  FROM Sales
  WHERE zone = v_zone;
-- the following displays output from the PL/SQL section
DBMS_OUTPUT.PUT_LINE('# Result ');
DBMS_OUTPUT.PUT_LINE('Zone '||v_zone||' is deleted.');
END;
```

Output

```
SQL> /
Enter value for zone: W
# Result
Zone W is deleted.
PL/SQL procedure successfully completed.
```

12.6 PL/SQL Expressions and Comparisons

1) Comparison Expression

These types of operators that can be used to compare multiple variables or constant values. Such operations are results in Boolean value.

Operators

Operator	Meaning
<	Less Then
<=	Less Then equal to
>=	Greater Then
=	Greater Then equal to
<>	Not Equal to

Examples

Let, var_a = 10 , var_b = 20 ;	
Expression	**Result**
var_a > var_b	False
var_a <= 100	True
var_a <> var_b	True

2) Logical Expression

This type of operators are generally used to concatenate multiple comparison expression.

Such operations are results in Boolean value.

Operators

Operator	Meaning
AND	Give output as TRUE if both compairing expressions are TRUE and FALSE if any of compairing expression is FALSE
OR	Give output as TRUE if any of compairing expression is TRUE and FALSE if both compairing expressions are FALSE
NOT	Give output as TRUE if any of expression is FALSE and FALSE if expressions are TRUE

Examples

Let, var_a = 10 , var_b = 20 ;	
Expression	**Result**
var_a > var_b **AND** var_a <= 100	False
var_a > var_b **OR** var_a <= 100	True
NOT var_a <> var_b	False

3) Range Comparison Operators

These operators can compare a variable with multiple available values either in range or from set of values.

Operators

Operator	Meaning
IN	Used to check variable is equivalent with multiple available values given in IN expression
BETWEEN	Used to check variable is equivalent with given range of values.

Examples

Let, var_a = 10 , var_b = 20 ;	
Expression	**Result**
var_a **IN** (10,20,30)	True
var_a **NOT IN** (10,20,30)	False
var_b **BETWEEN** 100 **AND** 200	False
var_b **NOT BETWEEN** 100 **AND** 200	True

4) Nullness Comparison Operators

These operators can compare a variable with multiple available values either in range or from set of values.

Operators

Operator	Meaning
IS NULL	Returns true if value of variable is null
IS NOT NULL	Returns true if value of variable is contains some not null value

Examples

Let, var_a = 10 , var_b = NULL ;	
Expression	**Result**
var_a IS NULL	False
var_b IS NOT NULL	True

5) Existence

These operators can check whether query result is available or not.

Operators

Operator	Meaning
EXIST	Returns true if query returns more then one tuple
NOT EXIST	Returns true if query returns no row

Examples

Let, SELECT Eid FROM Emp ;

Eid
10
20
30

Expression	Result
EXIST (SELECT Eid FROM Emp)	True
NOT EXIST (SELECT Eid FROM Emp)	False

12.7 PL/SQL Data Types

(MSBTE – S-15)

Q. What are the various data types of PL/SQL ? **S-15, 4 Marks**

12.7.1 Built-in Data Types

The basic data types available for attributes include the following :

1. Numeric Datatypes
2. Character String datatypes
3. Datetime Datatypes
4. Interval types
5. Cast Datatypes

1. Numeric Datatypes

This datatype is used to store a number values that can be decimal or floating point values.

(a) Integer number of various sizes

This type of system is used to store natural number which is not having any decimal values.

Example : 111, 23 etc.

As per size of number we can use following types of integers.

Data Type	Abbreviation	Description	Range
INTEGER	INT	Integer numerical with precision 10.	– 2,147,483,648 to 2,147,483,647

(b) Formatted numbers

This system used for storing some special numbers which may be of greater size than integers and floating point numbers.

Example : 1.12342 (Numeric (1, 5)), 12.234 (Numeric (2, 3)) etc.

NUMBER (i, j)

Where i = Precision

= Total number of decimal digits

(Default is implementation defined)

j = Scale

= Total number of digits after digits

(Default value is 0)

Data Type	Abbrevation	Description	Range
NUMBER(p,s)	NUM(p, s)	Exact numerical with precision p and scale s.	$1 \le p \le 45$ $0 \le s \le p$
NUMERIC	NUM	NUM is equivalent to NUM (15,0)	--
NUMERIC(p)	NUM (p)	NUM is equivalent to NUM (p,0)	--

2. Character String Datatypes

This data type is used to store a character string which is combination of some alphabets and enclosed in single quotation marks.

E.g. 'Mahesh','abc' etc.

(a) Fixed Length : CHAR (n);

Where n = number of characters

E.g. If 'abc' is stored in char (10) will be stored as 'abc' (abc padded with 7 blank spaces)

(b) Varying Length : VARCHAR (n)

Where n = maximum number of characters

E.g. If 'abc' is stored in varchar(10) will be stored as 'abc' (No blank spaces)

Data Type	Abbreviation	Description	Range
CHARACTER(n)	CHAR(n)	Character string of fixed length n.	$1 \le n \le 15000$
CHARACTER	CHAR	CHAR is equivalent to CHAR(1)	--
CHARACTER VARYING(n)	VARCHAR(n)	Variable length character string of maximum length n.	$1 \le n \le 15000$
CHARACTER VARYING	VARCHAR	VARCHAR is equivalent to VARCHAR(1)	--

3. Datetime Datatypes

(a) Date

- The DATE data type has ten positions and its components are YEAR, MONTH and DAY in form YYYY-MM-DD.
- The length is 10.

Example : Date '2009-01-01' (as 'YYYY-MM-DD')

CurrentDate : Returns current Date

(b) Timestamp

- The TIMESTAMP data type includes date and time fields, plus a minimum of six positions and for decimal fractions of second and optional with TIMEZONE Qualifier.
- Represented using the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND in the format YYYY-MM-DD HH:MM:SS[.sF] where F is the fractional part of the SECOND value.
- If a second precision is not specified, *s* defaults to 6. The length is 26 (or 19, if s = 0 or 20+*s*, if *s* > 0).

Example: TimeStamp '2009:01:01 11:16:59 648302' (As 'YYYY-MM-DD HH:MM:SS TIMEZONE')

CurrentTimeStamp: Local date and time without timezone

4. Interval Datatypes

This specifies an interval a relative value that can be used to increment or decrement an absolute value of date, time or timestamp.

INTERVAL YEAR TO MONTH Datatype

Example : '21-5' Year(2) To Month indicates an interval of 21 years and 5 months

'21-5' Year (2) indicates an interval of 21 Years

'5' Month (2) indicates an interval of 5 month

INTERVAL DAY TO SECOND Datatype

Example : '5 03:15:20' Day (2) To Second indicates an interval of 5 days, 3 hours, 15 minutes and 20 seconds.

5. Cast Datatypes

- The format of date, time and timestamp can be considered as a special type of string.
- Hence, they can generally be used in string comparison by being cast into equivalent strings.

Example : CAST joining_date AS DATE (or CAST '2009-01-01' AS DATE)

12.7.2 User Defined Data Types (UDT)

- A good database system should be able to detect wrong data stored in type (Character data stored in integer field) to support such checks SQL provides distinct types.

- We can use ***create type*** clause to define new types as follows,

  CREATE type T_Id as varchar (70)

- We will use above created new type in a table,

```
CREATE table
Employee (EMP_ID T_Id,
          E_Name char (25));
```

- SQL also provides **drop type** and **alter type** as follows,

```
DROP type T_Id;
ALTER type T_Id as varchar (75);
```

12.7.3 Large Object Types (LOB)

Introduction

- In real world database applications we need a large storage for storing various attributes which stores objects such as photograph of person, voice message in music file or video files.
- SQL provides new large object data type for character data (***clob***) and binary data (***blob***).***Lob*** do mean by ***large objects.***

Example

```
Book_E_Copy clob (200KB)
Emp_Photo blob (10MB)
Meeting_Video blob (10GB)
Call_Recording blob (10GB)
```

Locator

a. It is impractical to retrieve entire large object into a memory. Instead of that we can use a locator concept for identifying location of data.

b. Locator concept explains storing link (address) of data into databases instead of actual data.

c. This concept also helps you to retrieve data in small pieces rather than accessing it all at once.

12.8 CASE Expressions

- A CASE expression selects a result and returns it to the main query or program.
- A CASE expression returns the first value for which the result of the comparison predicate evaluates to True, including comparisons using the IS NULL and IS NOT NULL comparison predicates.
- The following example shows how to use a searched CASE expression to properly check for Null.

Syntax

```
SELECT CASE selector_variable
        WHEN expr_1 THEN Statement _1
        WHEN expr_2 THEN Statement _2
             ...
        WHEN expressionN THEN Statement _n
            [ELSE Statement _n+1;]
    END;
FROM TableName;
```

Example

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Amit	24/12/72	NULL	16
101	Mahesh	01/01/86	DT	10
102	Nitin	28/11/66	NULL	10
103	Omprakash	03/02/80	NULL	8
104	Yogesh	17/07/64	DSA	16

Find faculty details if he has allotted subject then print subject status as not allotted else print subject name.

```
SELECT   Faculty_Name,
CASE
WHEN Subject IS NULL
THEN 'Not Allotted'
-- This will be returned when Subject is NULL
WHEN Subject = 'DT'
THEN 'Database Technologies'
     -- This will be returned when Subject = 'DT'
WHEN Subject = 'DSA'
THEN 'Data structure and Algorithm'
     -- This will be returned when Subject = 'DSA'
END "Subject"
FROM        Faculty
```

Output

Faculty_Name	Subject
Amit	Not Allotted
Mahesh	Database Technologies
Nitin	Not Allotted
Omprakash	Not Allotted
Yogesh	Discrete structure and Algorithm

12.8.1 DECODE Expression

- DECODE takes three or more expressions as arguments.
- These functions works like if else statement.
- Each expression can be a column, a literal, a function, or even a sub query.

Syntax

```
DECODE (parameter_1, parameter_2, parameter_3, parameter_4)
```

This statement works like below If Else statement

```
IF parameter_1 =parameter_2
THEN result = parameter_3;
ELSE result = parameter_4;
END IF
```

Example

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Amit	24/12/72	NULL	16
101	Mahesh	01/01/86	DT	10
102	Nitin	28/11/66	NULL	10
103	Omprakash	03/02/80	NULL	8
104	Yogesh	17/07/64	DSA	16

Syntax

```
SELECT Faculty_Name,
 DECODE (Subject, NULL, 'Not Allotted', 'Allotted')
[Subject Status],
FROM Faculty
```

Output

Faculty_Name	Subject Status
Amit	Not Allotted
Mahesh	Allotted
Nitin	Not Allotted
Omprakash	Not Allotted
Yogesh	Allotted

- In this example, the first expression is a column, the second is NULL, and the third and fourth expressions are character literals.
- The intent is to determine whether each Faculty has allotted subject or not by checking whether Subject column is NULL. The DECODE function in this example compares each row's Subject column (the first expression) to NULL (the second expression).
- If the result of the comparison is true, DECODE returns 'Not Allotted' (the third expression); otherwise, 'Allotted' (the last expression) is returned.

Review Questions

Q. 1 State the advantages of PL/SQL.

Q. 2 Short note on variables.

Q. 3 Short note on constants.

Q. 4 Explain DECODE expression with example.

Q. 5 What are different types of expressions in PL/SQL.

Q. 6 Explain how data conversion is done in PL/SQL.

Q. 7 Explain date data type in PL/SQL.

Q. 8 Briefly explain various character datatypes available in PL/SQL.

Q. 9 Comment on : "CASE expression is more compact than IF – THEN – ELSE Statement.".

❑❑❑

CHAPTER 13

UNIT IV

PL/SQL Control Structure

Syllabus

Control Structure : Conditional Control, Iterative Control, Sequential Control.

Exception Handling : Predefined Exception, User Defined Exception

13.1 Control Structures in PL/SQL

(MSBTE -S-15)

Q. What are the various control structure statements used in PL/SQL ? **S-15, 4 Marks**

- A control structure is a primary concept in almost all the high-level programming languages.
- PL/SQL is working as a block of code; control structures are blocks of code that manages the flow of control.
- In other words, we can say control structure is a container for a series of instructions or statements.
- A simple example of a control structure is, if statement, If a>b then print "a" else print "b". These statements will be give which one is maximum between a and b.
- There are main three types of control structures as follows,
  1. Conditional Control (IF statement and CASE statement)
  2. Iterative Control (Basic Loop, While Loop and For Loop)
  3. Jump or unconditional branch (Goto or Exit)
- Control structures are used by Developers for reading or maintaining code should be easy.
- We can change the logical flow of statements within the PL/SQL block with a number of control structures.

13.2 Decision Making Statements

1. Decision making statements are used to select one out of many options available for execution,
2. There are two types of decision making statements,
   - IF Statement
   - CASE Statements

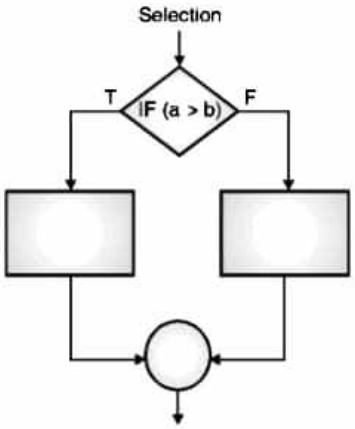


Fig. 13.2.1

13.2.1 IF Statement

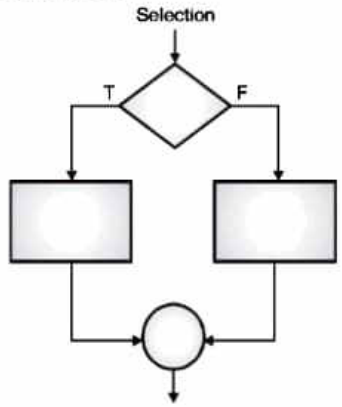


Fig. 13.2.2

(a) Introduction

- If statement is a basic decision making statement for almost all popular languages.
- Syntax of IF statement is very simple to understand.
- IF statement can be used to check one or more conditions.

(b) Syntax

```
IF condition THEN
    Statements;
[ELSEIF condition THEN
    Statements;]
[ELSE
    Statements;]
END IF;
```

(c) Types of IF statements

1. Simple IF statement (IF – THEN – END IF)
2. IF..ELSE statement (IF – THEN – ELSE – END IF)
3. Ladder IF..ELSE Statement (IF – THEN – ELSEIF – END IF)

1. Simple IF Statement

- This IF statement is used to execute statement based on some condition.
- There is a condition and block of code to be executed if condition satisfies there if condition does not satisfies control goes out of IF statement(next line of End If statement).
- This statement is used if we want to execute some block of code only if condition is meets valid no code for invalid condition.

Syntax

```
IF condition THEN
    Statements;
END IF;
```

Examples :

Program 13.2.1 : Write a PL/SQL block to print given number only if it is greater than 100.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
--  Number is accepted using substitution variable "Number"
    v_number        NUMBER(6):=&Number;
-- the following is the executable section, from BEGIN to END;
BEGIN
IF v_number > 100 THEN
-- the following displays output from the PL/SQL section
 DBMS_OUTPUT.PUT_LINE(v_number || ' is greater than 100');
END IF;
END;
```

Output

```
SQL> /
Enter value for Number: 122
122 is greater than 100
PL/SQL procedure successfully completed.
```

Program 13.2.2 : Write a PL/SQL block to accept total marks and obtained marks print 'successful' if student passes.(assume passing above 35%)

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  v_TotalMarks      INTEGER:=&TotalMarks;
  v_ObtndMarks      INTEGER:=&ObtainedMarks;
  v_PerMarks        INTEGER;
-- the following is the executable section, from BEGIN to END
BEGIN
V_PerMarks := V_ObtndMarks / V_TotalMarks;
V_PerMarks := V_PerMarks * 100;
IF v_PerMarks > 35 THEN
-- the following displays output from the PL/SQL section
  DBMS_OUTPUT.PUT_LINE('Successful');
END IF;
END;
```

Output

```
SQL> /
Enter value for totalmarks : 100
Enter value for obtainedmarks : 23
PL/SQL procedure successfully completed.
SQL> /
Enter value for totalmarks : 100
Enter value for obtainedmarks : 55
Successful
PL/SQL procedure successfully completed.
```

2. IF..ELSE Statement

- This statement is used to execute some statement if condition is true otherwise executing some other statement.
- There this IF statement we have two conditions first block is executed if condition is true else alternative block will be executed.
- This statement is used if we want to execute some block of code only if some condition is meets valid otherwise execute another block of code.

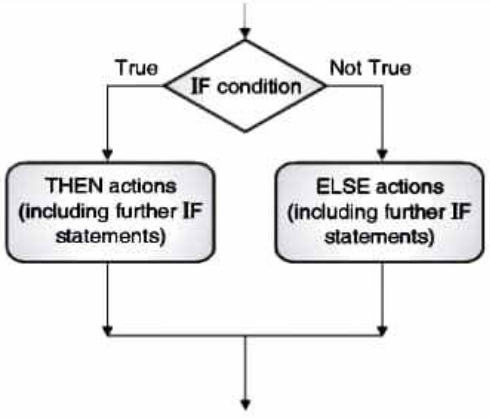


Fig. P.13.2.2

Syntax

```
IF condition THEN
    Statements;
ELSE
    Statements;
END IF;
```

Examples :

Program 13.2.3 : Write a PL/SQL block to check given number is greater than 100 or not.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  v_number        NUMBER(6):=&Number;
-- the following is the executable section, from BEGIN to END
BEGIN
IF v_number > 100 THEN
-- the following displays output from the PL/SQL section
  DBMS_OUTPUT.PUT_LINE(v_number || ' is greater than 100');
ELSE
-- the following displays output from the PL/SQL section
```

```
 DBMS_OUTPUT.PUT_LINE(v_number || ' is not greater than 100');
END IF;
END;
```

Output

```
SQL> /
Enter value for Number : 22
22 is not greater then 100
PL/SQL procedure successfully completed.
SQL> /
Enter value for Number: 122
122 is greater than 100
PL/SQL procedure successfully completed.
```

Program 13.2.4 : Write a PL/SQL block to accept total marks and obtained marks and cutoff percentage. print 'successful' if student passes otherwise print 'unsuccessful'(assume passing above 35% of marks)

```
-- the following is an optional declarative section
DECLARE
  v_TotalMarks      INTEGER:=&Total_Marks;
  v_ObtndMarks      INTEGER:=&Obtained_Marks;
  v_CutOff          INTEGER:=&CutOff_Percentage;
  v_PerMarks        INTEGER;
-- the following is the executable section, from BEGIN to END
BEGIN
V_PerMarks := V_ObtndMarks / v_TotalMarks;
V_PerMarks := V_PerMarks * 100;
IF V_PerMarks > V_CutOff THEN
  DBMS_OUTPUT.PUT_LINE('Successful');
ELSE
  DBMS_OUTPUT.PUT_LINE('Unsuccessful');
END IF;
END;
```

Output

```
SQL> /
Enter value for total_marks : 206
Enter value for obtained_marks : 126
Enter value for cutoff_percentage : 45
Successful
PL/SQL procedure successfully completed.
SQL> /
Enter value for total_marks : 254
Enter value for obtained_marks : 111
Enter value for cutoff_percentage : 55
Unsuccessful
PL/SQL procedure successfully completed..
```

3. Ladder IF..ELSE Statement / Nested IF..ELSE statement

- This statement is used to execute some statements based on many conditions.
- Likewise if we need to decide the grade from marks we have to check it against many conditions. Is it above 75, 60 or 50 ? And other.

Syntax

```
IF condition THEN
    Statements;
ELSEIF condition THEN
    Statements;
ELSE
    Statements;
END IF;
```

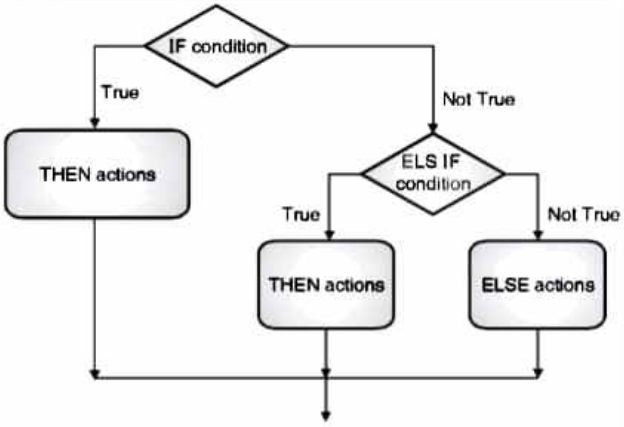


Fig. P. 13.2.4

Examples :

Program 13.2.5 : Write a PL/SQL block to accept total marks and obtained marks print report of student (Class: A, B, C or Unsuccessful)

```
-- the following is an optional declarative section
DECLARE
  V_TotalMarks      INTEGER:=&Total_Marks;
  V_ObtndMarks      INTEGER:=&Obtained_Marks;
  V_CutOff          INTEGER:=&CutOff_Percentage;
  V_PerMarks        NUMBER(5,2);
-- the following is the executable section, from BEGIN to END
BEGIN
V_PerMarks := V_ObtndMarks / V_TotalMarks;
V_PerMarks := V_PerMarks * 100;
IF V_PerMarks >= 60 THEN
  DBMS_OUTPUT.PUT_LINE('Class : A');
ELSEIF V_PerMarks >= 50 THEN
  DBMS_OUTPUT.PUT_LINE('Class : B');
ELSEIF V_PerMarks >= V_CutOff THEN
  DBMS_OUTPUT.PUT_LINE('Class : C');
ELSE
  DBMS_OUTPUT.PUT_LINE('Unsuccessful');
END IF;
END;
/
```

Output

```
SQL> /
Enter value for total_marks : 200
Enter value for obtained_marks : 120
Enter value for cutoff_percentage : 35
Class : A
PL/SQL procedure successfully completed.
SQL> /
Enter value for total_marks : 100
Enter value for obtained_marks : 55
Enter value for cutoff_percentage : 35
Class : B
PL/SQL procedure successfully completed.
SQL> /
Enter value for total_marks : 100
Enter value for obtained_marks : 40
Enter value for cutoff_percentage : 35
Class : C
PL/SQL procedure successfully completed.
SQL> /
Enter value for total_marks : 200
Enter value for obtained_marks : 20
Enter value for cutoff_percentage : 35
Unsuccessful
PL/SQL procedure successfully completed.
```

Program 13.2.6 : Create a table place with attributes room_id, building, room_number, number_seats, description. Write a PL/SQL block to comment on the type of place as Fairly Small, a little bigger, lots of room depending upon the value of the number_seats for a given room_id in place table.

Query 1 :

```
-- Creating and populating Place table
CREATE TABLE PLACE
(
RoomId              VARCHAR2(10),
```

```
Building        VARCHAR2(20),
RoomNo          VARCHAR2(10),
NoSeats         Integer,
Descr           VARCHAR2(50)
)
```

Output

ROOMID	BUILDING	ROOMNO	NOSEATS	DESCR
1	A	1	2	
2	B	2	4	
3	C	7	6	

Query 2 :

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  V_number_of_seats INTEGER;
  V_room_id INTEGER:=&Room_Id;
-- the following is the executable section, from BEGIN to END
BEGIN
        SELECT    noseats
        INTO      V_number_of_seats
        FROM      place
        WHERE     roomid = V_room_id;
        IF V_number_of_seats<=2 THEN
                  UPDATE place
                  SET descr='fairly small'
                  WHERE roomid=V_room_id;
        DBMS_OUTPUT.PUT_LINE('Room ID '||V_ROOM_ID ||' is fairly small.');
       ELSIF V_number_of_seats BETWEEN 3 AND 5 THEN
                  UPDATE place
                  SET descr='little bigger'
                  WHERE roomid=V_room_id;
        DBMS_OUTPUT.PUT_LINE('Room ID '||V_ROOM_ID ||' is little bigger.');
        ELSE
                  UPDATE place
                  SET descr = 'lots of space'
                  WHERE roomid=v_room_id;
        DBMS_OUTPUT.PUT_LINE('Room ID '||V_ROOM_ID ||' is having lots of space.');
    END if;
END;
```

Output

```
SQL> /
Enter value for room_id: 1
1 is fairly small.
PL/SQL procedure successfully completed.
SQL> /
Enter value for room_id: 2
Room ID 2 is little bigger.
PL/SQL procedure successfully completed.
SQL> /
Enter value for room_id: 3
Room ID 3 is having lots of space.
PL/SQL procedure successfully completed.
```

Query 3 :

```
SQL> SELECT * FROM PLACE
```

Output

ROOMID	BUILDING	ROOMNO	NOSEATS	DESCR
1	A	1	2	fairly small
2	B	2	4	little bigger
3	C	7	6	lots of space

Program 13.2.7 :Consider following of PL/SQL block :

```
BEGIN
a := NULL;
b := NULL;
...
IF a = b THEN
     Statement1;
ELSE
     Statement2;
END IF;
END;
```

Explain which statement will get executed ? And Why ?

Statement 1 get executed because value of a and value of b are same. And If value of a is not equal to value of b then statement 2 will be executed.

13.2.2 CASE Statement

- The case statement evaluates a condition and performs an action, such as an entire PL/SQL block, for each case.
- In SQL, CASE expression is used to make decisions.
- A CASE expression can select a result and returns it using SELECT clause of SQL.
- CASE statement is used in PL/SQL for decision making purpose.
- It is working like IF..Else decision making statement But, CASE statement is more readable and more efficient.
- CASE statement evaluates a condition and returns a value for each case.

Types of CASE Statement

1. Selector CASE
2. Searched CASE

1. Selector Case

- To select the result, the CASE statement uses an expression whose value is used to select one of several alternatives.

```
CASE selector_variable
WHEN expr_1 THEN Statement _1
WHEN expr_2 THEN Statement _2
...
WHEN expressionN THEN Statement _n
[ELSE Statement _n+1;]
END;
```

- The selector is followed by one or more WHEN clauses, which are checked sequentially.
- The value of the selector determines which clause is executed.
- If the value of the selector equals the value of a WHEN -clause expression, that WHEN clause is executed.

Example

- If the example is written using a selector CASE expression it will look like this :

```
SET SERVEROUTPUT ON
DECLARE
v_grade CHAR(1) := UPPER('&p_grade');
BEGIN
CASE v_grade
WHEN 'A' THEN
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal is Excellent');
WHEN 'B' THEN
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal is Very Good');
WHEN 'C' THEN
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal is Good');
ELSE
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal is No such grade');
END CASE;
END;
OR (Alternative solution)
DECLARE
v_grade CHAR(1) := UPPER('&p_grade');
v_appraisal VARCHAR2(20);
BEGIN
v_appraisal :
CASE v_grade
WHEN 'A' THEN 'Excellent'
WHEN 'B' THEN 'Very Good'
WHEN 'C' THEN 'Good'
ELSE 'No such grade'
END;
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal ' || v_appraisal);
END;
```

- In the example, the CASE expression uses the value in the v_grade variable as the expression.
- This value is accepted from the user using a substitution variable. Based on the value entered by the user, the CASE expression evaluates the value of the v_appraisal variable based on the value of the v_grade value.
- The output of the above example will be as follows :

Output

```
SQL> /
Enter value for p_grade: a
Grade: A Appraisal Excellent
PL/SQL procedure successfully completed.
SQL> /
Enter value for p_grade: b
Grade: B Appraisal Very Good
PL/SQL procedure successfully completed.
```

2. Searched CASE

PL/SQL also provides a Searched CASE expression, which has the form :

```
CASE
WHEN search_condition1 THEN result1
WHEN search_condition2 THEN result2
...
WHEN search_conditionN THEN resultN
[ELSE resultN+1;]
END;
```

A searched CASE expression has no selector. Also, it's WHEN clauses contain search conditions that yield a Boolean value, not expressions that can yield a value of any type.

Example

If the example is written using a searched CASE expression it will look like this :

```
DECLARE
v_grade CHAR(1) := UPPER('&p_grade');
v_appraisal VARCHAR2(20);
BEGIN
v_appraisal :
CASE
WHEN v_grade = 'A' THEN 'Excellent'
WHEN v_grade = 'B' THEN 'Very Good'
WHEN v_grade = 'C' THEN 'Good'
ELSE 'No such grade'
END;
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || ' Appraisal '
|| v_appraisal);
END;
```

OR Alternative solution can be as follows,

```
DECLARE
v_grade CHAR(1) := UPPER('&p_grade');
BEGIN
CASE
WHEN v_grade = 'A'
THEN DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || '
Appraisal is Excellent');
WHEN v_grade = 'B'
THEN DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || '
Appraisal is Very Good');
WHEN v_grade = 'C'
THEN DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || '
Appraisal is Good');
ELSE DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || '
Appraisal is No such grade');
END CASE;
END;
```

Output

```
SQL> /
Enter value for p_grade: a
Grade: A Appraisal Excellent
PL/SQL procedure successfully completed.
SQL> /
Enter value for p_grade: b
Grade: B Appraisal Very Good
PL/SQL procedure successfully completed.
```

13.3 Iterative Control

- A LOOP statement allows us to execute a sequence of statements multiple times till the given condition is satisfied.
- Looping constructs are the next type of control structure.
- We will place the keyword LOOP before the first statement in the sequence and the keywords END LOOP after the last statement in the sequence it is used as curly brackets{} in high level language.

Types of Iterative Control

1. **Basic loop** to provide repetitive actions without overall conditions
2. **FOR loops** to provide iterative control of actions based on a count
3. **WHILE loops** to provide iterative control of actions based on a condition.

13.3.1 Basic LOOP Statement

(a) Introduction

- The simplest form of Iterative statements is a basic loop, which encloses a sequence of statements between the keywords LOOP and END LOOP.
- As soon as flow of execution reaches the END LOOP statement, control is returned to the corresponding LOOP statement above it.
- A basic loop allows execution of its statement at least once, as like do while statement even if the condition is already met upon entering the loop.
- Without the EXIT statement, the loop will become infinite loop.
- The EXIT Statement,
  1. You can terminate a basic loop using the EXIT statement.
  2. Flow of control is passed to the next statement after the END LOOP statement.

(b) Syntax

```
LOOP                        -- Start of Loop
Statement 1 ;               -- Statement
. . . .
EXIT [WHEN condition]       -- Exit Statement
. . . .
END LOOP;                   -- End of Loop
Where, condition is True, False or Null
```

(c) Examples

Program 13.3.1 : Write a PL/SQL block to generate 10 odd numbers using Basic loop.

```
DECLARE
  v_counter NUMBER:=1;
  v_Number NUMBER:=1;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Before loop:Counter='||
TO_CHAR(v_counter));
  DBMS_OUTPUT.PUT_LINE('Before loop:Number ='||
TO_CHAR(v_Number));
  DBMS_OUTPUT.PUT_LINE('----------');
 LOOP
 EXIT WHEN (v_counter > 10)
   DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_Number));
   v_counter:=v_counter + 1;
   v_Number:=v_Number + 2;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('----------');
  DBMS_OUTPUT.PUT_LINE('After loop:counter='||
TO_CHAR(v_counter));
  DBMS_OUTPUT.PUT_LINE('After loop:Number ='||
TO_CHAR(v_Number));
END;
```

Output

```
SQL> /
Before loop:Counter=1
Before loop:Number =1
1
3
5
7
9
```

```
11
13
15
17
19
----------------------
After loop:v_counter=11
Before loop:Number =21
PL/SQL procedure successfully completed.
```

Program 13.3.2 : Create table part_details then write a PL/SQL block to insert 10 rows to part_details table which have part_id accepted by user and its sequential (i.e 1 to 10) subparts_id .

Query 1 :

```
-------------------------------------------------------------------------
-------------------------------
CREATE TABLE part_details
( part_id    VARCHAR(10),
subpart_id VARCHAR(10))
```

Output

```
SQL> /
Table created.
-------------------------------------------------------------------------
-------------------------------
```

Query 2 :

```
DECLARE
  v_partid   part_details.part_id%TYPE := &Part_Id;
  v_counter NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO part_details(part_id, subpart_id)
    VALUES(v_partid, v_counter);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 10;
  END LOOP;
END;
```

Output

```
SQL> /
PL/SQL procedure successfully completed.
-------------------------------------------------------------------------
-------------------------------
```

Query 3 :

```
SELECT *
FROM    part_details
```

Output

```
SQL> /
PART_ID          SUBPART_ID
----------       ----------
50                1
50                2
50                3
50                4
50                5
50                6
50                7
50                8
50                9
50                10
10 rows selected
```

13.3.2 FOR LOOP Statement

(MSBTE - W-13, W-17)

Q. Explain for loop in PL/SQL with example. **W-13, W-17, 4 Marks**

(a) Introduction

- The FOR loop have the simple syntax as compared to other loops.
- In addition FOR loop have a control statement before LOOP keyword which determine the number of iterations that PL/SQL block performs.
- In case of FOR loop there is no need to declare the counter in code; it is declared implicitly as an integer by system itself.

- The statements in PL/SQL will be executed each time the counter is incremented, which is determined by the two bounds.
- We can refer the counter inside loop only, we can use an expression to reference the existing value of a counter.

(b) Syntax

```
FOR counter in [REVERSE] lower_bound ..upper_bound
LOOP                          -- Start of Loop
Statement 1 ;                 -- Statement  . . . .
                                            . . . .
END LOOP;                     -- End of Loop
```

In the syntax

counter	Is an implicitly declared as a integer Whose value automatically incremented by 1 on each iteration of the loop until the upper or lower limit is reached
REVERSE	The counter will decrement for each iteration from the upper bound to the lower bound
lower_bound	specifies the lower bound for the range of counter values specifies
upper_bound	specifies the upper bound for the range of counter values

(c) Advantages

- Implicit counter declaration in code.
- Each time the counter is incremented or decremented automatically.

(d) Limitations

- Cannot refer the counter as the target of an assignment.
- The lower bound and upper bound of the loop range must be integers.

(e) Examples

Program 13.3.3 : Write a PL/SQL block to generate 10 odd numbers using for loop.

```
DECLARE
v_Number NUMBER:=1;
BEGIN
DBMS_OUTPUT.PUT_LINE('Before loop:v_Number='||
TO_CHAR(v_Number));
FOR i IN 1..10
LOOP
DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_Number));
v_Number := v_Number+2;
END LOOP;
DBMS_OUTPUT.PUT_LINE('After loop:v_Number='||
TO_CHAR(v_Number));
END;
```

Output

```
SQL> /
Before loop:v_Number=1
1
3
5
7
9
11
13
15
17
19
After loop:v_Number=21
PL/SQL procedure successfully completed.
```

Program 13.3.4 : Create table part_details then write a PL/SQL block to insert 10 rows to part_details table which have part_id accepted by user and its sequential (i.e 1 to 10) subparts_id .

Query 1 :

```
--------------------------------------------------------------------
-------------------------------
CREATE TABLE part_details
( part_id   VARCHAR(10),
subpart_id VARCHAR(10))
```

Output

```
SQL> /
Table created.
--------------------------------------------------------------------------------
------------------------------
```

Query 2 :

```
DECLARE
  v_partid   part_details.part_id%TYPE := 50;
  v_counter NUMBER(2) := 1;
BEGIN
FOR i IN 1..10
  LOOP
    v_counter := i;
    INSERT INTO part_details(part_id, subpart_id)
    VALUES(v_partid, v_counter);
  END LOOP;
END;
```

Output

```
SQL> /
PL/SQL procedure successfully completed.
--------------------------------------------------------------------------------
------------------------------
```

Query 3 :

```
SELECT *
FROM    part_details
```

Output

```
SQL> /
PART_ID          SUBPART_ID
----------            ----------
50                    1
50                    2
50                    3
50                    4
50                    5
50                    6
50                    7
50                    8
50                    9
50                    10
10 rows selected
```

Program 13.3.5 : Demonstration of FOR REVERSE Loop to generate 3 numbers.

```
BEGIN
FOR i IN REVERSE 1..3 LOOP
DBMS_OUTPUT.PUT_LINE(TO_CHAR(i));
END LOOP;
END;
```

Output

```
SQL> /
3
2
1
PL/SQL procedure successfully completed.
```

13.3.3 WHILE LOOP Statement

(MSBTE - W- 14)

Q. Explain while-loop in PL/SQL with example. **W-14, 4 Marks**

(a) Introduction

- We can use the WHILE loop to repeat set of statements till the time controlling condition is TRUE as soon as condition becomes FALSE loop will break or exit.
- In case of WHILE condition is evaluated at the start of each iteration.
- The WHILE loop terminates as soon as the condition is FALSE.
- If the condition is FALSE at the start of the loop, then no statement will be executed.
- If the counter variables involved in the conditions do not change inside the body of the loop, then the condition will remains TRUE and the loop does not terminate.

(b) Syntax

```
WHILE condition
LOOP                          -- Start of Loop
Statement 1 ;                 -- Statement
. . . .
. . . .
END LOOP;                     -- End of Loop
```

In the syntax

Condition	This condition is evaluated at the starting of each iteration; this loop will be repeated while a condition is TRUE.

(c) Examples

Program 13.3.6 : Write a PL/SQL block to generate 10 odd numbers using while loop.

```
DECLARE
  v_counter NUMBER:=1;
  v_Number NUMBER:=1;
BEGIN
  DBMS_OUTPUT.PUT_LINE('Before loop:Counter='||
TO_CHAR(v_counter));
  DBMS_OUTPUT.PUT_LINE('Before loop:Number ='||
TO_CHAR(v_Number));
  DBMS_OUTPUT.PUT_LINE('----------');
 WHILE(v_counter<=10)
  LOOP
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(v_Number));
    v_counter:=v_counter + 1;
    v_Number:=v_Number + 2;
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('----------');
  DBMS_OUTPUT.PUT_LINE('After loop:v_counter='||
TO_CHAR(v_counter));
  DBMS_OUTPUT.PUT_LINE('Before loop:Number ='||
TO_CHAR(v_Number));
END;
```

Output

```
SQL> /
Before loop:Counter=1
Before loop:Number =1
1
3
5
7
9
11
13
15
17
19
----------
After loop:v_counter=11
Before loop:Number =21
PL/SQL procedure successfully completed.
```

Program 13.3.7 : Create table part_details then write a PL/SQL block to insert 10 rows to part_details table which have part_id accepted by user and its sequential (i.e 1 to 10) subparts_id .

Query 1 :

```
CREATE TABLE part_details
( part_id    VARCHAR(10),
subpart_id VARCHAR(10))
```

Output

```
SQL> /
Table created.
----------
-
```

Query 2 :

```
DECLARE
  v_partid   part_details.part_id%TYPE := 50;
  v_counter NUMBER(2) := 1;
BEGIN
WHILE v_counter <= 10
  LOOP
    INSERT INTO part_details(part_id, subpart_id)
```

```
  VALUES(v_partid, v_counter);
  v_counter := v_counter + 1;
 END LOOP;
END;
```

Output

```
SQL> /
PL/SQL procedure successfully completed.
--------------------------------------------------------------------------------
------------------------------
```

Query 3 :

```
SELECT   *
FROM     part_details
--------------------------------------------------------------------------------
------------------------------
```

Output

```
--------------------------------------------------------------------------------
------------------------------
SQL> /
PART_ID    SUBPART_ID
----------      ----------
50                 1
50                 2
50                 3
50                 4
50                 5
50                 6
50                 7
50                 8
50                 9
50                 10
10 rows selected
```

13.4 EXIT Statement

(a) Introduction

- The EXIT statement will be execute as either an action within an IF statement or as a single statement inside loop.
- The EXIT statement must be placed inside a loop at the point where we want to break loop.
- We attach a WHEN clause to allow termination of loop based on some condition.
- Whenever the EXIT statement is encountered, the condition inside WHEN clause is evaluated and loop will be terminated if condition is true and control will pass to the next statement after the loop.
- A basic loop may contain more than one EXIT statements.

(b) Syntax

```
LOOP                              -- Start of Loop
Statement 1 ;                     -- Statement
. . . .
IF v_Counter > 10 THEN
EXIT;
END IF;
. . . .
END LOOP;                         -- End of Loop
```

(c) Examples

Program 13.4.1 : Create table part_details then write a PL/SQL block to insert 10 rows to part_details table.

Query 1 :

```
--------------------------------------------------------------------------------
------------------------------
CREATE TABLE part_details
( part_id    VARCHAR(10),
subpart_id VARCHAR(10))
--------------------------------------------------------------------------------
------------------------------
```

Output

```
--------------------------------------------------------------------------------
------------------------------
SQL> /
Table created.
--------------------------------------------------------------------------------
------------------------------
```

Query 2 :

```
DECLARE
  v_partid   part_details.part_id%TYPE := 50;
  v_counter NUMBER(2) := 1;
BEGIN
  LOOP
    INSERT INTO part_details(part_id, subpart_id)
    VALUES(v_partid, v_counter);
    v_counter := v_counter + 1;
    IF  v_counter > 10 THEN
        EXIT;
    END IF;
  END LOOP;
END;
```

Output

```
SQL> /
PL/SQL procedure successfully completed.
-------------------------------------------------------------------
-------------------------------
```

Query 3 :

```
SELECT   *
FROM     part_details
-------------------------------------------------------------------
-------------------------------
```

Output

```
-------------------------------------------------------------------
-------------------------------
SQL> /
PART_ID   SUBPART_ID
----------          ----------
50                  1
50                  2
50                  3
50                  4
50                  5
50                  6
50                  7
50                  8
50                  9
50                  10
10 rows selected
```

13.5 GOTO Statement

(MSBTE - S-17)

Q. Explain GOTO statement with example. **S-17, 4 Marks**

(a) Introduction

- The GOTO statement will be as a unconditional jump statement from loop to any point mentioned by marker.
- The GOTO statement must be placed inside a loop at the point where we want to break loop or continue execution of other part of code.
- We embed GOTO inside IF statement to allow checking of some condition.
- Whenever the GOTO statement is encountered, loop will be terminated control will pass to the marker statement in program.
- Marker statement written as <<..>>
- Use of GOTO statements is generally not recommended as it is unconditional.

(b) Syntax

```
. . . .
LOOP                    -- Start of Loop
Statement 1 ;                   -- Statement
. . . .
IF v_Counter > 10 THEN
GOTO g_marker;
END IF;
. . . .
END LOOP;                       -- End of Loop
<<g_marker>>
. . . .
```

(c) Example

Program 13.5.1 : Write a PL/SQL block to check given number is prime number or not.

```
DECLARE -- declare variables
  p       integer;
  n       PLS_INTEGER := &Number;
BEGIN
-- loop through divisors to determine if a prime number
  FOR i in 2..ROUND(SQRT(n))
  LOOP
    IF n MOD i = 0 THEN       -- test for prime
      p := 0;                   -- not a prime number
      GOTO print_data;
    END IF;
  END LOOP;
  p := 1;
<<print_data>>
IF p=1 THEN
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(n)  ||  '  is  prime number');      -- display data
ELSE
  DBMS_OUTPUT.PUT_LINE(TO_CHAR(n) || ' is not prime number'); -- display data
End IF;
END;
```

Output

```
SQL> /
Enter value for number: 17
17 is a prime number
PL/SQL procedure successfully completed.
SQL> /
Enter value for number: 12
12 is NOT a prime number
PL/SQL procedure successfully completed.
```

13.6 Sample Programs

Program 13.6.1 : Write a PL/SQL block to print given number of odd numbers.

```
-- The following parameter is required for printing DBMS_OUTPUT.
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  v_count  NUMBER(2) := 1;
  v_Num  NUMBER(2)  := &Number;
BEGIN
  WHILE v_count <= v_num
  LOOP
    dbms_output.put_line(2*v_count-1);
    v_count := v_count + 1;
  END LOOP;
END;
```

Output

```
SQL> /
Enter value for num: 5
1
3
5
7
9
PL/SQL procedure successfully completed.
```

Program 13.6.2 : Write a PL/SQL block to determine the salary increment of given employee based on the hire date of the employee.

```
DECLARE
  v_bonus  NUMBER(6,2);
  v_emp_id NUMBER(6) := &Employee_ID;
  v_hire_date DATE;
BEGIN
  SELECT hire_date
   INTO    v_hire_date
  FROM    employees
```

```
WHERE employee_id = v_emp_id;
IF v_hire_date > TO_DATE('01-JAN-98') THEN
   v_bonus := 500;
ELSIF v_hire_date > TO_DATE('01-JAN-96') THEN
   v_bonus := 1000;
ELSE
   v_bonus := 1500;
END IF;
DBMS_OUTPUT.PUT_LINE('Employee Id: ' || emp_id);
                -- display data
DBMS_OUTPUT.PUT_LINE('Employee name: ' || bonus);
                -- display data
END;
```

Program 13.6.3 : Table Place contains columns Room_id, Room_Num, No_Seats, Desc. Write a PL/SQL block to comment on the type of the place depending upon the values of the no. of seats for a given room id.

```
DECLARE
v_RoomId NUMBER := &RoomID;
v_No_Seats  PLACE.NO_SEATS%TYPE ;
BEGIN
SELECT NO_SEATS INTO v_No_Seats
FROM PLACE
WHERE ROOM_ID= v_RoomId;
IF v_No_Seats<3 THEN
DBMS_OUTPUT.PUT_LINE('FAIRLY SMALL ROOM');
ELSIF v_No_Seats<=6 THEN
DBMS_OUTPUT.PUT_LINE('LITTLE BIGGER ROOM');
ELSIF v_No_Seats<=10 THEN
DBMS_OUTPUT.PUT_LINE('LOTS OF ROOM');
END IF;
END;
```

Output

```
SQL> /
Enter value for r: 100
old   5: v_RoomId:= &RoomId;
new   5: v_RoomId:= 100;
FAIRLY SMALL ROOM
PL/SQL procedure successfully completed.
SQL> /
Enter value for r: 300
old   5: v_RoomId:= &RoomId;
new   5: v_RoomId:= 300;
LITTLE BIGGER ROOM
PL/SQL procedure successfully completed.
SQL> /
Enter value for r: 500
old   5: v_RoomId:= &RoomId;
new   5: v_RoomId:= 500;
LOTS OF ROOM
PL/SQL procedure successfully completed.
```

Program 13.6.4 : Table Lecturer contains attributes as Lect_Id, Major_Sub, Course_Name. Write a PL/SQL block to fetch major subject of lecturer from lecturer table and print its name.

```
DECLARE
v_major lecturer.major_sub%type;
v_course_name varchar2(20);
BEGIN
SELECT major_sub INTO v_major
FROM  Lecturer
WHERE  Lect_Id = &Lecturer_Id;
CASE v_major
WHEN 'COMPUTER SCIENCE' THEN
    v_COURSE_NAME := 'CS101';
WHEN 'ECONIMICS' THEN
    v_COURSE_NAME := 'ECN203';
WHEN 'PHYSICS' THEN
    v_COURSE_NAME := 'PHY105';
ELSE
    v_COURSE_NAME := 'UNKNOWN';
END CASE;
DBMS_OUTPUT.PUT_LINE('Major subject Name is '||
v_COURSE_NAME);
END;
```

Program 13.6.5 : Create Table shape contains attributes as Id, Type, Perimeter and Area. Write a PL/SQL block to complete table by accepting required details. lecturer table and print its name.

```
CREATE TABLE Shape
(
ShapId          VARCHAR2(10),
SType           VARCHAR2(20),
Perimeter       NUMBER(10,2),
Area            NUMBER(10,2)
)
SQL> select * from shape
```

Output

SHAPID	STYPE	PERIMETER	AREA
1	Triangle		
2	Circle		
3	Square		

PL/SQL Program

```
DECLARE
  v_shape_id   Shape.ShapeId%TYPE:=&Shape_Id;
  v_Shape_Type     Shape.ShapeType%TYPE;
  v_Var1   Number(10,2);
  v_Var2   Number(10,2);
  v_Area   Shape.Area%TYPE;
  v_Peri   Shape.Perimeter%TYPE;
-- the following is the executable section, from BEGIN to
END
BEGIN
SELECT     ShapeType
INTO       v_Shape_Type
FROM       Shape
WHERE      ShapId = v_shape_id;
IF v_Shape_Type LIKE 'T%' THEN
    v_var1 := '&Base';
    v_var2 := '&Height';
    v_Area := 0.5 * v_var1 * v_var2;
    V_Peri := 3 * v_var1;
    UPDATE     Shape
    SET     Area=v_Area,Perimeter=v_Peri
    WHERE      ShapeId=v_Shape_id;
  DBMS_OUTPUT.PUT_LINE('Shape ID = '|| V_Shape_ID);
  DBMS_OUTPUT.PUT_LINE('Area = '|| V_Area);
  DBMS_OUTPUT.PUT_LINE('Perimeter = '|| V_Perimeter);
ELSIF v_Shape_Type LIKE 'C%' THEN
    v_var1 := '&Radius';
    v_Area := 3.14 * v_var1 * v_var1;
    V_Peri := 2 * 3.14 * v_var1;
    UPDATE     Shape
    SET     Area=v_Area,Perimeter=v_Peri
    WHERE      ShapeId=v_Shape_id;
  DBMS_OUTPUT.PUT_LINE('Shape ID = '|| V_Shape_ID);
  DBMS_OUTPUT.PUT_LINE('Area = '|| V_Area);
  DBMS_OUTPUT.PUT_LINE('Perimeter = '|| V_Perimeter);
ELSIF v_Shape_Type LIKE 'S%' THEN
    v_var1 := '&Length';
    v_Area := v_var1 * v_var1;
    V_Peri := 4 * v_var1;
    UPDATE     Shape
    SET     Area=v_Area,Perimeter=v_Peri
    WHERE      ShapeId=v_Shape_id;
  DBMS_OUTPUT.PUT_LINE('Shape ID = '|| V_Shape_ID);
  DBMS_OUTPUT.PUT_LINE('Area = '|| V_Area);
  DBMS_OUTPUT.PUT_LINE('Perimeter = '|| V_Perimeter);
ELSE
    DBMS_OUTPUT.PUT_LINE('Invalid Shape ID.');
END if;
END;
/
```

Program 13.6.6 : Create table circle. Write a PL/SQL block to generate radius from 0 to all multiples of 5 till 50 and calculate area and perimeter to complete and save circle table.

Query 1

```
1  SQL> create table circle
2  (
3  radius number(10,2),
4  area number(10,2),
5  perimeter number(10,2)
6  )
7  /
```

Output

```
Table created.
```

Query 2

```
DECLARE
 v_radius   circle.radius%TYPE := 0;
 v_area       circle.area%TYPE := 0;
 v_perimeter       circle.perimeter%TYPE := 0;
BEGIN
  WHILE v_radius <= 50
  LOOP
   v_area := 3.14*V_radius*v_radius;
   v_perimeter:=2*3.14*V_radius;
    INSERT INTO circle (radius, area,perimeter)
    VALUES (v_radius, v_area,v_perimeter);
    v_radius := v_radius + 5;
  END LOOP;
  COMMIT;
END;
```

Output

```
SQL> /
PL/SQL procedure successfully completed.
```

Query 3

```
SQL> select *from circle;
```

Output

```
    RADIUS       AREA  PERIMETER
         0          0          0
         5       78.5       31.4
        10        314       62.8

        15      706.5       94.2
        20       1256      125.6
        25     1962.5        157
        30       2826      188.4
        35     3846.5      219.8
        40       5024      251.2
        45     6358.5      282.6
        50       7850        314
11 rows selected.
```

13.7 Exception Handling

(MSBTE- W-18)

Q. Describe exception handling in brief. **W-18, 4 Marks**

Q. What do you mean by Exception? explain with example.

1. Introduction

- All errors that occur in a PL/SQL block or a procedure is known as exceptions.
- Exception means some abnormal condition occurred during execution of a program.
- PL/SQL programs have provision to handle (catch) such abnormal condition with help of EXCEPTION block, program can take an appropriate action against such error condition.

Example

- Divide by zero exception can be determined at runtime only.
- As this error is caused after accepting denominator value as zero from user.

2. Types of Exceptions

(i) Internally defined exceptions

(ii) Predefined exceptions

(iii) User-defined exceptions

(i) Internally Defined Exception

- Oracle has Internally defined exceptions, which are in the form of ORA-n, where -n is a negative number.
- Internally defined exceptions generally do not have any names, also they are raised by Oracle implicitly.

Example

ORA-00001 : is exception for a unique constraint has been violated.

```
DECLARE
  n_counter PLS_INTEGER;
BEGIN
  n_counter := 9999999999; -- overflow number
END;
Output is
ORA-01426: numeric overflow
```

ORA-01426 is the error number and numeric overflow is the error message issued by oracle

(ii) Predefined Exception

- Predefined exceptions are defined internally and have own name, it is declared in the STANDARD package.
- Predefined exceptions are raised implicitly by oracle like internally defined exceptions.
- We can catch predefined exceptions using exception handling block and perform the appropriate actions for it.

Example

Simple PL/SQL Block to increment salary of all employees.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET serveroutput ON;
-- the following is an optional declarative section
DECLARE
  v_num1      NUMBER(6):=5;
  v_num2      NUMBER(6):=0;
-- the following is the executable section, from BEGIN to END
BEGIN
    v_num1 := v_num1 / v_num2;
-- the following displays output from the PL/SQL section
DBMS_OUTPUT.PUT_LINE('Answer is ' ||v_num1);
EXCEPTION
    WHEN ZERO_DIVIDE THEN
    dbms_output.put_line('Error occurred due to divide by zero');
END;
```

Output

```
SQL> /
Error occurred due to divide by zero
PL/SQL procedure successfully completed.
```

(iii) User Defined Exception

- User-defined exceptions are exceptions defined by developers in the declaration section of a PL/SQL block.
- We can customize the action taken if exception occurs during program execution.

3. Trapping an Exception

- If the exception is raised in the executable section of PL/SQL coding block, it will sent the control to corresponding exception handler in the exception handling section of the block.
- If compiler handles the exception successfully, then the exception does not propagate to the database environment.
- The PL/SQL block terminates without any abnormal error.

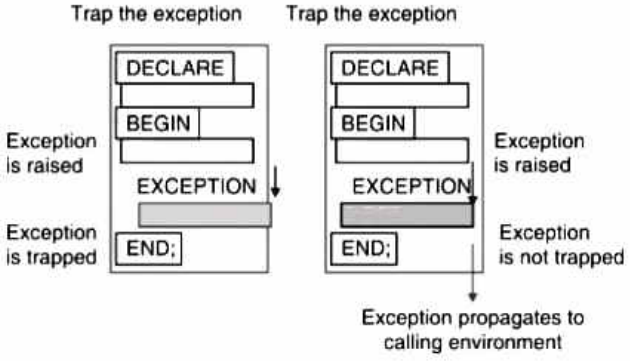


Fig. 13.7.1

4. Propagating an Exception

If the exception is raised in the executable section of the block and there is no corresponding exception handler section available in block, then block terminates with failure and the exception is propagated to database environment.

13.8 Declaring User Defined Exception

Q. Explain how to declare user defined exception.

- User Defined exception is defined in declaration section of PL/SQL program.
- The word EXCEPTION can be used to define the exception.
- The name of exception can be customized by developer and all identifier rules are applicable for exception name.
- Exception is defined like a variable declaration at same section.
- It is possible to trap any error by including a corresponding routine within the exception handling section.
- Each exception handler is consisting of a WHEN clause, which specifies an exception, followed by a multiple statement, which will be executed when that exception is appeared in program.

Exception data

- **SQLCODE**
  It will return the numeric value for the error code which will uniquely identify error.
- **SQLERRM**
  It will return the message associated with the error number which will uniquely identify error.

Syntax

```
DECLARE
  Exception_name  EXCEPTION;
  ....
BEGIN
  ....
EXCEPTION
WHEN exception1 [OR exception2 . . .] THEN
  statement1;
  statement2;
  . . .
[WHEN exception3 [OR exception4 . . .] THEN
  statement1;
  statement2;
  . . .]
[WHEN OTHERS THEN
  statement1;
  statement2;
  . . .]
```

- Where, Exception name can be any name selected by developer.
- EXCEPTION is a keyword used
- OTHERS clause must be placed after all other exception-handling clauses.

13.8.1 User Defined Exception Handling Process

- PL/SQL will allow you to define your own exceptions.
- User-defined exceptions can be declared in the declare section of a PL/SQL block
- We can Raise the exception explicitly with help of RAISE statements.

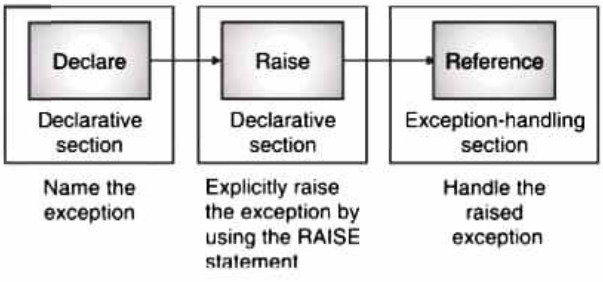


Fig. 13.8.1

Syntax

```
DECLARE
  e_invalid_Student_Number EXCEPTION;
BEGIN
  UPDATE student
SET  descrip = '&Student_name'
```

```
WHERE   prodid = &Student_number;
 IF SQL%NOTFOUND THEN
 RAISE e_invalid_Student_Number;
 END IF;
 COMMIT;
EXCEPTION
 WHEN e_invalid_Student_Number THEN
DBMS_OUTPUT.PUT_LINE('Such Student number not found.');
END;
```

User defined exception handling guidelines

- Exception-handling section of the block will start with the keyword EXCEPTION.
- There can be multiple exception handlers, each has its own set of actions.
- If exception occurs in program, compiler will process only one handler.

Program 13.8.1 : Write a PL/SQL program which accept the customer ID from the user if user enters an invalid ID then the exception invalid_id is raised using exception handling. W-18. 6 Marks

```
DECLARE
c_idcustomers.id%type := &cc_id;
c_namecustomerS.Name%type;
c_addrcustomers.address%type;
  -- user defined exception
invalid_id  EXCEPTION;
BEGIN
  IF c_id<= 0 THEN
    RAISE invalid_id;
  ELSE
    SELECT  name, address INTO  c_name, c_addr
    FROM customers
    WHERE id = c_id;
  DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
   DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
  END IF;
EXCEPTION
  WHEN invalid_id THEN
dbms_output.put_line('ID must be greater than zero!');
  WHEN no_data_found THEN
dbms_output.put_line('No such customer!');
  WHEN others THEN
dbms_output.put_line('Error!');
END;
/
```

Review Questions

Q. 1 What is PL/SQL control structures?

Q. 2 Explain IF statement in details with example.

Q. 3 Explain Decision making statements in details with example.

Q. 4 Explain iterative statements in details with example.

Q. 5 Explain exit statements in PL/SQL with example.

Q. 6 Write short notes on :

(a) FOR Loops (b) CASE Statements

(c) WHILE Loops

Q. 7 Describe the structure of for Loop in PL/SQL. Give example.

❑❑❑

Writing Explicit Cursors

Syllabus

Cursors : Implicit and Explicit Cursors, Declaring, Opening and Closing a cursor,

Fetching a Record from Cursor, Cursor for loops, Parameterized Cursors.

14.1 Concept of Cursor

(MSBTE - W-13, S-15, W-15, S-16, W-16, S-17, W-18)

Q. Define cursor ? List the two types of cursor. **W-13, S-16, W-16, W-18, 2 Marks**

Q. What is cursor ? **S-15, W-15, S-17, 2 Marks**

- A **cursor** is a method by which we can assign a name to a result of SELECT statement and manipulate the data within that SQL statement.
- Oracle server uses SQL work areas or result set to execute SQL statements and store its processing information.
- A PL/SQL cursor allows us to name a work area or result set and accesses its stored information.
- There are two main types of cursors,

1. Implicit Cursor 2. Explicit Cursor

14.1.1 Implicit Cursor

(MSBTE - W-15, S-16, S-17, W-17)

Q. Explain implicit cursors with example. **W-15, S-16, S-17, W-17, 4 Marks**

1) Introduction

- Every SQL DML (data manipulation language) and DRL (Data Retrieval Language - SELECT) statement executed by the Oracle Server is having an individual cursor associated with it.
- PL/SQL implicitly declares a cursor for all SQL SELECT and DML statements, including queries that return only one row.
- An implicit cursor is opened and closed by the oracle server automatically to process each SQL statement.
- For example whenever you execute any SELECT statement or any of INSERT, DELETE or UPDATE statement oracle server automatically declares on cursor associate with it.

2) Implicit Cursor Attributes

- An Oracle server implicitly opens and closes such cursors to process each SQL statement.
- Every SQL implicit cursor has many attributes which returns some useful information about the execution of a data manipulation statement or select statement.

Sr. No.	Attribute
1.	SQL%ISOPEN
2.	SQL%FOUND
3.	SQL%NOTFOUND
4.	SQL%ROWCOUNT

SQL%ISOPEN

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will always returns FALSE, as the database closes the SQL cursor automatically after executing its associated SQL statement.

SQL%FOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns TRUE if an INSERT, UPDATE or DELETE statement is affecting more than one row or a SELECT INTO statement returned more than one row. Otherwise, it returns FALSE.

SQL%NOTFOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns FALSE if an INSERT, UPDATE or DELETE statement is affecting more than one row or a SELECT INTO statement returned more than one row. Otherwise, it returns TRUE.

SQL%ROWCOUNT

- This attribute will return integer output.
- This statement will exactly return the number of rows affected by an DML statement, or returned by a SELECT INTO statement.

3) Example

Program 14.1.1 : Write a PL/SQL block to increment salary of all employees above age 25 in company by 5000 and print number of rows updated by above DML operation.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  v_number        NUMBER(6);
-- the following is the executable section, from BEGIN to END
BEGIN
UPDATE Emp
SET Sal = Sal + 5000
WHERE age > 25;
V_number := SQL%ROWCOUNT ;
-- the following displays output from the PL/SQL section
IF SQL%FOUND THEN
  DBMS_OUTPUT.PUT_LINE('Salary of  ' || V_number || ' Employees updated.');
ELSE
  DBMS_OUTPUT.PUT_LINE('No Employees having age above 25');
END IF;
END;
```

Output

```
SQL> /
Salary of 20 Employees updated.
PL/SQL procedure successfully completed.
```

14.1.2 Explicit Cursor

(MSBTE - W-15, S-16, S-17, W-17, W-18)

Q. Write step by step syntax to create, open and close cursor in PL/SQL block. **W-14, W-18, 4 Marks**

Q. Explain explicit cursors with example. **W-15, S-16, S-17, W-17, 4 Marks**

1) Introduction

- We can use explicit cursors to process each row individually returned by a multiple-row SELECT statement in cursor definition.
- Explicit cursor allows us to name a work area and access its stored information.
- Explicit cursors are used in query which returns many rows.
- The set of rows in table fetched by a cursor query is also called active set.
- The size of the active set is rows returned by a select statement.
- Explicit cursor is declared in the DECLARE section of PL/SQL block.

– The diagram given below shows how an explicit cursor points to the current row in the active data set. So program will process only a single row at a time.

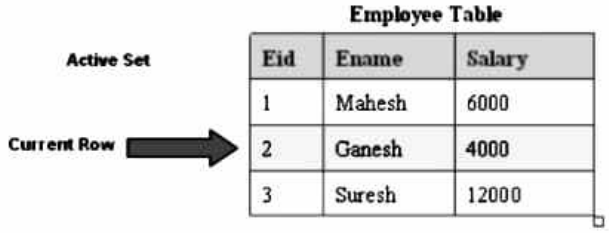


Fig. 14.1.1

2) Explicit Cursor Life cycle / Cursor Implementation

Inside a PL/SQL block we open a cursor, processes rows returned by a query, and then at last closes the cursor.

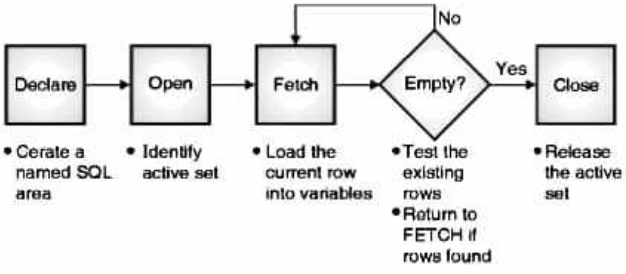


Fig. 14.1.2

(a) Declare the cursor

In declare section we define the structure of the query in cursor.

Syntax

```
CURSOR cursor_name IS select_statement;
```

Example

```
CURSOR Emp_Cur
IS SELECT Eid, Ename FROM Emp;
CURSOR Employee_Cur
IS SELECT * FROM Emp;
```

(b) Open the cursor

– The OPEN statement will execute the query and loads active set.

– After opening cursor, active set is available for fetching.

Syntax

```
OPEN cursor_name;
```

Example

```
OPEN Emp_Cur ;
OPEN Employee_Cur ;
```

(c) Fetch data from the cursor

– After loading data in active set we can fetch row and test the cursor for any existing row.

– We can fetch current row into variables so that it can be accessed and processed.

– If there are no more rows then we need to close the cursor.

Syntax

```
FETCH  cursor_name
INTO   [variable1, variable2, ...] | record_name];
```

cursor_name name of the above declared cursor *variable*
record_name name of the record in which we retrieved data This record variable can be declared with help of %ROWTYPE attribute.

Include the same number of variables in the INTO clause of as columns returned by SELECT statement, and they should be of compatible datatype. We need to match each variable to correspond to the columns positionally.

Example

```
FETCH Emp_cur INTO Eid, Ename;
FETCH Employee_cur INTO Employee_Rec;
```

(d) Close the cursor

– The CLOSE statement will release the current active set and releases memory occupied by active set.

– It is always possible to reopen the cursor.

Syntax

```
CLOSE cursor_name;
```

Example

```
CLOSE Emp_Cur ;
CLOSE Employee_Cur ;
```

3) Explicit Cursor Attributes

- A SQL explicit cursor is opened when we execute OPEN cursor statement.
- Every SQL explicit cursor has many attributes which returns some useful information about cursor state.

Sr. No.	Attribute
1.	Cursor_Name%ISOPEN
2.	Cursor_Name %FOUND
3.	Cursor_Name %NOTFOUND
4.	Cursor_Name %ROWCOUNT

Cursor_Name%ISOPEN

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns TRUE if cursor OPEN statement is already executed or else it will return value FALSE.
- This statement is used to check cursor is open or not.

Cursor_Name%FOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns TRUE if cursor active set is having more than one row to be processed after current row. Otherwise, it returns FALSE.
- This statement is used to check all rows in active set are processed or not.

Cursor_Name%NOTFOUND

- This attribute will return Boolean output i.e. TRUE or FALSE.
- This statement will returns FALSE if cursor active set is having more than one row to be processed after current row. Otherwise, it returns TRUE.
- This statement is used to check all rows in active set are processed or not.

Cursor_Name%ROWCOUNT

- This attribute will return integer output.
- This statement will exactly return the number of rows processed by explicit cursor statement.

4) Example

Program 14.1.2 : Write a PL/SQL block to print employee name and salary of given employee.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
    FROM  EMP
    WHERE  Empno = &Eid;
  v_ename  emp.ename%TYPE;
  v_salary  emp.sal%TYPE;
 -- the following is the executable section, from BEGIN to END
BEGIN
   IF NOT Emp_Cur%ISOPEN THEN
     OPEN Emp_Cur;
   END IF;
  FETCH Emp_Cur INTO v_ename,v_salary;
   IF Emp_Cur%NOTFOUND THEN
     DBMS_OUTPUT.PUT_LINE('No Such Employee Found');
   ELSE
     DBMS_OUTPUT.PUT_LINE('Name='||v_ename||', salary='||v_salary);
   END IF;
END;
```

Output

```
SQL> /
Enter value for eid: 7934
Name=MILLER,salary=1130
PL/SQL procedure successfully completed.
SQL> /
Enter value for eid: 10
No Such Employee Found
PL/SQL procedure successfully completed.
```

14.1.3 Cursor Basic Loops

1) Introduction

- The simplest form of Iterative statements that can be used to process multiple rows in cursor is a basic loop, which encloses a fetch statement and sequence of statements which is to be repeated between the keywords LOOP and END LOOP.
- Without the EXIT statement, the loop will become infinite loop.
- The EXIT Statement
  1. You can terminate a basic loop using the EXIT statement by checking whether it is last row in table.
  2. Flow of control passes to the next statement after the END LOOP statement.

2) Syntax

```
LOOP                                    -- Start of Loop
FETCH   cursor_name
INTO   [variable1, variable2, ...]| record_name];
Statement 1 ;                           -- Statement
. . . .
EXIT [WHEN condition]                   -- Exit Statement
. . . .
END  LOOP;                              -- End of Loop
```

3) Example (Using variables)

Program 14.1.3 : Write a PL/SQL block to print employee name and salary of all employees above given age.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
    FROM   EMP
    WHERE  age > &Age;
  v_ename   emp.ename%TYPE;
  v_salary  emp.sal%TYPE;
 -- the following is the executable section, from BEGIN to END
 BEGIN
   IF NOT Emp_Cur%ISOPEN THEN
     OPEN Emp_Cur;
   END IF;
 DBMS_OUTPUT.PUT_LINE('--------------------------');
 DBMS_OUTPUT.PUT_LINE('Name      | salary');
 DBMS_OUTPUT.PUT_LINE('--------------------------');
 LOOP
   FETCH Emp_Cur INTO v_ename,v_salary;
 EXIT WHEN Emp_Cur%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(RPAD(v_ename,10,'          ')||'|
'||v_salary);
 END LOOP;
 DBMS_OUTPUT.PUT_LINE('--------------------------');
 DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT||'      rows
Selected.');
 DBMS_OUTPUT.PUT_LINE('--------------------------');
END;
```

Output

```
SQL> /
Enter value for age: 25
---------------------------
Name      | salary
---------------------------
SMITH     | 10800
ALLEN     | 11600
WARD      | 11250
JONES     | 12975
MARTIN    | 11250
BLAKE     | 12850
CLARK     | 12450
---------------------------
7 rows Selected.
---------------------------
PL/SQL procedure successfully completed.
```

4) Example (Using Record set)

Program 14.1.4 : Write a PL/SQL block to print employee name and salary of all employees above given age.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
    FROM  EMP
    WHERE age > &Age;
  Emp_Rec  Emp_Cur %ROWTYPE;
-- the following is the executable section, from BEGIN to END
 BEGIN
   IF NOT Emp_Cur%ISOPEN THEN
     OPEN Emp_Cur;
   END IF;
 DBMS_OUTPUT.PUT_LINE('---------------------------');
 DBMS_OUTPUT.PUT_LINE('Name      | salary');
 DBMS_OUTPUT.PUT_LINE('---------------------------');
 LOOP
   FETCH Emp_Cur INTO Emp_Rec;
 EXIT WHEN Emp_Cur%NOTFOUND;
 DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,10,' ')||'|
'||Emp_Rec.sal);
 END LOOP;
 DBMS_OUTPUT.PUT_LINE('---------------------------');
 DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT||'      rows
Selected.');
 DBMS_OUTPUT.PUT_LINE('---------------------------');
END;
```

Output

```
SQL> /
Enter value for age: 25
---------------------------
Name      | salary
---------------------------
SMITH     | 10800
ALLEN     | 11600
WARD      | 11250
JONES     | 12975
MARTIN    | 11250
BLAKE     | 12850
CLARK     | 12450
---------------------------
7 rows Selected.
---------------------------
PL/SQL procedure successfully completed.
```

14.1.4 Cursor While Loops

1) Introduction

– We can use the WHILE loop to repeat FETCH operation till the time controlling condition is TRUE as soon as condition becomes FALSE loop will break or exit.

- WHILE condition is always evaluated at the start of each iteration.
- The WHILE loop terminates as soon as the condition is FALSE.

2) Syntax

```
WHILE condition
LOOP                              -- Start of Loop
FETCH   cursor_name
INTO     [variable1, variable2, ...] | record_name];
Statement 1 ;                     -- Statement
. . . .
END LOOP;                         -- End of Loop
```

In the syntax :

condition	This condition is evaluated at the starting of each iteration; this loop will be repeated while a condition is TRUE.

3) Example

Program 14.1.5 : Write a PL/SQL block to print employee name and salary of all employees above given age.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
    FROM   EMP
    WHERE  age > &Age;
  Emp_Rec   Emp_Cur%ROWTYPE;
-- the following is the executable section, from BEGIN to END
 BEGIN
   IF NOT Emp_Cur%ISOPEN THEN
     OPEN Emp_Cur;
   END IF;
 DBMS_OUTPUT.PUT_LINE('--------------------------');
 DBMS_OUTPUT.PUT_LINE('Name      | salary');
 DBMS_OUTPUT.PUT_LINE('--------------------------');
 WHILE Emp_Cur%FOUND
 LOOP
   FETCH Emp_Cur INTO Emp_Rec;
 DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,10,'  ')||'|
'||Emp_Rec.sal);
 END LOOP;
 DBMS_OUTPUT.PUT_LINE('--------------------------');
 DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT||'      rows
Selected.');
 DBMS_OUTPUT.PUT_LINE('--------------------------');
END;
```

Output

```
SQL> /
Enter value for age: 25
--------------------------
Name      | salary
--------------------------
SMITH     | 10800
ALLEN     | 11600
WARD      | 11250
JONES     | 12975
MARTIN    | 11250
BLAKE     | 12850
CLARK     | 12450
--------------------------
7 rows Selected.
--------------------------
PL/SQL procedure successfully completed.
```

14.2 Cursor FOR Loops

1) Introduction

- FOR loop will have a control statement before LOOP keyword which determine the number of rows fetched by a cursor.
- In case of FOR loop there is no need to declare the record variable in code; it is declared implicitly as a cursor_name%ROWTYPE by system itself.
- We can refer the record variable inside loop only; we can use an expression to reference the existing value of this record set variable.

2) Syntax

```
FOR record_name IN cursor_name
LOOP                          -- Start of Loop
Statement 1 ;                 -- Statement
. . . .
END LOOP;                     -- End of Loop
```

- FOR Loop is shortcut for above type of loops as there is no need to OPEN, CLOSE and FETCH cursor explicitly it will be done automatically.
- Even Record set variable is declared automatically.
- No need to write terminating condition as soon as cursor reaches to last row it will exit from fetch operations and closes cursor automatically.

3) Example

Program 14.2.1 : Write a PL/SQL block to print employee name and salary of all employees above given age.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
    FROM  EMP
    WHERE age > &Age;
-- the following is the executable section, from BEGIN to END
 BEGIN
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  DBMS_OUTPUT.PUT_LINE('Name     | salary');
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  FOR Emp_Rec IN Emp_Cur
  LOOP
  DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,10,' ')||'| '||Emp_Rec.sal);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT||'      rows Selected.');
  DBMS_OUTPUT.PUT_LINE('--------------------------');
END;
```

Output

```
SQL> /
Enter value for age: 25
--------------------------
Name     | salary
--------------------------
SMITH    | 10800
ALLEN    | 11600
WARD     | 11250
JONES    | 12975
MARTIN   | 11250
BLAKE    | 12850
CLARK    | 12450
--------------------------
7 rows Selected.
--------------------------
PL/SQL procedure successfully completed.
```

4) Cursor FOR loops with subquery

- In FOR loop we can declare subquery in place of cursor variable.

– So any query that is solved by cursor can be solved without use of cursor by using cursor query in FOR loop.

5) Syntax

```
FOR record_name IN (SubQuery)
LOOP                          -- Start of Loop
Statement 1 ;                 -- Statement
. . . .
END LOOP;                     --- End of Loop
```

– FOR Loop is shortcut for above type of loops as there is no need to OPEN, CLOSE and FETCH cursor explicitly it will be done automatically.

– Even Record set variable is declared automatically.

– No need to write terminating condition as soon as subquery reaches to last row it will exit from fetch operations.

6) Example

Program 14.2.2 : Write a PL/SQL block to print employee name and salary of all employees above given age.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
BEGIN
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  DBMS_OUTPUT.PUT_LINE('Name      | salary');
  DBMS_OUTPUT.PUT_LINE('--------------------------');
  FOR Emp_Rec IN (SELECT Ename,Sal FROM EMP WHERE age>&Age)
  LOOP
  DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,10,' ')||'| '||Emp_Rec.sal);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('--------------------------');
END;
```

Output

```
SQL> /
Enter value for age: 25
--------------------------
Name      | salary
--------------------------
SMITH     | 10800
ALLEN     | 11600
WARD      | 11250
JONES     | 12975
MARTIN    | 11250
BLAKE     | 12850
CLARK     | 12450
--------------------------
PL/SQL procedure successfully completed.
```

14.3 Parameterized Cursor

(MSBTE-W-15)

Q. Explain parameterized cursor with example. **W-15, 4 Marks**

1) Introduction

– Sometime values for cursor declaration need to pass at runtime such values can be passed while opening cursor using parameter in cursor.

– Parameters allow values to be passed to a cursor while opening it and to be used in the execution of query.

– That is we can open and close an explicit cursor several times in a PL/SQL block and return different active set each time.

– Each of formal parameter in the cursor declaration must have a corresponding actual parameter at the time of OPEN cursor.

– Parameter data types should be same as scalar variables, but there is no need to give them sizes.

– The parameter names are just for references in the query expression of the cursor.

2) Syntax

```
CURSOR cursor_name
            [(parameter_name datatype, ...)]
IS  select_statement;
```

– Cursor_name is a PL/SQL identifier for cursor.

- Parameter_name is the name of a parameter to be passed.
- Datatype is a scalar datatype of the parameter without length.
- Select_statement is a SELECT Query statement

3) Example

Program 14.3.1 : Write a PL/SQL block to print employee name and salary of all employees above given age.

```
-- the following parameter is required for printing DBMS_OUTPUT
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR Emp_Cur (p_age NUMBER)
  IS SELECT Ename,Sal
     FROM  EMP
     WHERE age > p_age;
  Emp_Rec Emp_Cur%ROWTYPE;
-- the following is the executable section, from BEGIN to END
 BEGIN
  IF NOT Emp_Cur%ISOPEN THEN
     OPEN Emp_Cur(&Age);
  END IF;
  DBMS_OUTPUT.PUT_LINE('----------------------------');
  DBMS_OUTPUT.PUT_LINE('Name      | salary');
  DBMS_OUTPUT.PUT_LINE('----------------------------');
  LOOP
    FETCH Emp_Cur INTO Emp_Rec;
  EXIT WHEN Emp_Cur%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE(RPAD(Emp_Rec.ename,10,' ')||'|
'||Emp_Rec.sal);
  END LOOP;
  DBMS_OUTPUT.PUT_LINE('----------------------------');
  DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT||'      rows
Selected.');
  DBMS_OUTPUT.PUT_LINE('----------------------------');
END;
```

Output

```
SQL> /
Enter value for age: 25
----------------------------
Name      | salary
----------------------------
SMITH     | 10800
ALLEN     | 11600
WARD      | 11250
JONES     | 12975
MARTIN    | 11250
BLAKE     | 12850
CLARK     | 12450
----------------------------
7 rows Selected.
----------------------------
PL/SQL procedure successfully completed.
```

14.4 Sample Programs

Program 14.4.1 : Write a PL/SQL block to print employee name and salary and show salary increment of 10% of all employees crossing age of 60.

```
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR emp_cur
  IS
    SELECT empno, sal
    FROM  emp
    WHERE age > 60;
v_empno emp.empno%TYPE;
v_sal emp.sal%TYPE;
BEGIN
OPEN emp_cur;
  DBMS_OUTPUT.PUT_LINE('Empno'||'    '||'sal'||'    '||'Incr.
salary');
```

```
LOOP
  FETCH emp_cur INTO v_empno,v_sal;
  EXIT WHEN emp_cur%NOTFOUND;
  DBMS_OUTPUT.PUT_LINE(v_empno||'                '||v_sal||'
'||1.1*v_sal);
END LOOP;
  DBMS_OUTPUT.PUT_LINE(emp_cur%ROWCOUNT||'  Rows
selected..!');
CLOSE emp_cur;
END;
```

Output

```
SQL> /
Empno        Sal          Incr. salary
7369         10800        11880
7499         11600        12760
7521         11250        12375
7566         12975        14272.5
7654         11250        12375
7698         12850        14135
7782         12450        13695
7788         13000        14300
7839         15000        16500
7844         11500        12650
```

Program 14.4.2 : Write a PL/SQL block to print all employees name with last digit of age is 6.

```
-- The following parameter is required for printing
DBMS_OUTPUT.
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
  CURSOR Emp_Cur
   IS SELECT Ename,Sal
      FROM  EMP;
 BEGIN
  DBMS_OUTPUT.PUT_LINE('Employee Name');
  FOR Emp_Rec IN Emp_Cur
  LOOP
   IF Emp_Rec.age - trunc(Emp_Rec.age,-1) = 6
   THEN DBMS_OUTPUT.PUT_LINE(Emp_Rec.ename);
   END IF;
  END LOOP;
END;
```

Program 14.4.3 : Write a PL/SQL BLOCK to accept radius of circle from user and insert radius and area in area table.

```
create table area
( radius Number(7,2),
Area  Number(7,2)
)
-- The following parameter is required for printing
DBMS_OUTPUT.
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
v_radius area.radius%TYPE:=&Radius;
v_area   area.area%TYPE;
BEGIN
  v_area := 3.14 * v_radius * v_radius;
  INSERT INTO Area(radius,area)
  VALUES (v_radius,v_area);
IF SQL%found Then
DBMS_OUTPUT.PUT_LINE('1 row Inserted.');
END IF;
COMMIT;
END;
```

Program 14.4.4 : Write a implicit cursor to count number of rows updated by update statement.

```
-- The following parameter is required for printing
DBMS_OUTPUT.
SET SERVEROUTPUT ON;
-- the following is an optional declarative section
DECLARE
rows_affected CHAR(4);
BEGIN
  UPDATE emp
  SET sal = sal + 1000;
  rows_affected := to_char(SQL % rowcount);
  IF SQL % rowcount > 0 THEN
```

```
  DBMS_OUTPUT.PUT_LINE(rows_affected);
 ELSE
  DBMS_OUTPUT.PUT_LINE('NOROWS AFFECTED');
 END IF;
END;
```

Review Questions

Q. 1 What is PL/SQL Cursors?

Q. 2 Explain various types of cursors with example.

Q. 3 Explain Implicit cursor attributes in details with example.

Q. 4 Explain cursor attributes in details with example.

Q. 5 Explain cursor for loops in details with example.

Q. 6 Write short notes on :

a) Implicit Cursor. b) Cursor For Loops.

Q. 7 What are the parameterized cursors in PL/SQL ?

❑❑❑

Advanced SQL

Syllabus

Procedures : Advantages, Creating, Executing and Deleting a Stored Procedure.

Functions : Advantages, Creating, Executing and Deleting a Function.

15.1 Stored Procedure

(MSBTE - W-14, S-17)

Q. List advantages of Procedures. **W-14, 4 Marks**

Q. Define following term with example : Procedure **S-17, 4 Marks**

- There are two main categories of data objects that store data and things that access or provide access to data.
  1. Table : ##No content
  2. Views : A view stores select statements that provide a "window" into a table or collection of tables.
- Stored procedures are similar to views.
- Stored procedures are just text objects and don't store any data. They do provide access to data. They can return datasets, just like a view, but that's where the similarity ends.
- A Stored procedure is named PL/SQL block that can take some parameters (referred to as arguments) and be invoked to produce some output.
- Generally speaking you use a procedure to perform a specific action.

Benefits of Stored Procedures

- Stored Procedures offers Modularity of code.
- Procedures promote better reusability and maintainability for code.
- Once validated, they can be used any number of times without compiling again and again in order to make execution faster.
- It can be used in any number of applications to make faster data access.
- If the definition changes only the procedure code is affected. So it will be simple for maintenance.

15.1.1 Creating Stored Procedure

- A stored procedure has a header, a declaration section, an executable section, and an optional exception-handling section.

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter1 [mode1] datatype1,  parameter2 [mode2] datatype2,  . . .)]
IS|AS
PLSQL BLOCK /
[BEGIN PL/SQL Statements; END;]
```

- We can create new procedures with help of the CREATE PROCEDURE statement, which may declare a list of parameters, and also must define the actions to be performed by the standard PL/SQL block.

- Stored procedure blocks can start with either BEGIN or the declaration of local variables and end with either END or END *procedure_name*.
- We cannot reference any host or bind variables inside a stored procedure.
- Using REPLACE option if that procedure exists, it will be dropped and replaced with the new procedure created by the statement.
- Create a procedure to access first and last name of students.

```
CREATE PROCEDURE usp_Test
AS
SELECT fname, lname
FROM   Students
ORDER BY lname
```

- How to Create a Stored Procedure Using Oracle
  a. Enter the text of the CREATE PROCEDURE statement and save it as a script file (*.sql* extension).
  b. From SQL*Plus, run the script file to compile the source code.
  c. Use SHOW ERRORS command to see any compilation errors.
- How to Create a Stored Procedure Using SQL Server
  a. Enter the text of the CREATE PROCEDURE statement in SQL query analyzer.
  b. Run (Press F5 or Run Button) the script file to compile the source code.

15.1.2 Executing Stored Procedure

- We can execute the stored procedure with help of EXEC command or with help of any application program.
- Executing stored procedure with Oracle.

```
EXEC usp_Test
```

- Invoking a Procedure from PL/SQL Block.

```
DECLARE
  ....
BEGIN
  usp_Test;                -- Invoke procedure
  ....
END;
```

- Invoking a procedure from another Procedure.

```
CREATE OR REPLACE PROCEDURE access_stud
IS
BEGIN
  usp_Test;                -- Invoke procedure
  ....
END access_stud;
```

15.1.3 Parameter Types in Stored Procedure

1. Formal parameters

Variables declared in the parameter list of a stored procedure are called as Formal parameters.

Example

```
CREATE PROCEDURE raise_Fees (p_id NUMBER, p_fees
NUMBER)
...
END raise_fees;
```

2. Actual parameters

Variables or expressions referenced in the parameter list of a stored procedure call are nothing but actual parameters.

Example

```
raise_fees ('1001', 2000)
```

15.1.4 Parameter Modes in Stored Procedure

1. IN

- This is Default mode for any variable declared in PLSQL.
- In this type value is passed to stored procedure.
- Formal parameter acts as constant for that procedure.
- Actual parameter for this type of parameter can be a literal, expression, constant.
- We can assign a default value to IN type of variable.

– Write a procedure to increase 10 % fees of student having given id number.

```
CREATE OR REPLACE PROCEDURE raise_fees
(v_id IN stud.studno%TYPE)
IS
BEGIN
      UPDATE stud
      SET fees = fees * 1.10
      WHERE studno = v_id;
END raise_fees;
```

– Executing above stored Procedure

```
In SQL > EXECUTE raise_fees (69)
In PLSQL Block > raise_fees (69)
```

2. OUT

– We must specify OUT to declare such type of variables.

– Such variable Returns value to calling environment

– These are not initialized and cannot have a default value assigned to it.

– Write a procedure access student details having given id number.

```
CREATE OR REPLACE PROCEDURE query_stud
  (p_id IN stud.studno%TYPE,
  p_name OUT stud.name%TYPE,
  p_fees OUT stud.fees%TYPE,
  p_course OUT stud.course%TYPE)
IS
BEGIN
   SELECT name, fees, course
   INTO p_name, p_fees, p_course
   FROM stud
   WHERE studno = p_id;
END query_stud;
```

– Executing above stored Procedure

```
1. Load and run the emp_query.sql script file to create the QUERY_EMP procedure.
2. Declare host variables
3. Execute the QUERY_STUD procedure and print the value of the global G_NAME variable.
  SQL > VARIABLE g_name VARCHAR2(20)
  SQL > VARIABLE g_fees NUMBER
  SQL > VARIABLE g_course NUMBER
  SQL > EXECUTE query_emp(25, :g_name, :g_fees, :g_course)
  SQL > PRINT g_name
  SQL > PRINT g_fees
  SQL > PRINT g_course
```

3. IN OUT

– We must specify IN OUT to declare such type of variables.

– Such variable Passed into procedure and Returns value to calling environment

– These can be initialized and cannot have a default value assigned to it.

– Write a procedure to increase 10 % fees of student having given id number.

```
CREATE OR REPLACE PROCEDURE format_Mob_Num
(p_mobile_no IN OUT VARCHAR2)
IS
BEGIN
   p_mobile_no := '(91) 0' || SUBSTR(p_ mobile _no,1,4) ||
   ' ' || SUBSTR(p_ mobile _no,5,7) ||
   ' ' || SUBSTR(p_ mobile _no,8);
END format_Mob_Num ;
```

– Executing above stored Procedure

```
1. Create a host variable using the VARIABLE command.
2. Then Populate this host variable with a value, using an PL/SQL block.
3. Invoke the format_Mob_Num procedure supplying the host variable as the IN OUT parameter. We will use of the colon (:) to reference the host variable in the EXECUTE command.
4. To view the value passed back to the calling environment we will use the PRINT command.

VARIABLE g_ mob VARCHAR2(15)
BEGIN
```

```
 :g_ mob := '9821666231';
END;
/
PRINT g_ mob
EXECUTE Format_Mob_Num (:g_ mob)
PRINT g_mob
```

15.1.5 Altering Stored Procedure

a. We can overwrite definition of stored procedure by using ALTER command.

b. It also changes procedure details in data dictionary.

c. We can change type of parameter in alter option.

d. If existence of procedure in not known then we will use option CREATE OR REPLACE which will create new procedure if not existing otherwise replace original procedure.

Syntax

```
CREATE OR REPLACE PROCEDURE procedure_name
 [(parameter1 [mode1] datatype1,  parameter2 [mode2] datatype2, . . .)]
IS|AS
PLSQL BLOCK /
[BEGIN PL/SQL Statements; END;]
```

Example

```
CREATE OR REPLACE PROCEDURE format_Mob_Num
  (p_mobile_no IN OUT VARCHAR2)
IS
BEGIN
   p_mobile_no := '(91) 0' || SUBSTR(p_ mobile _no,1,4) ||
   ' ' || SUBSTR(p_ mobile _no,5,7) ||
   ' ' || SUBSTR(p_ mobile _no,8);
END format_Mob_Num ;
```

15.1.6 Dropping Stored Procedure

a. DROP command is used to remove the procedure from database memory.

b. It will remove procedure details from data dictionary also.

Syntax

```
DROP PROCEDURE procedure_name
```

Example

```
DROP PROCEDURE raise_fees
```

15.2 Stored Function

(MSBTE - W-16)

Q. Explain function in PL/SQL with suitable example. **W-16, 4 Marks**

- A stored function is a stored procedure that can accept many inputs and returns only a single value.
- We can make use of stored functions to encapsulate common formulas or business policies that can reused again and again within SQL programs.
- User defined functions are routines that encapsulates SQL logic inside it.
- Like stored procedures User defined functions can also be passed input parameters but user defined functions are compiled and executed at runtime so pretty slower than stored procedures.

Syntax

```
CREATE FUNCTION dbo.Function
(
     /* @parameter1 datatype = default value,
     @parameter2 datatype */
)
RETURNS /* datatype */
DETERMINISTIC
BEGIN
/* sql statement ... */

RETURN
```

```
/* value */

END
```

– Write a sample function program that will print Hello PLSQL by accepting PLSQL as input string.

```
CREATE FUNCTION helloTest (str CHAR(20))
RETURNS CHAR(50) DETERMINISTIC
RETURN CONCAT('Hello, ',str,'!');

Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('PLSQL');
+----------------+
| hello('PLSQL') |
+----------------+
| Hello, PLSQL!  |
+----------------+
1 row in set (0.00 sec)
```

3. Example

Write a function that accepts student credit out of 10 marks and returns grade based on eligibility.

```
CREATE FUNCTION StudentGrd (p_credit double) RETURNS VARCHAR(10)
   DETERMINISTIC
BEGIN
   DECLARE lvl varchar(10);
   IF p_credit >= 7 THEN
      SET lvl = 'A Grade';
   ELSEIF (p_credit >= 5) THEN
      SET lvl = 'B Grade';
   ELSE
      SET lvl = 'C Grade';
   END IF;
RETURN (lvl);
END
```

4. Calling Function from SQL

The Function can be called from SQL code.

```
SELECT    StudName, StudentGrd(credit)
FROM      Student
ORDER BY  customerName;
```

4. Calling Function from SQL Procedure

The Function can be called from SQL procedure,

```
CREATE PROCEDURE GetStudGrd
(
   IN p_studentNumber INT (11),
   OUT p_studGrd  varchar(10)
)
BEGIN
   DECLARE cred DOUBLE;
   SELECT credit INTO cred
   FROM Student
   WHERE StudNumber = p_ studentNumber;
   SELECT StudentGrd(cred)
   INTO p_ studGrd ;
END
```

15.2.1 Advantages of Stored Functions

1. Maintainability

As SQL stored function stored at one location, updates and tracking becomes easier.

2. Testing

Stored function can be tested independently of any other application.

3. Speed of Operation

Stored functions are stored on the server so execution plans for the process are easily reviewable without running the application.

4. Security of code

This system will limit the direct access to tables by defining various user roles in the database, it provides

interface to underlying data structure for implementation.

5. **Limits Direct Access to database**

15.3 Comparison of Stored Procedure and Stored Function

Sr. No.	Stored Procedure	Stored Function
1	Invoke as a PLSQL Block of code or using execute statement	Executed as a part of PLSQL or SQL expression.
2	May not return any parameter	Always return parameter
3	It can return one or more output as variables	It must return atleast one variable.
4	It can contain one or more RETURN statements.	It must contain a single RETURN statement.

Review Questions

Q. 1 Write a short note on stored Procedures.

Q. 2 Write a short note on Procedures.

Q. 3 How to create stored Procedures.

Q. 4 Comparison of stored procedure and stored function.

Q. 5 Explain stored functions.

Q. 6 What are the advantages of stored functions

❑❑❑

CHAPTER 16

UNIT IV

Trigger

Syllabus

Database Triggers : Use of Database Triggers, how to apply database Triggers,

Types of Triggers, Syntax for Creating Trigger, Deleting Trigger

16.1 Trigger

(MSBTE - W-13, S-14, W-14, S-15, S-16, W-16)

Q.	What is database trigger ?	W-13, S-16, 4 Marks
Q.	Explain trigger with suitable example.	S-14, 4 Marks
Q.	What is trigger ? List its types.	W-14, 4 Marks
Q.	How to create trigger ? State any two advantages of trigger.	S-15, 4 Marks
Q.	What is database Trigger ?	W-16, 4 Marks

- A trigger is a procedure that is automatically invoked by the DBMS in response to specific alteration to the database or a table in database.
- Triggers are stored in database as a simple database object.
- A database that has a set of associated triggers is called an active database.
- A database trigger enables DBA (Database Administrators) to create additional relationships between separate databases.

16.1.1 Components of Trigger (E-C-A model) - How to Apply Trigger ?

- Trigger is used as ECA Model (Event-Control-Action Model).
- ECA will explain how to use trigger in applicable scenario.
- **Event (E)** - SQL statement that causes the trigger to fire (or activate). This event may be insert, update or delete operation database table.
- **Condition (C)** - A condition that must be satisfied for execution of trigger.
- **Action (A)** - This is code or statement that execute when triggering condition is satisfied and trigger is activated on database table.

16.1.2 Trigger syntax

```
CREATE [OR REPLACE]
TRIGGER <Trigger_Name>
        [<ENABLE | DISABLE>]
        <BEFORE | AFTER>
        <INSERT | UPDATE | DELETE>
ON      <Table_Name> [FOR EACH ROW]
DECLARE
        <Variable_Definitions>;
BEGIN
        <Trigger_Code>;
END;
```

Table 16.1.1 : Trigger parameters

OR REPLACE	If trigger is already present then drop and recreate the trigger
<Trigger_Name>	Name of trigger to be created.
BEFORE	Indicates that trigger is to be fired before the triggering event occurs.
AFTER	Indicates that trigger is to be fired after the triggering event occurs.
INSERT	Indicates that trigger is to be fired whenever insert statement adds a row to table.
UPDATE	Indicates that trigger is to be fired whenever Update statement modifies a row in a table.
DELETE	Indicates that trigger is to be fired whenever delete statement removes a row from table.
FOR EACH ROW	Trigger will be fired only once for each row.
WHEN	Contains condition that must be satisfied to execute trigger.
<trigger_code>	Code to be executed whenever triggering event occurs

16.1.3 Trigger Types

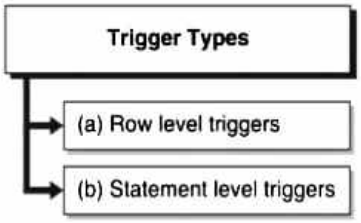


Fig. 16.1.1 : Trigger types

(a) Row level triggers

- A **row level trigger** is fired each time the table is affected by the triggering statement.
- For example, if an UPDATE statement changes multiple rows in a table, a row trigger is fired once for each row affected by the UPDATE statement.
- If triggering statements do not affect any row then a row trigger will not run only.
- If FOR EACH ROW clause is written that means trigger is row level trigger.

(b) Statement level triggers

- A statement level trigger is fired only once on behalf of the triggering statement, irrespective of the number of rows in the table that are affected by the triggering statement
- This trigger executes once even if no rows are affected.
- For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only one time.
- This is Default type, when FOR EACH ROW clause is not written in trigger that means trigger is statement level trigger

Trigger example

- Creating a trigger on employee table whenever new employee added a comment is written to EmpLog Table.
- Let us write a trigger and study its effect.

Example 1

```
SQL> CREATE OR REPLACE TRIGGER AutoRecruit  AFTER
    INSERT ON EMP
        FOR EACH ROW
    BEGIN
        INSERT into EmpLog
        VALUES ('Employee Inserted');
    END;
/
```

Output

```
Trigger created.
SQL> INSERT INTO EMP
        VALUES(1, 'Mahesh', 'Manager', '1-JAN-1986',
        3000, null, 10);
1 row created.
SQL> SELECT *
        FROM    EmpLog;
```

Output

```
STATUS
------------------------------
Employee Inserted
```

- Consider another example, whenever there comes a new student add him to CS (Computer Science).

Example 2

```
SQL> CREATE TRIGGER CSAutoRecruit
        AFTER INSERT
     ON Student FOR EACH ROW
        BEGIN
     INSERT INTO Take VALUES (111, 'CS');
        END;
```

16.1.4 Trigger operations

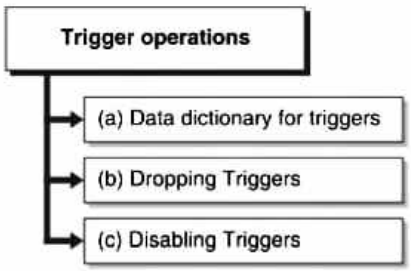


Fig. 16.1.2 : Triggers operations

(a) Data dictionary for triggers

- Once triggers are created their definitions can be viewed by selecting it from system tables as shown as follows :

Syntax

```
MySQL> Select *
       FROM User_Triggers
       Where  Trigger_Name = '<Trigger_Name>';
```

This statement will give you all properties of trigger including trigger code as well.

(b) Dropping Triggers

To remove trigger from database we use command DROP

Syntax

```
MySQL> Drop trigger <Trigger_Name>;
```

(c) Disabling Triggers

To deactivate trigger temporarily this can be activated again by enabling it.

Syntax

```
MySQL> Alter trigger <Trigger_Name>
       {disable | enable};
```

16.2 Trigger Advantages \ Uses of Trigger

1. Triggers are useful for enforcing referential integrity, which preserves the defined relationships between tables when you add, update or delete the rows in those tables. Make sure that a column is filled with default information.
2. After finding that the new information is inconsistent with the database, raise an error that will cause the entire transaction to roll back.

16.3 Trigger Disadvantages

A trigger **hampers the performance of system** as database operations will go on slower due to triggering action.

1. Restrictions on triggers

- You cannot modify the same table on which triggering event is written.
- You cannot modify a table which is connected to the triggering table by primary key, foreign key relation.

2. Mutating table errors

- This happens when the trigger is querying or modifying table whose modification activates the trigger, or a table that might need to be updated because of a foreign key constraint with a CASCADE policy.
- This problem is called as **mutating table problem.**
- **Error** : ORA-04091: table name is mutating, trigger/function may not see it.

16.4 Mutating Table Error

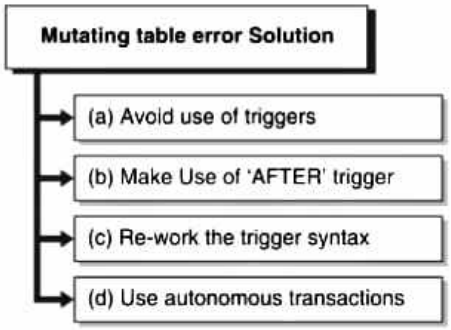


Fig. 16.4.1 : Mutating table error solution

The mutating table error is usually the result of a poor application design and mutating triggers should be avoided whenever possible.

(a) Avoid use of triggers

The best way to avoid the mutating table error is not to use such triggers. While the Oracle provides methods or procedures that are associated with tables, generally PL/SQL developers avoid triggers unless absolutely necessary.

(b) Make use of 'AFTER' trigger

- If use a trigger is must for you, then it is best to avoid the mutating table error by using 'after' trigger, to avoid the currency issues associated with a mutating table.
- For example, using a trigger "after update on salary", the original update has completed and the table will not be mutating.

(c) Re-work the trigger syntax

We can avoid mutating tables with a combination of row-level and statement-level triggers.

(d) Use autonomous transactions

You can avoid the mutating table error by marking your trigger as an autonomous transaction, making it independent from the table that calls the procedure.

Review Questions

Q. 1 What are Triggers ? Give an example.

Q. 2 Explain working of triggers with help of example. In which conditions Triggers cannot be used in database.

Q. 3 What are advantages of triggers ?

Q. 4 What are disadvantages of triggers ?

Q. 5 What are the mutating table errors?

❑❑❑

Database Security

Syllabus

Database Security : Introduction to Database security, Data security requirements, Types of Database Users- Creating, altering and Deleting Users.

Protecting the data within database - Database Privileges : Systems privileges and Object Privileges, Granting and Revoking Privileges : Grant and Revoke command.

17.1 Database Security

(MSBTE - W-13, W-16, W-18)

Q. Explain the database security with its requirements in detail. **W-13, W-16, W-18, 4 Marks**

1. Introduction

A DBMS system always has a separate system for security which is responsible for protecting database against accidental or intentional loss, destruction or misuse.

2. Threats to Database

(a) Confidentiality

- Data in database should be given to only authorized users.
- For e.g. in HR department employee's personnel data should be accessible to that particular employee and the HR person only.

(b) Integrity

- Only authorized users should be allowed to modify data.
- For e.g. only account department can change financial details of company.

(c) Availability

- Authorized users can be able to accesses data any time he wants.
- For e.g. Employee should be able to accesses own salary any time.

3. Security Levels

(a) Database level : DBMS system should ensure that the authorization restriction needs to be there on users.

(b) Operating system level : Operating system should not allow unauthorized users to enter in system.

(c) Network level : Database is at some remote place and it is accessed by users through the network so security is required.

4. Security Requirements

(a) Access Control (Authorization)

- Which identifies valid users who may have any access to the valid data in the Database and which may restrict the operations that the user may perform e.g. ROLE function in SQL.

  For Example : The movie database might designate two roles : "users" (query the data only) and "designers" (add new data) user must be assigned to a role to have the access privileges given to that role.

- Access privileges are assigned to users and roles.
- Each application is associated with a specified role. Each role has a list of authorized users who may execute / Design /administers the application.

(b) Authenticate the User

- Which identify valid users who may have any access to the data in the Database.
- Restrict each user's view of the data in the Database.
- This may be done with help of concept of Views in Relational databases.

(c) Cryptographic control / Data Encryption

- Encode data in a cryptic form (Coded) so that although data is captured by unintentional user still he won't be able to decode the data.
- Used for sensitive data, usually when transmitted over communications links but also may be used to prevent by passing the system to gain access to the data.

(d) Interference Control

- Ensure that confidential information can't be retrieved even by deduction.
- Prevent disclosure of data through statistical summaries of confidential data.

(e) Flow control or Physical Protection

- Prevents the copying of information by unauthorized person.
- Computer systems must be physically secured against any unauthorized entry.

(f) Virus control

- At user level authorization should be done to avoid intruder attacks through humans.
- There should be mechanism for providing protection against data virus.

(g) User defined control

- Define additional constraints or limitations on the use of database.
- These allow developers or programmers to incorporate their own security procedures in addition to above security mechanism.

17.2 Access Control / Authorization in SQL

1. Introduction

- Authorization is finding out if the person, once identified, is permitted to have the resource.
- Authorization explains that what you can do and is handled through the DBMS unless external security procedures are available.
- This is usually determined by finding out if that person is a part of a particular group, if that person has paid admission, or has a particular level of security clearance.
- Authorization is equivalent to checking the guest list at an exclusive party or checking for your ticket when you go to the opera.
- Database management system allows DBA to give different access rights to the users as per their requirements.
- In SQL Authorization can be done by using Read, Insert, Update or Delete privileges.

2. Basic Authorizations

We can use any one form or combination of the following basic forms of authorizations :

(a) Resource authorization

Authorization to access any system resource.
e.g. sharing of database, printer etc.

(b) Alternation Authorization

Authorization to add attributes or delete attributes from relations.

(c) Drop Authorization

Authorization to drop a relation.

3. Discretionary Access control

- This access control is done by two commands GRANT and REVOKE.

- GRANT command helps us to give user privileges to base table and views.
- Above give commands like Grant and Revoke Privilege will be used by this technique for access control.

4. Mandatory Access Control

- This access control mechanisms can be explained as objects (e.g. tables, views, and rows), subjects (e.g. users, programs), security classes and clearances.
- Each database object is assigned a security class, which define its security parameters and each subject is assigned clearance for a security class.
- Comparing discretionary access control and mandatory access control

5. Role Based Access Control

- In Role-Based Access Control (RBAC), access decisions are based on an individual's roles and responsibilities within the organization or user base.
- The process of defining roles is usually based on analyzing the fundamental goals and structure of an organization and is usually linked to the security policy.
- For instance, in an educational organization, the different roles of users may include those such as staff, faculty, attendant, student, principal, etc.
- Obviously, these members require different levels of access in order to perform their functions, but also the types of web transactions and their allowed context vary greatly depending on the security policy and any relevant.

6. Database administrator

- The main authority of database system is database administrator (DBA).
- The SQL standard specifies modification to the schema can be done only by the database owner of schema or DBA of schema.
- The DBA may authorize new users; restructure the database and so on.
- It is analogous to that of **super user** or operator for operating systems.

17.3 Database Users

(MSBTE - S-14, S-17)

Q. List and explain types of DBMS users.

S-14, S-17, 4 Marks

Q. What are the different types of database system users ?

- There are **four** broad categories of users as per their needs to access data :

1. Naive users
2. Application programmers
3. Sophisticated users
4. Specialized users

1. Naive users

- Naive users are users who interact with the system using application programs that have been developed previously.
- For example, Student wants to pay fees Rs. 50 then accountant will invoke a program called fees_payment. This program asks the accountant for the amount of fees to be paid.
- The typical graphical user interface for naive users is a kind of form interface, where the user can fill in appropriate fields of the form.
- A given end user can access the database via one of the applications or can use an interface provided as an integral part of the database system software (such interfaces are also supported by means of applications, of course, but those applications are built-in, not user-written, e.g., query language processor).
- Naive users can read reports generated from the database.

2. Application programmers

- Application programmers responsible for writing application programs that use the database.
- Application programmers are **developers or computer professionals** who write application programs.
- Application programmers develop user interfaces using any preferred language.

- Rapid Application Development (RAD) tools are available now a days that enable an application programmer to construct application without writing code.
- Some programming languages combine control structures with database language statements. Such languages, sometimes called fourth-generation languages.

3. Sophisticated users

- Sophisticated users interact with application without writing programs by using a database query language.
- This query will be solved by query processor.
- Online Analytical Processing (OLAP) tools is used to view summaries of data in different ways which helps analysts (e.g. sales of region, city etc.) with OLAP analysts can use data mining tools, which help them find certain kinds of patterns in data.

4. Specialized users

- Creates the actual database and implements technical controls needed to enforce various policy decisions.
- Specialized users are sophisticated users who develop database applications.
- The DBA is also responsible for ensuring that the system operates with adequate performance and for providing a variety of other related technical services.

17.3.1 Database Administrator (DBA)

- The database administrator is responsible for the overall planning of the company's data resources, for the design of data, and for the day-to-day operational aspects of data management.
- A database administrator is a person responsible for the installation, configuration, up gradation, maintenance and monitoring databases in an organization.
- The overall planning of corporate data is the strategic aspect of the database administration function and involves company-wide planning of existing data and assessment of organization-wise data standards.

Responsibilities of DBA in enterprises

- Designing schema.
- Deciding on the storage and access methods.
- Selecting database software and hardware.
- Designing the means of reorganizing databases periodically.
- Designing database searching strategies.
- Designing authorization checks and validation procedures.
- Designing restart and recovery procedures to take care of system crashes.
- Specifying techniques for monitoring database performance.

- The operations management of database administration deals with data problems arising on a day-to-day basis. Specifically, the responsibilities include:
  - Investigation of errors found in the data.
  - Supervision of restart and recovery procedures in the event of a failure.
  - Supervision of reorganization of databases.
  - Initiation and control of all periodic dumps of data.

Skills required for DBA

- Good communication skills.
- Excellent knowledge of databases architecture and design and RDBMS (Oracle, SQL Server etc.).
- Knowledge of Structured Query Language (SQL).

- In addition, this aspect of database administration includes maintenance of data **security,** which involves maintaining security authorization tables, conducting periodic security audits, investigating all known security breaches.
- To carry out all these functions, it is crucial that the DBA has all the accurate information about the company's data readily on hand. For this purpose, he maintains a data dictionary.

- The data dictionary contains definitions of all data items and structures, the various schemes, the relevant authorization and validation checks and the different mapping definitions.
- It should also have information about the source and destination of a data item and the flow of a data item as it is used by a system. This type of information is a great help to the DBA in maintaining centralized control of data.

17.4 Discretionary Access Control

1. Introduction

- Discretionary access control is based on the concept of access privileges and mechanism for giving users such privileges.
- It grants the privileges to multiple users on different objects, including the access specific records or fields in a specified mode, like read, insert, delete or update or combination of these.
- A user who creates a database object such as a table or a view, they automatically get all applicable privilege for that object.
- The DBMS keeps track of these privileges.

2. Type of actions

- **Account creation** : Account creation action creates a new user account and password for a user or a group of users (Role).
- **Privilege granting** : Privilege granting action permits the DBA to grant (give) certain privileges to certain user or role.
- **Privilege revocation** : Privilege revoking action permits the DBA to revoke (cancel) certain privileges from user or role.
- **Security level assignment** : It will allow user accounts some security level.

17.4.1 Creating a User

Q. Write syntax for creating User with suitable example.

1. Introduction

- In real time database applications having multiple users with same access rights and all needs to have same grant privileges.
- For Example, there are many users who are of type managers and having same access rights on system in a company.
- Instead of giving grant to each and every Manager user we can create a generalized set of grant privileges with name manager and this can be assigned to all employees belongs to category manager.
- Before creating any user, we need to have adequate.

2. Syntax

```
CREATE USER '<User_name>'@'localhost'
IDENTIFIED BY '<Password>'
PASSWORD EXPIRE NEVER;
```

3. Example

We can create user as follows.

```
CREATE USER 'Mahesh'@'localhost'
IDENTIFIED By '123'
PASSWORD EXPIRE INTERVAL 180 DAY;
```

17.4.2 Altering the User

1. Introduction

- An user password can be changed can be deleted from created roles in databases.
- We can also remove role from selected user entered in database system as per authorization required.

2. Syntax

```
ALTER USER <User_name>
IDENTIFIED BY <newPassword>;
```

3. Example

Changing the password of user.

```
ALTER USER 'Mahesh'@'localhost'
IDENTIFIED BY 'newPass';
```

17.4.3 Dropping a User

1. Introduction

- A role can be deleted from created roles in databases.
- We can also remove role from selected user entered in database system as per authorization required.

2. Syntax

```
DROP USER <User_name>
```

3. Example

We can create user as follows

```
CREATE USER 'Mahesh'@'localhost';
```

17.5 Database Privileges

Q. Explain what do mean by database privileges.

Q. Enlist the database privileges with suitable example.

Q. Write a short note on,

1. Ownership Privileges 2. Object Privileges
3. System Privileges

1. Introduction

- The set of actions that a user can perform on a database object are called the privileges.
- Privilege is right to execute particular SQL statement on database.
- The high-level user (Like DBA) has power to grant access to database and its object.

2. System privileges

- System privileges are rights and restriction that are implemented on databases to control which users can access how much data in the database.
- User requires system privileges to gain access to database.
- System privileges are generally provided by DBA.
- Few system privileges are as shown in Table 17.5.1.

Table 17.5.1

System Privileges	Authorized to
CREATE USER	Create number of users in DBMS
DROP USER	Drop any other users in DBMS
CREATE ANY TABLE	Create table object in any schema.
SELECT ANY TABLE	Query table object or view in any schema.
DROP ANY TABLE	Drop table object in any schema.

3. Object privileges

- Object privileges are rights and restrictions to change contents of database objects.
- User requires object privileges to manipulate the content of object within database.
- Once we have created object in a database, after some time there may be few changes needs to be introduced in object.
- Not all database users are allowed to make such changes in database; hence administrator should have control over all objects modification.
- The user which has GRANT ANY PRIVILEGE system privilege granted to him then he can act like administrator to control database modifications.
- Different objects have different privileges assigned for him.
- Few object privileges are as shown in Table 17.5.2.

Table 17.5.2

Object privileges	Authorized to
SELECT	Select rows from table or view
INSERT	Add new rows to table or view
DELETE	Remove some rows from table or view
UPDATE	Modify content of rows from table or view
EXECUTE	To run procedure
REFERENCES	To reference a particular table using foreign key and check constraint

4. Ownership privileges

- Whenever you create a database object (like table or view) with the CREATE statement, you will become its owner and get full privileges for the table. (Like SELECT, INSERT, DELETE, UPDATE, and all other privileges)
- All other users are having no privileges on the newly created database object.
- You as owner of database object can explicitly give grant privileges to any other user by using the GRANT statement.
- Whenever you are creating a view with the CREATE VIEW statement, you become the owner of that view, but you do not necessarily receive all privileges as you require the SELECT privilege on each of base tables on which view is defined.

17.5.1 Granting Privileges

(MSBTE - S-15, S-16, W-16, W-17)

Q. What is the use of GRANT and REVOKE ? **S-15, 4 Marks**

Q. Describe Grant and Revoke commands. **S-16, W-17, 4 Marks**

Q. Give the use of grant and revoke command with syntax and example. **W-16, 4 Marks**

1. Introduction

- A system privilege is the right to perform a particular action, or to perform an action on any schema objects of a particular type.
- An authorized user may pass on this authorization to other users. This process is called as granting of privileges
- Generally GRANT statement is used by owner of table or view to give other users access permissions.
- In SQL user accounts must present in system before we can grant privileges to him.

2. Syntax

```
Grant  <ALL | privilege list>
ON     <relation name or view name>
TO     <user | role list | PUBLIC >,
       [WITH GRANT OPTION]
```

Table 17.5.3

Privilege list	Meaning
ALTER	Table and views
CREATE	Table and views
DROP	Table and views
DELETE	Tables and views
INSERT	Tables and views
SELECT	Tables and views
UPDATE	Tables and views
ALL	Tables and views

WITH GRANT OPTION

Is used to allow user to grant privileges (which are granted to him) to other users.

3. Example

- Consider an example for granting update authorization to the *Emp_Salary* relation of the company database. Assume that, initially that the DBA grants update authorization on *Emp_Salary* to other users U1, U2 and U3, who may in turn pass on this authorization to other users. This passing of authorization from one user to other users is called **authorization graph**.
- The following **grant** statement grants user U1, U2 and U3 the **select** privilege on *Emp_Salary* relation :

```
GRANT SELECT, INSERT
ON    mydb.*
TO    'mahesh'@'somehost';
```

- Following **grant** statement gives all users all authorization on the amount attributes of the *Emp_Salary* relation using public keyword;

```
GRANT ALL
ON    *.*
TO    'mahesh'@'somehost';
```

Some other types of privileges :

(a) Database privileges

- SQL permits a user to declare foreign keys while creating relations.
- **Example** Allow user U1 to create relation that references key 'Eid' of Emp_Salary relation.

```
GRANT   ALL
ON      mydb.*
TO      'mahesh'@'somehost';
```

(b) Table privileges

- This privilege authorizes a user to execute a function or procedure.

```
GRANT   ALL
ON      mydb.mytbl
TO      'mahesh'@'somehost';
GRANT   SELECT, INSERT
ON      mydb.mytbl
TO      'mahesh'@'somehost';
```

(c) Column privileges

- This privilege authorizes a user to execute a function or procedure.

```
GRANT   SELECT (col1), INSERT (col1, col2)
ON      mydb.mytbl
TO      'mahesh'@'somehost';
```

17.5.2 Revoking of Privileges

(MSBTE - S-15, S-16, W-16, W-17)

Q. What is the use of GRANT and REVOKE ? **S-15, 4 Marks**

Q. Describe Grant and Revoke commands. **S-16, W-17, 4 Marks**

Q. Give the use of grant and revoke command with syntax and example. **W-16, 4 Marks**

1. Introduction

- We can reject the privileges given to particular user with help of revoke statement.
- To revoke an authorization, we use the **revoke** statement.

2. Syntax

```
REVOKE   <ALL | privilege list>
ON       <relation name or view name>
FROM     <user | role list | PUBLIC>
         [RESTRICT/ CASCADE]
```

- **CASCADE** : Will revoke all privileges along with all dependent grant privileges
- **RESTRICT** : This will not revoke all related grants only removes that GRANT only.

3. Examples

The revocation of privileges from user or role may cause other user or roles also have to leave that privilege. This behaviour is called cascading of the revoke.

(a) To remove select privilege from users U1, U2 and U3.

```
REVOKE   select
ON       mydb.mytbl
FROM     'mahesh'@'somehost';
```

(b) To remove update rights on amount column of Emp_Salary from U1, U2 and U3.

```
REVOKE   update (amount)
ON       Emp_Salary
FROM     'mahesh'@'somehost';
```

(c) To remove reference right on amount column from user U1.

```
REVOKE   references (amount)
ON       Emp_Salary
FROM     'mahesh'@'somehost';
```

The revoke statements may alternatively specify **restrict** if we don't want cascade behaviour.

```
REVOKE   select
ON       Emp_Salary
FROM     'mahesh'@'somehost'
         RESTRICT
```

Ex. 17.5.1 :

i) Create user 'Rahul'.

ii) Grant create, select, insert, update, delete, drop privilege to'Rahul'.

iii) Removes the select privilege from user 'Rahul'

W-18. 6 Marks

Soln. :

```
(i) CREATE user Rahul
                Identified by 'abc'
(ii) GRANT create, select, insert, update, delete, drop ON
          emp
     To      rahul;
(iii) REVOKE Select
     ON      Emp
     FROM    Rahul;
```

17.6 Viewing Privileges

Q. What is Viewing Privileges? Write syntax with example?

1. Introduction

- In order to view all rights and previledges granted on specific data objects, we may need a metadata which keeps track of all data objects in databse.
- To view priviledges in MySQL we use data dictionary.

2. Syntax

```
SELECT    *
FROM      information_schema.user_privileges
```

3. Examples

- The revocation of privileges from user or role may cause other user or roles also have to leave that privilege.
- To view privileges given to table.

```
SELECT    *
FROM      information_schema.user_privileges
WHERE     grantee like " 'user'% "
```

Review Questions

Q. 1 Write syntax for creating USER with suitable example.

Q. 2 Explain data privileges.

Q. 3 Enlist the privileges with suitable example.

Q. 4 Write syntax for GRANT privileges.

Q. 5 Write a short note on Revoking of privileges.

Q. 6 Write a short note on : .

1. Ownership Privileges
2. Object Privileges
3. System Privileges

Q. 7 What is Viewing Privileges ? Write syntax with example.

Q. 8 Explain database security.

Q. 9 Short note on Access control.

Q. 10 Explain the database users.

Q. 11 Explain discretionary access control.

❑❑❑

Transaction Processing

Syllabus

Transaction : Concept, Properties and States of Transaction.

18.1 Concept of Transaction

Q. What is transaction? Explain concept of transaction using example.

- Single SQL command is sent to database server as a query and server will reply with answer.
- Multiple SQL commands (DML,DRL etc.) are sent to database server which executed one after other (as shown in Fig. 18.1.1)

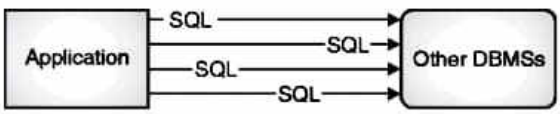


Fig. 18.1.1 : Executing single operation in DBMS

- In place of sending one by one SQL command to server we can combine multiple operations that are logically similar and send to serve as a single logical unit called **transaction.**

Example :

Transferring Rs. 100 from one account to other

1. Withdraw Rs.100 from account_1
2. Deposit Rs.100 to account_2

- Simple query fired on DBMS is called **SQL operation.**
- Collection of multiple operations that forms a single logical unit is called as **transaction.**
- A transaction is a sequence of one or more SQL statements that combined together to form a single logical unit of work.

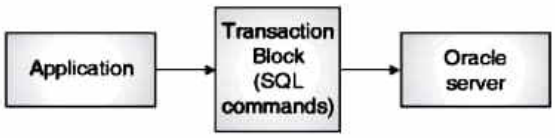


Fig. 18.1.2 : Executing transaction in DBMS

Types of operations

Types of operations that can be done inside transaction,

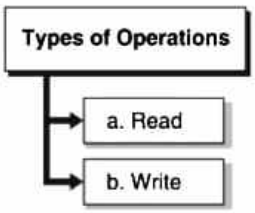


Fig. 18.1.3 : Types of operations

1. Read operation

Read operation transfers data item from (e.g. table) the database memory to a local buffer of the transaction that executed the read operation.

Example : Data selection / retrieval language.

```
SELECT  *
FROM    Students
```

2. Write operation

- Write operation transfer data from local memory to database.
- Write operation transfers one data item from the local buffer of the transaction that executed the write back to the database.

Example : Data Manipulation Language (DML)

```
UPDATE Student
SET    Name = 'Bhavna'
WHERE  Sid = 1186
```

- Information processing in DBMS divides operation individual, indivisible operational logical units, called transactions.
- A transaction is a sequence of small database operations.
- Transactions will execute to complete all set of operations successfully.

Example : During the transfer of money between two bank accounts it is problematic if one of transaction fails.

```
BEGIN TRANSACTION transfer
    UPDATE accounts
    SET balance=balance – 100
    WHERE account=A

    UPDATE accounts
    SET balance=balance + 100
    WHERE account=B
If no errors then
    Commit Transaction
Else
    Rollback Transaction
End If
END TRANSACTION transfer
```

Fig. 18.1.4 : Sample implementation of transaction in SQL

18.1.1 Transaction Structure and Boundaries

1. Transaction structure

- The transaction consists of all SQL operations executed between the begin transaction and end transaction.
- A transaction starts with a BEGIN transaction command.
- BEGIN command instructs transaction monitor to start monitoring the transaction status.
- All operations done after a BEGIN command is treated as a single large operation.

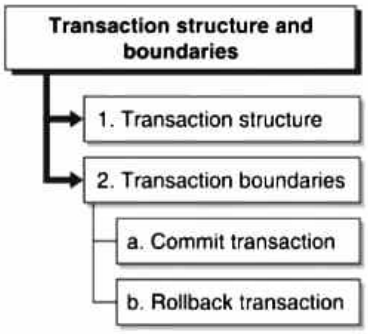
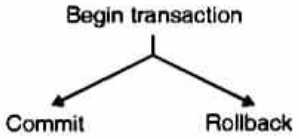


Fig. 18.1.5 : Transaction structure and boundaries

2. Transaction boundaries

- Transaction must ends either by executing a COMMIT or ROLLBACK command.
- Until a transaction commits or rolls back the data in database remains unchanged.



(a) Commit transaction

All successful transactions are required to be committed by issuing commit command. To make all changes permanent and available to other users of the database.

(b) Rollback transaction

- If transaction is unsuccessful due to some error then it must be rolled back.
- Roll back command will remove all changes to the database which are undone and the database remains unchanged by effect of transaction.
- The DBMS should take care of transaction it should be either complete or fail.
- When a transaction fails all its operations and the database is returned to the original state it was in before the transaction started.

18.2 Fundamental Properties of Transaction / ACID Properties

(MSBTE - S-15, W-15, S-16, W-17, W-18)

Q. Explain ACID properties of transaction.

S-15, W-15, S-16, W-17, W-18, 4 Marks

To understand transaction properties, we consider a transaction of transferring 100 rupees from account A to account B as below.

Let T_i be a transaction that transfers 100 from account A to account B. This transaction can be defined as,

(a) Read balance of account A.

(b) Withdraw 100 rupees from account A and write back result of balance update.

(c) Read balance of account B.

(d) Deposit 100 rupees to account B and write back result of balance update.

T_i	Read (A);
	A := A – 100;
	Write(A);
	Read(B);
	B := B + 100;
	Write(B)

Fig. 18.2.1 : Sample transaction

Transaction Properties are as follows :

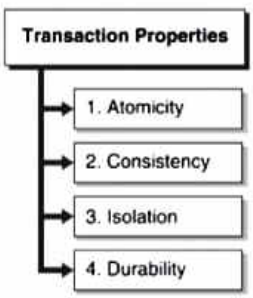


Fig. 18.2.2 : Transaction Properties

1. Atomicity

- Transaction must be treated as a single unit of operation.
- That means when a sequence of operations is performed in a single transaction, they are treated as a single large operation.

Examples

(a) Withdrawing money from your account.

(b) Making an airline reservation.

- The term atomic means thing that cannot be divided in parts as in atomic physics.
- Execution of a transaction should be either complete or nothing should be executed at all.
- No partial transaction executions are allowed. (No half done transactions)

Example 1 : Money transfer in above example

Suppose some type of failure occurs after Write (A) but before Write (B) then system may lose 100 rupees in calculation which may cause error as sum of original balance(A+B) in accounts A and B is not preserved (such situation is called as Inconsistency explained later), In such case database should automatically restore original value of data items.

Example 2 : Making an airline reservation

(a) Check availability of seats in desired flight.

(b) Airline confirms your reservation

(c) Reduces number of available seats

(d) Charges your credit card (deduct amount from your balance)

(e) Increases number of meals loaded on flight (Sometimes)

- In above case either all above changes are made to database or nothing should be done as half-done transaction may leave data as incomplete state.
- If one part of the transaction fails, the entire transaction fails and the database state is left unchanged.

2. Consistency

- Consistent state is a database state in which all valid data will be written to the database.
- If a transaction violates some consistency rules, the whole transaction will be rolled back and the database will be restored to its previous consistent state with those rules.

- On the other hand, if a transaction is executed successfully then it will take the database from one consistent state to another consistent state.
- DBMS should handle an inconsistency and also ensure that the database is clean at the end of each transaction.
- Consistency means transaction will never leave your database in a half finished (inconsistent) state.
- If one part of the transaction fails, all of the pending changes made by that transaction are rolled back.

Example : Money transfer in above example

- Initially in total balance in accounts A is 1000 and B is 5000 so sum of balance in both accounts is 6000 and while carrying out above transaction some type of failure occurs after Write (A) but before Write (B) then system may lose 100 rupees in calculation.
- As now sum of balance in both accounts is 5900 (which should be 6000) which is not a consistent result which introduces inconsistency in database.
- This means that during a transaction the database may not be consistent.

3. Isolation

- Isolation property ensures that each transaction must remain unaware of other concurrently executing transactions.
- Isolation property keeps multiple transactions separated from each other until they are completed.
- Operations occurring in a transaction (example, insert or update statements) are invisible to other transactions until the transaction commits (on transaction success) or rolls back (on transaction fails).
- For example, when a transaction changes a bank account balance other transactions cannot see the new balance until the transaction commits.
- Different isolation levels can be set to modify this default behaviour.
- Transaction isolation is generally configurable in a variety of modes. For example, in one mode, a transaction locks until the other transaction finishes.
- Even though many transactions may execute concurrently in the system. System must guarantees that, for every transactions (T_i) all other transactions has finished before transactions (T_i) started, or other transactions are started execution after transactions (T_i) finished.
- That means, each transaction is unaware of other transactions executing in the system simultaneously.

Example : Money transfer in above example.

The database is temporarily inconsistent while above transaction is executing, with the deducted total written to A and the increased total is not written to account B. If some other concurrently running transaction reads balance of account A and B at this intermediate point and computes A+B, it will observe an inconsistent value (Rs. 5900). If that other transaction wants to perform updates on accounts A and B based on the inconsistent values (Rs. 5900) that it read, the database may be left database in an inconsistent state even after both transactions have completed.

A way to avoid the problem of concurrently executing transactions is to execute one transaction at a time.

4. Durability

- The results of a transaction that has been successfully committed to the database will remain unchanged even after database fails.
- Changes made during a transaction are permanent once the transaction commits.
- Even if the database server is fails in the between transaction, it will return to a consistent state when it is restarted.
- The database handles durability by transaction log.
- Once the execution of the above transaction completes successfully, and the user will be notified that the transfer of amount has taken place, if there is no system failure in this transfer of funds.
- The durability property guarantees that, all the changes made by transaction on the database will be available permanently, although there is any type of failure after the transaction completes execution.

18.3 Transaction States

(MSBTE – S-14, S-15, S-16, S-17, W-17)

Q. Explain states of transaction with neat diagram. **S-14, 4 Marks**

Q. Draw transaction state diagram. **S-15, S-16, W-17, 2 Marks**

Q. Describe state of transaction with neat diagram. **S-17, 4 Marks**

- If transaction complete successfully it may be saved on to database server called as **committed** transaction.
- A committed transaction transfers database from one consistent state to other consistent state, which must persist even if there is a system failure.
- Transaction may not always complete its execution successfully. Such a transaction must be **aborted**.
- An aborted transaction must not have any change that the aborted transaction made to the database. Such changes must be undone or **rolled back**.
- Once a transaction is committed, we cannot undo its effects by aborting it.
- A transaction must be in one of the following states :

1. Active

- This is initial state of transaction execution.
- As soon as transaction execution starts it is in active state.
- Transaction remains in this state till transaction finishes.

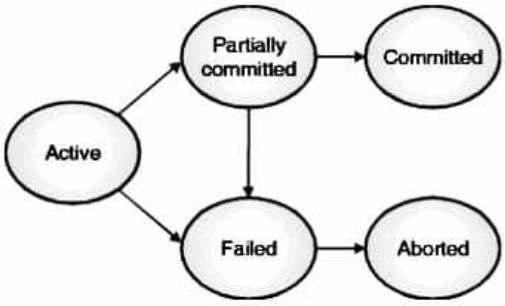


Fig. 18.3.1 : State diagram of a transaction

2. Partially committed

- As soon as last operation in transaction is executed transaction goes to partially committed state.
- At this condition the transaction has completed its execution and ready to commit on database server. But, it is still possible that it may be aborted, as the actual output may be there in main memory, and thus a hardware failure may prohibit its successful completion.

3. Failed

A transaction enters the failed state after the system determines that the transaction can no longer proceed with its normal execution.

Example : In case of Hardware or logical errors occurs while execution.

4. Aborted

- Failed transaction must be rolled back. Then, it enters the aborted state.
- Transaction has been rolled back restoring into prior state.
- In this stage system have two options :
  - **Restart the transaction** : A restarted transaction is considered to be a new transaction which may recover from possible failure.
  - **Kill the transaction** : Because the bad input or because the desired data were not present in the database an error can occurs. In this case we can kill transaction to recover from failure.

5. Committed

- When last data item is written out, the transaction enters into the committed state.
- This state occurs after successful completion of transaction.
- A transaction is said to have terminated if has either committed or aborted.

Review Questions

Q. 1 What is transaction? Explain concept of transaction using example.

Q. 2 Explain ACID properties of transaction.

Q. 3 Describe state of transaction with neat diagram.

Q. 4 State and explain Transaction structure.

❑❑❑

Database Backup and Recovery

Syllabus

Database Backup : Types of failures, Causes of failures, Database Backup introduction, Types of Database Backups-Physical and Logical.

Database Recovery : Recovery concept, Recovery Techniques-Roll forward, Rollback

19.1 Database System Failure

Q. Explain various types of system failure.

- Like any other real-world database systems also suffer from failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room even rain. The result of any failure is loss of information.
- Therefore, the database system must take actions in advance to ensure that the atomicity and durability properties of transactions are preserved.
- An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure.
- Provide high availability; that is, it must minimize the time for which the database is not usable after a crash.

19.1.1 Types and Causes of Failure

Q. Explain system failure classification.

- A computer system, like any other electrical or mechanical system is tends to failure. There are many causes, including disk crash, power failure, software errors, a fire in the machine room, or even sabotage. Whatever the cause, information may be lost.
- There are various types of failure that may occur in a system, each of these needs to be dealt with a different manner.

1. Hardware Failure / System crash

- There is a hardware malfunction that causes the loss of the content of volatile storage, and brings transaction processing to a halt.
- The content of non-volatile storage remains intact, and is not corrupted or changed.

2. Software Failure

The database software or the operating system may be corrupted or failed to work correctly, that may causes the loss of the content of volatile storage, and brings about database failure.

3. Media failure

- A disk block loses its content as a result of either a head crash or failure during a data-transfer operation.
- Copies of the data on other disks such as tapes, CDs are used to recover from the failure.

4. Network Failure

- The problem with network interface card can cause network failure.
- There may be problem with network connection.

5. Transaction failure

There are two types of errors that may cause a transaction to fail :

(a) Logical error

The transaction can no longer continue with its normal execution because of some internal condition, such as wrong input values, data not found in database, data overflow, or resource limit exceeded etc.

(b) System error

The system has entered an undesirable state like deadlock as a result, transaction cannot continue with its normal execution.

6. Application software error

- The problem with software accessing the data from database.
- This may cause database failure as data cannot be updated using such application to it..

7. Physical disasters

The problem caused due to flood, fire, earthquake etc.

8. Application software error

These are some logical errors in the program that is accessing database, which cause one or more transactions failure.

19.1.2 Database Backup

- Database recovery is the process of restoring the database to original (correct) state as it was before database failure occurs.
- The process of solving any type of database failures, quickly and without data loss and keep database available is called database recovery.

Database Backup Concept

- Everyone can lose something during his life. He may lose money, lose heart, and lose his friend similarly computer also can lose something as well. Suppose your computer crashed one day and all your important data is lost, and could never be recovered? Though this does not necessarily mean the end of the world for you, but it does make lots of trouble for us.
- Databases have become an integral part of the information systems of many organizations. So, it is very important to preserve our valuable data as entire business is depending on it.
- Backup does not take much time, as it is easy to put your files onto another medium, and is not too expensive; depending upon what mediums you use to backup your data.

1. Database Backup

- Data backup do mean by storing all our files from our computer in another location. In this way, if there is ever any loss of data on your primary machine, you still have copy of same data with us on another machine.
- Data backup is easy to do and can save you great amounts of time as well as ensure that your data is secure in the case of disaster.

2. Database Backup Techniques

(i) Full Backup - Physical Backup

- Full database backup is maintained at one recovery site as backup copy of that site.
- A full and incremental database back up option targets to make it more feasible to store several copies of the source data.
- At first step a full backup (of all files) is made. After that, any number of updates can be applied to that as incremental backup.

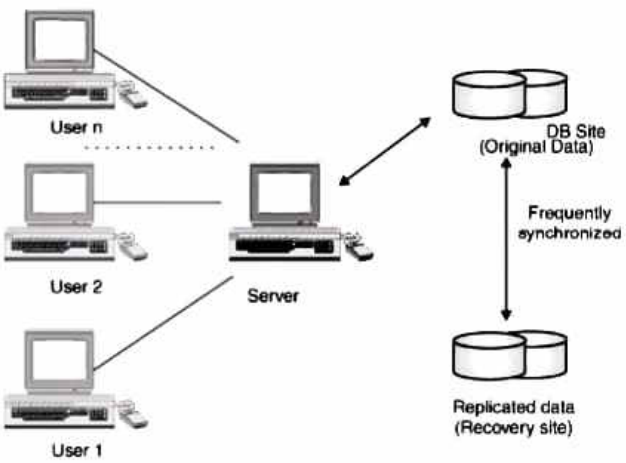


Fig. 19.1.1 : Physical Backup

(ii) Differential Backup

- A differential backup copies files that have been created or changed since the last full backup.
- Only last modification done in data will be saved as a backup.
- It does not make back up of files already backed up on system.
- If you are performing a combination of full and differential backups, restoring files and folders requires that you have the last full as well as the last differential backup.

(iii) Partial Backup (Log Based Backup) (or Logical Backup)

- We will save log file for all transactions done on databases as a backup.

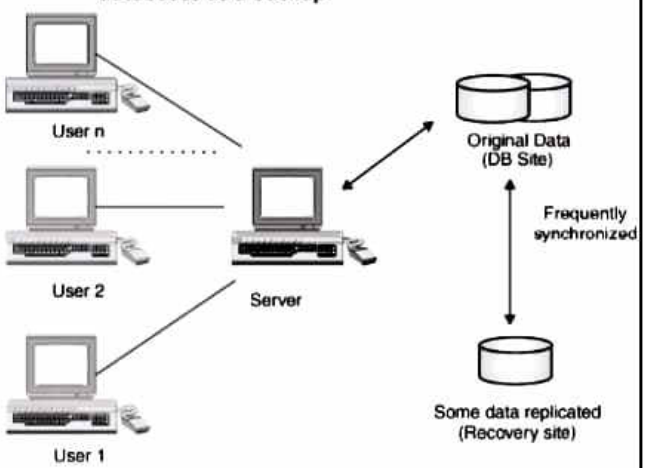


Fig. 19.1.2 : Logical Backup

19.2 Database Recovery Concepts

Q. Explain concept of database recovery and also explain the techniques.

- Database recovery is the process of restoring the database to original (correct) state as it was before database failure occurs.
- The process of solving any type of database failures, quickly and without data loss and keep database available is called database recovery.
- The main element of database recovery is the most recent database backup. If you maintain database backup efficiently, then database recovery is very straight forward process.

Example : To recover data from system having full backup option is as below,

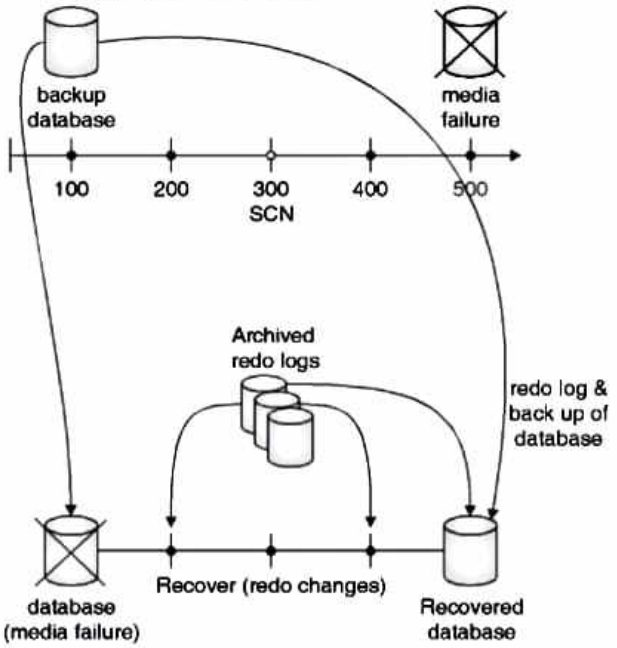


Fig. 19.2.1 : Database Recovery Concepts

Restoring entire system to a certain point may require time, depends on when last full backup taken and incremental backups which covers the period of time between the full backup and restore point.

Database recovery algorithms

(a) During normal transaction executions ensure that enough information is backed up to allow recovery from possible failures.

(b) Data should ensure that database consistency, transaction atomicity, and durability.

Types of database recovery / recovery phases

(a) Forward database recovery / Roll Forward (REDO Phase)

(b) Backward database recovery / Roll Backward (UNDO Phase)

Database recovery techniques

(a) Log based recovery

(b) Shadow paging recovery

(c) Checkpoints

19.3 Log-Based Recovery

Q. Explain concept of log and log based recovery algorithm.

(1) Introduction

- There can be problem in accessing database due to any reason can causes a database system failure.
- The most widely used structure for recording database modifications is **Transaction log (or log).**
- The log is a sequence of log records, recording all the update activities done on the database by all database users.

(2) Transaction log

- Transaction log records are maintained to record various events during transaction processing.
- Transaction log file is recorded when transaction performs a write operation to record data changes.
- Log records to be useful for recovery from system and disk failures, the log must reside in stable storage.
- We assume that every log record is written to the end of the operation on stable storage as soon as log is created.
- Transactional log contains following data :
  - (i) **Transaction Identifier (T_i) :** This is the unique identifier of the transaction that is recorded at write operation.
  - (ii) **Data item Identifier(X) :** Unique identifier to recognize data item written.
  - (iii) **Old Value (V_1) :** Value of the old data item.
  - (iv) **New Value (V_2) :** Value of new data item after the write is done.

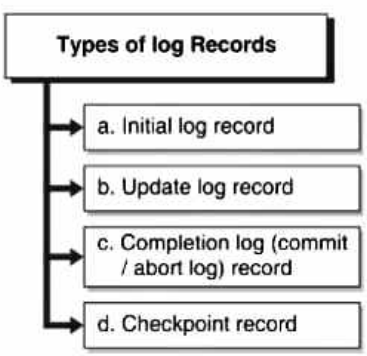


Fig. 19.3.1 : Types of log records

(a) Initial log record

- To start a transaction initial transaction log is recorded.
- This log indicates that recording log file is started.

Log Record : < T_n Start>

Transaction (T_n) has started.

Example :

<T_1 Start> : Transaction T_1 is started.

(b) Update log record

- An update log record describes a single database write.
- It also includes the value of the bytes of page before and after the page change.

Log record : < T_n, X, V_1, V_2>

Transaction (T_n) has performed a write on data item X.

- It modify current value V_1 of data item X to updated value V_2 after the write.

Example :

<T_1 , A, 100,500> : Transaction T_1 changed value of A to 500.

Or <T_1 , A, 500> : Transaction T_1 changed value of A to 500.

(c) Completion log (commit / abort log) record

- If transaction completes operations successfully then commit a transaction or if any problem while **executing** transaction decision to abort and hence rollback a transaction.

Operational Update Log : < T_n Commit > or < T_n Abort >

- It commits or rolls back the operations performed by transaction.

Example :

<T_1 Commit> : Transaction T_1 is committed to server.

Or <T_1 Rollback> : Transaction T_1 abort its operations.

(d) Checkpoint record

- Records a point when checkpoint has been made.
- These are used to speed up recovery.
- It also record information that eliminates the need to read a log's past.

- The time of recording varies according to checkpoint algorithm.

  Log record : <T_n Checkpoint A>

  It marks transaction status.

  Example :

  <T_1 Checkpoint A> : Transaction T_1 is committed to server.

(3) Recovery actions

Redo operation :

- If a transaction has recorded commit operation before failure occurs, then transaction must be redone.
- Recovery Action : **REDO** (T_i)

  Example : REDO (T_1)

Undo operation :

- If a transaction has not recorded commit operation before failure occurs, then transaction must be undone.
- Recovery Action : **UNDO** (T_i)

  Example : UNDO (T_1)

19.3.1 Deferred-Modification Technique (REDO Algorithm)

Q. Explain Deferred log based in detail.

(1) Introduction

- The deferred-modification technique will record database modifications in the transaction log files after transaction partially commits.
- **Partial commit** : When the final action of the transaction has been executed a transaction said to be partially committed.
- The version of the deferred-modification technique that we describe assumes that serial execution of transactions.
- Transaction perform partially commit,
- Transaction log is recorded.
- If the system crashes before the transaction commit or if the transaction aborts, then the information on the log will not be recorded.

(2) Working

The execution log based recovery of transaction T_n as follows :

(a) T_n starts its execution by writing **<T_n start>** to transaction log file.

(b) If Transaction performs (T_n) performs write(X) operation it will record a **new operational log**.

(c) If T_n **commits data to server,** than a record < T_n commit> is written to transaction log.

(d) Ensure that all the transactional log record are written to stable storage. As it is possible to that database may fail at any point of time.

(e) Once they have been written, the actual updating takes place to server then it will be in committed state.

(3) Example

- Consider a simple banking system. Let T_0 be a transaction that transfers Rs.50 from account A to account B :

T_0
read(A)
A :=A -50
write(A)
read(B)
B:= B + 50
write(B)

- Let T_1 be a transaction that withdraws Rs. 100 from account C.

T_1
read(C)
C:= C + 100
write(C)

- Suppose that these transactions are executed serially, in the order T_0 followed by T_1, and that the values of accounts A, B, and C before the execution took place were Rs. 100, Rs. 100, and Rs. 100, respectively.
- The portion of the log containing the relevant information on these two transactions appears as below,

<T_0 start>
<T_0, A, 50>
<T_0,B, 150>
<T_0 commit>
<T_1 start>
<T_1, C, 200>
<T_1 commit>

Fig. 19.3.2 : Database log corresponding to T_0 an T_1

- There are various orders in which the actual outputs can take place to both the database system and the log as a result of the execution of T_0 and T_1. As shown in Fig. 19.3.2.
- The value of A is changed in the database only after the record <T_0, A, 50> has been placed in the log.

Log	Database
<T_0 start >	
<T_0, A, 50 >	
<T_0, B 150 >	
<T_0 Commit>	
	A = 50
	B = 150
<T_1 Start>	
<T_1 C, 200>	
<T_1 Commit>	
	C = 200

Fig. 19.3.3

- Transaction log are used to handle any failure that results in the loss of data.
- The recovery scheme uses the following recovery procedure :

(I) Method 1 :

- Use two lists of transactions : the committed transaction since the last checkpoint and the active transaction.
- Apply the REDO operation to all the WRITE operations of the committed transactions from the log in the order in which they were to the log.
- Restart the active transaction. REDO(T_n) sets the value of all data items updated by transaction T_n to the new values.
- The data items updated by T_n and their new value is updated to the log file. The redo operation must be executing once.
- After a failure, the recovery system contacts log based recovery to check for recovery action.
- Transaction T_n will be redone if and only if the log contains **both** the record < T_n start> and the record < T_n commit>.
- Thus, if the system crashes after the transaction completes its execution, the recovery scheme uses the information in the log to restore the system to a previous consistent state after the transaction had completed.
- Let us return to our banking example with transactions T_0 and T_1 executed one after the other in the order T_0 followed by T_1. Fig. 19.3.4 shows the log those results from the complete execution of T_0 and T_1.

<T_0 start>
<T_0,A, 50>
<T_0, B, 150>

Fig. 19.3.4 : Same log at different times - no commit recorded

(II) Method 2 :

(A) System crashes before the completion of the transactions

(a) When the system starts after failure, no redo action is required. As, no commit record appears in the log.

(b) All data values of A & B remain same.

(c) The log records of the incomplete T_0 can be deleted from the log.

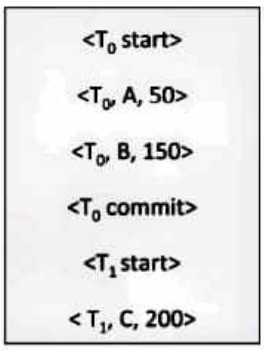


Fig. 19.3.5 : < T_0 commit > recorded

(B) The crash comes just after the log record

(a) For the step WRITE(C) of transaction T_1 has been written to stable storage.

(b) When the system comes back up, the operation redo (T_0) is performed since the record < commit> appears in the log on the disk.

(c) After this operation is executed T_0, the values A and B are Rs. 50 and Rs.150, respectively.

(d) The value of account C remain before, the log records of the incomplete transaction T_1 can be deleted from the log.

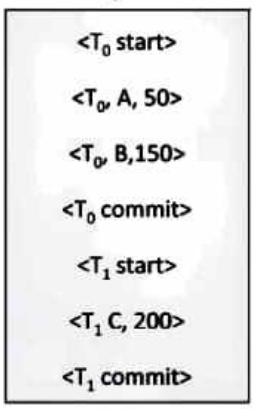


Fig. 19.3.6

(C) A crash occurs just after the log record < T_1 commit > is written to is stable storage

(a) The log at the time of this crash is as in Fig. 19.3.6 the system comes back up.

(b) Two commit records are in the log: one for T_1 and one for T_0.

(c) Therefore, the system must perform operations redo (T_0) and redo (T_1) in the order in which their commit records appear in the log.

(d) After the system executes these operations, the values of accounts A, B, and C, are Rs.50, Rs.150, and Rs. 200,

Finally, let us consider a case in which a second system crash occurs after recovery from the first crash.

(4) Summary

- For each commit record <T_n commit>found in the log, the system performs the operation redo (T_n). In other words, it restarts the recovery actions from the beginning.
- Since redo writes values to the database independent of the values currently in the database, the result of a successful second attempt at 'redo' is the same as though 'redo' had succeeded the first time.

19.3.2 Immediate Modification Technique (UNDO Algorithm)

Q. Explain Immediate log base in detail.

(1) Introduction

- The **immediate-modification technique** writes database modifications to be written to the database as soon as it is performed or in the active state.
- In the event of transaction failure, the system makes use of old-value of the log records to restore the data items.

(2) UNDO operation

- Before a transaction T_n starts its execution, the system writes the record <T_n start> to the log.
- During transaction execution, any **WRITE(X) operation by T_n** is preceded by the writing of the appropriate new update record to the log.
- When **T_n partially commits**, the system writes the record <T_n commit> to the log.
- Since the information in the log is used in restoring the original database state, we cannot allow the actual update to the database to take place before the corresponding log record is written out to stable storage.

- We therefore require that, before the execution of an output (B) operation its log records is written to stable storage.
- This procedure is defined as follows :
  1. Use two lists of transactions maintained by the system;
     (a) Committed transactions since the last checkpoint
     (b) Active transactions
  2. Undo all the WRITE operations of the active transaction from the log, using the UNDO procedure
  3. Redo the WRITE operations of transaction which are committed.
- **Undo WRITE operation :** It consists of check its log entry of write T and reading Old value and setting the value of item X in the database to old value.

(3) Example

- Consider a simple banking system, with transactions T_0 and T_1 executed one after the other in the order T_0 followed by T_1.
- Fig. 19.3.6 shows possible order of execution in both the database and the log as a result of the execution of T_0 and T_1

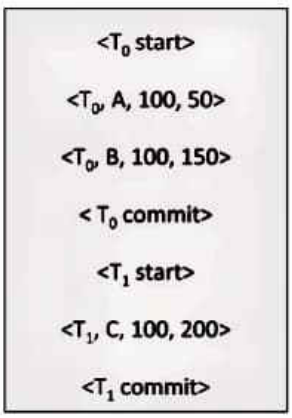


Fig. 19.3.6

Log	Database
$<T_0$ start>	
$<T_0$, A, 1200, 950>	
$<T_0$, B, 2000,2050>	
	A = 950
	B = 2050
$<T_0$ commit>	
$<T_1$ start>	
$<T_1$, C, 700, 800>	
$< T_1$ commit >	
	C = 800

Fig. 19.3.7

- Using the log file, the system can handle any failure that does not result in the loss of information in non volatile storage.
- The recovery scheme uses two recovery procedures:

(I) Method 1 :

(a) Undo(T_n) restores the value of all data items updated by transaction T_n to the old values and redo(T_n) sets the value of all data items updated by transaction T_n to the new values.

(b) The set of data items updated by T_n and their respective old and new values can be found in the log.

(c) The undo and redo operations must guarantee to correct behaviour, even if a failure occurs during the recovery process.

(d) After a failure has occurred, the recovery scheme consults the log to determine which transactions need to be redone, which need to be undone:

- Transaction T_n needs to be undone if the log contains the record $< T_n$ start>, but does not contain the record $<T_n$ commit>.
- Transaction T_n needs to be redone if the log contains both the record $<T_n$ start> and the record $<T_n$ commit>.
- In our banking example, with transaction T_0 and T_1 executed one after the other in the order T_0 followed by T_1.

<T_0 start>

<T_0, A, 100, 50>

<T_0, B, 100, 150>

(II) Method 2 :

(a) If system crashes before the completion of the transactions

- First, let us assume that the crash occurs just after the log record for the step of transaction T_0 has been written to stable storage.
- When the system comes back up, it finds the record <T_0 start> in the log, but no corresponding <T_0 commit> record.
- Thus, transaction T_0 must be undone, so an undo (T_0) is performed.
- As a result, the values in accounts A and B (on the disk) are restored to Rs.100 and Rs.100, respectively.

<T_0 start>

<T_0, A, 100, 50>

<T_0, B, 100, 150>

<T_0 commit>

<T_1 start>

<T_1, C, 300, 200>

Fig. 19.3.8

(b) If crash comes occurs after the log record for the step write (C) of transaction T_1 has been written to stable storage

Recovery Action

(i) UNDO(T_1) :

- Record <T_1 start> appears in the log, but there is no commit record for transaction
- <T_1 commit>. So, Transaction needs to be Undone its effects.

(ii) REDO(T_0) :

- Log contains both the record <T_0 start> as well as <T_0 commit>. Than transaction scold be written again to show its effect on transaction.
- In this example, the same outcome would result if the order were reversed.

<T_0 start>

<T_0, A, 100, 50>

<T_0, B, 100, 150>

<T_0 commit>

<T_1 start>

<T_1, C, 200, 300>

< T_1, commit>

Fig. 19.3.9

(c) If crash occurs just after the log record <T_1 commit> has been written to stable storage.

- When the system comes to backup, both T_0 and T_1 need to be redone, since the records <T_0 start> and <T_0 commit> appear in the log, as do the records <T_1 start> and <T_1 commit>.
- After the system performs the recovery procedures REDO(T_0) and REDO(T_1), the values in accounts A, B, and C, are Rs.50, Rs.150, and Rs.300, respectively.

19.4 Checkpoint - Recovery Related Structures

Q. Explain concept of checkpoint with suitable example.

(1) Introduction

- A database checkpoint is where all committed transactions are written to the redo/audit logs.
- The database administrator determines the frequency of the checkpoints based on volume of transactions.
- When a system fails, It check log to determine recovery action. Too frequent checkpoints can affect the performance.

(2) Problems in this approach

- The search process is time-consuming.
- Most of the transactions that, according to our algorithm, need to be redone as they have already written their updates into the database.

(3) Need of check points

- To record status of transaction execution, the system maintains the log, using one of the two techniques.
- The system periodically performs checkpoints, with following sequence of actions:
  (a) Output all log records onto stable storage which are currently stored in main memory.
  (b) Output to the disk.
  (c) Output onto stable storage a log record <checkpoint>.
- Transactions are not allowed to perform any update actions, such as writing to a buffer block or writing a log record, while a checkpoint is in working state.
- The presence of a <checkpoint> record in the log allows the system to restructure its recovery procedure.

(4) Working

- Consider a transaction T_n that committed prior to the checkpoint.
- For such a transaction, the <T_n commit> record appears in the log before the <checkpoint> record.
- Any database modifications made by transaction T_n must have been written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on T_n.
- After a database failure has occurred, the recovery scheme examines the log to determine the most recent transaction T_n that started executing before the most recent checkpoint took place.
- It can find such a transaction by searching the log backward, from the end of the log, until it finds the first <checkpoint> record (since we are searching backward, the record found is the final <checkpoint> record in the log); then it continues the search backward until it finds the next <T_n start> record. This record identifies a transaction T_n.
- Once the system has identified respective transaction T_n, the redo and undo operations need to be applied to only for transaction T_n and all transactions that started executing after transaction T_n.
- The exact recovery operations to be performed depend on the modification technique being used.

  For the immediate-modification technique, the recovery operations are:

  (a) For all transactions T_k in T that have no <T_k commit> record in the log, execute UNDO(T_k).
  (b) For all transactions T_k in T such that the record <T_k commit> appears in the log, execute REDO (T_k).
- Obviously, the undo operation does not need to be applied when the deferred-modification technique is being employed.
- Consider the set of transactions {T_0, T_1 .... , T_{10}} executed in the order of the subscripts. Suppose that the recent checkpoint took place during the execution of transaction T_6. Thus, only transactions T_6, T_7, ... , T_{10} need to be considered during the recovery scheme. Each of them needs to be redone if it has committed; otherwise, it needs to be undone.

(5) Advantages

(a) A database checkpoint keeps track of change information and enables incremental database backup.
(b) A database storage checkpoint can be mounted, allowing regular file system operations to be performed.
(c) Database checkpoints can be used for application solutions which include backup, recovery or database modifications.

(6) Disadvantages

(a) Database storage checkpoints can only be used to restore from logical errors (E.g. a human error).
(b) Because all the data blocks are on the same physical device, database storage checkpoints cannot be used to restore files due to a media failure.

19.5 Shadow Paging

Q. Explain concept of shadow paging.

(1) Introduction

- It is not always convenient to maintain logs of all transactions for the purposes of recovery.
- An alternative is to use a system of shadow paging.
- This is where the database is divided into pages that may be stored in any order on the disk.
- In order to identify the location of any given page, we use something called a **page table**.

(2) Method

(a) During the life of a transaction two-page tables are maintained as below,

(i) Shadow page table

(ii) Current page table.

(b) When a transaction begins both of these page tables point to the same locations (are identical).

(c) During the lifetime of a transaction the shadow page table doesn't change at all.

(d) However, during the lifetime of a transaction update values etc. may be changed.

(e) For pages updated by the transaction, two versions are kept. The old version is referenced by the shadow directory and the new version by the current directory.

(f) So whenever we update a page in the database we always write the updated page to a new location.

(g) This means that when we update our current page table it reflect the changes that have been made by that transaction.

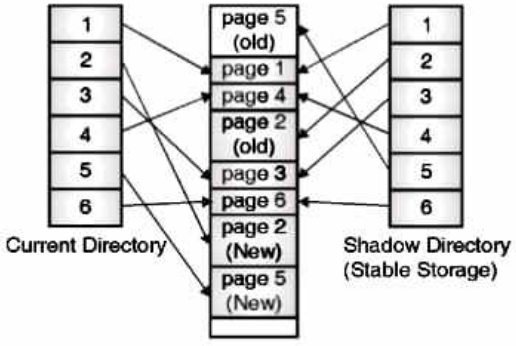


Fig. 19.5.1 : Page table-shadow paging

(h) As we can see the shadow page table shows the state of the database just prior to a transaction, and the current page table shows the state of the database during or after a transaction has been completed.

(3) Process of recovery

- We now have a system whereby if we ever want to undo the actions of a transaction all we have to do is recover the shadow page table to be the current page table.
- As such this method makes the shadow page table particularly important, and so it must always be stored on stable storage.
- On disk we store a single pointer location that points to the address of the shadow page table.
- This means that to swap the shadow table for the current page table (committing the data) we just need to update this single pointer (very unlikely to fail during this very short fast operation).
- In case of no failure means while committing transaction just discard the shadow directory.
- In case of multi-user environment with concurrent transaction, logs and checkpoints must be incorporated in shadow paging.

(4) Advantages

(a) Shadow page method does not require any Undo or Redo algorithm for recovery purpose.

(b) Recovery using this method will be faster.

(c) No overhead for writing log records.

(5) Disadvantages

(a) Data fragmentation

The main disadvantage of this technique is the updated Data will suffer from fragmentation as the data is divided up into pages that may or not be in linear order for large sets of related hence, complex storage management strategies.

(b) Commit overhead

If the directory size is large, the overhead of writing shadow directories to disk as transaction commit is significant.

(c) Garbage collection

– Garbage will accumulate in the pages on the disk as data is updated and pages lose any references. For example if i have a page that contains a data item X that is replaced with a new value then a new page will be created. Once the shadow page table is updated nothing will reference the old value of X.

The operation to migrate between current and shadow directories must be implemented as an atomic mode.

Review Questions

Q. 1 What is recovery? Explain types of recovery? (Forward, Backward Recovery)

Q. 2 Describe the use of logs and check points in a database.

Q. 3 Explain database backup and its types.

Q. 4 Explain concept of database recovery

Q. 5 Explain Immediate log base in detail.

Q. 6 Explain concept of shadow paging.

Q. 7 Explain concept of shadow paging.

❑❑❑

Appendix - A

Solved MSBTE Question Papers of Summer 2019 and Winter 2019

Summer 2019

Q. 1(a) Define: (i) Instance (ii) Schema **(Section 2.3)** **(2 Marks)**

Q. 1(b) List any four advantages of DBMS. **(Section 1.2)** **(2 Marks)**

Q. 1(c) State any two E.F. Codd's rule for RDBMS. **(Section 5.2.1)** **(2 Marks)**

Q. 1(d) List DCL commands. **(Section 7.2)** **(2 Marks)**

Q. 1(e) Define Normalization and list its types. **(Sections 6.1 and 6.5)** **(2 Marks)**

Q. 1(f) Write syntax for creating synonyms with example. **(Section 11.4.1)** **(2 Marks)**

Q. 1(g) State any four PL/SQL data types. **(Section 12.7)** **(2 Marks)**

Q. 2(a) Explain overall structure of DBMS with the help of diagram. **(Section 2.5)** **(4 Marks)**

Q. 2(b) Explain difference between delete and truncate command with example. **(Sections 7.3.4 and 7.3.6)** **(4 Marks)**

Q. 2(c) Write and explain syntax for creating view with example. **(Section 10.2)** **(4 Marks)**

Q. 2(d) Explain PL/SQL block structure with the help of diagram. **(Section 12.1.1)** **(4 Marks)**

Q. 3(a) State and explain 2NF with example. **(Section 6.5.2)** **(4 Marks)**

Q. 3(b) Explain any four aggregate functions with example. **(Section 8.5)** **(4 Marks)**

Q. 3(c) Explain exception handling in PL/SQL with example. **(Section 13.7)** **(4 Marks)**

Q. 3(d) Explain state of transaction with the help of diagram. **(Section 18.3)** **(4 Marks)**

Q. 4(a) State the difference between Relational and Hierarchical model. **(Section 3.3)** **(4 Marks)**

Q. 4(b) List the SQL operations and explain range searching operation 'between' and pattern matching operator 'like' with example. **(Section 7.8)** **(4 Marks)**

Q. 4(c) Explain cursor with example. **(Section 14.1)** **(4 Marks)**

Q. 4(d) State the use of database trigger and also list types of trigger. **(Sections 16.2 and 16.1.3)** **(4 Marks)**

Q. 4(e) Explain recovery techniques with example. **(Section 19.2)** **(4 Marks)**

Q. 5(a) Draw ER diagram for library management system considering issue and return, fine collection facility. Consider appropriate entities. **(Ex. 4.8.3)** **(6 Marks)**

Q. 5(b) Consider the table

Student (name, marks, dept, age, place, phone, birthdate) Write SQL query for following :

(i) To list students having place as 'Pune' or' Jalgaon'.

(ii) To list students having same department (dept) as that of' Rachana'.

(iii) To change marks of 'Rahul' from 81 to 96.To list student name and marks from 'Computer'dept.

(iv) To list student name who have marks less than 40.

(v) To list students who are not from' Mumbai'.

Ans. :

(i) To list students having place as 'Pune' or'Jalgaon'.

```
SELECT    name
FROM      Students
WHERE     place in('Pune','Jalgaon');
```

(ii) To list students having same department (dept) as that of 'Rachana'.

```
SELECT    name
FROM      Student
WHERE     dept= (SELECT  dept
                 FROM      student
                 WHERE     name='Rachana');
```

(iii) To change marks of 'Rahul' from 81 to 96.

```
UPDATE Student
SET       marks=96
WHERE     name= 'Rahul';
```

(iv) To list student name and marks from 'Computer 'dept.

```
SELECT  name,marks
FROM      Student
WHERE     dept='Computer';
```

(v) To list student name who have marks less than 40.

```
SELECT    name
FROM      Student
WHERE     marks<40;
```

(vi) To list students who are not from 'Mumbai'.

```
SELECT    *
FROM      Student
WHERE     place != 'Mumbai';
```

Q. 5(c) Create simple and composite index. Write command to drop above index.
(Sections 11.3.1,11.3.2 and 11.3.4) **(6 Marks)**

Q. 6(a)(i) Write a command to create table student (RNo., name, marks, dept.) with proper data type and RNo as primary key.
(ii) Write a command to create and drop sequence. **(6 Marks)**

Ans. :

1. Create table

```
CREATE TABLE STUDENT
(
RNO  DECIMAL(10,2) PRIMARY KEY,
NAME VARCHAR(100),
MARKS DECIMAL(10,2),
DEPT INT
)
```

2. Create sequences

```
CREATE SEQUENCES:
 Create sequence employee_seq
 start with 1
 increment by 1
 maxvalue 999
 no cycle
 no cache;
```

Drop sequence

```
DROP sequence employee_seq;
```

Q. 6(b) Write a PL/SQL program to calculate factorial of a given number. **(6 Marks)**

Ans. :

```
DECLARE
        n number;
       fac number=1;
       i number;
BEGIN
       n:=&n;
       FOR  i in 1..n
       LOOP
          fac:=fac*i;
       END LOOP;
       dbms_output.put_line('factorial='||fac);
END;
/
```

Q. 6(c) Write SQL command for following : (i) Create user (ii) Grant privileges to user (iii) Remove privileges from user. **(Sections 17.4.1, 17.5.1 and 17.5.2)** **(6 Marks)**

Winter 2019

Q. 1(a)	State any two advantages of DBMS over file processing system. **(Section 1.3)**	**(2 Marks)**
Q. 1(b)	Draw three level architecture of DBMS. **(Section 2.1)**	**(2 Marks)**
Q. 1(c)	Define table and field. **(Section 3.1.1)**	**(2 Marks)**
Q. 1(d)	Enlist DML commands. **(Section 7.5)**	**(2 Marks)**
Q. 1(e)	Define primary key and foreign key. **(Sections 7.4.2 and 7.4.3)**	**(2 Marks)**
Q. 1(f)	List any four string functions in SQL **(Section 8.2).**	**(2 Marks)**
Q. 1(g)	State any two advantages of functions in PL/SQL. **(Section 15.2.1)**	**(2 Marks)**
Q. 2(a)	Distinguish between Network and Hierarchical model. (Any four points) **(Section 3.3)**	**(4 Marks)**
Q. 2(b)	Explain any four set operators in SQL with example.**(Section 7.9)**	**(4 Marks)**
Q. 2(c)	Describe Views and write a command to create view.**(Sections 10.1 and 10.2)**	**(4 Marks)**
Q. 2(d)	Explain implicit and explicit cursors.**(Sections 14.1.1 and 14.1.2)**	**(4 Marks)**
Q. 3(a)	State and explain 3NF with example.**(Section 6.5.3)**	**(4 Marks)**
Q. 3(b)	Define index. Explain it's types. **(Section 11.3)**	**(4 Marks)**
Q. 3(c)	Explain Exception handling with it's types. **(Section 13.7)**	**(4 Marks)**
Q. 3(d)	Explain ACID properties of traction. **(Section 18.2)**	**(4 Marks)**
Q. 4(a)	Explain strong and weak entity set. **(Section 4.2)**	**(4 Marks)**
Q. 4(b)	Describe create & alter command with syntax and example. **(Sections 7.3.2 and 7.3.3)**	**(4 Marks)**
Q. 4(c)	Define database trigger. How to create and delete trigger? **(Sections 16.1, 16.1.2 and 16.1.4)**	**(4 Marks)**
Q. 4(d)	Explain any one control structure in PL/SQL with example. **(Sections 13.3 and 13.3.2)**	**(4 Marks)**
Q. 4(e)	Describe database backups with it's types. **(Section 19.1.2)**	**(4 Marks)**
Q. 5(a)	Draw an ER diagram for library management system. (Use Books, Publisher and Member entities).	**(4 Marks)**

Ans. :

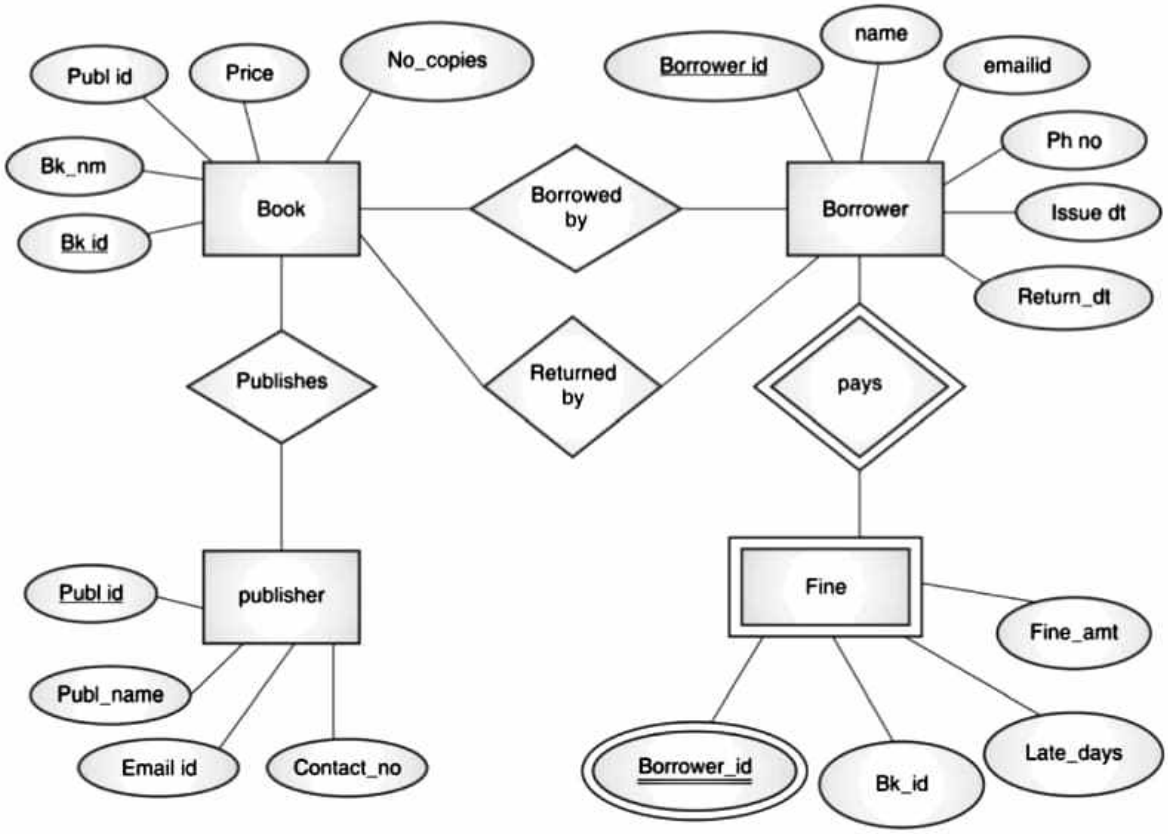


Fig. 1- Q. 5(a)

Q. 5(b) Write a command to create table student (rollno, Stud_name, branch, class, DOB, City, Contact_no) and write down queries for following:

(i) Insert one row into the table

(ii) Save the data

(iii) Insert second row into the table

(iv) Undo the insertion of second row

(v) Create save point S_1.

(vi) Insert one row into the table. **(4 Marks)**

Ans. :

(i) Insert one row into the table

```
INSERT INTO STUDENT VALUES (111,'YUVAN','IT','07-07-17','MUMBAI',9988779988);
```

(ii) Save the data

```
COMMIT;
```

(iii) Insert second row into the table

```
INSERT INTO STUDENT VALUES (112,'PRISHA','CS','31-06-17','MUMBAI',9988009900);
```

(iv) Undo the insertion of second row

```
ROLLBACK;
```

(v) Create save point S_1.

```
SAVEPOINT A;
```

(vi) Insert one row into the table.

```
INSERT INTO STUDENT VALUES (113,'JIJA','ELEX','14-07-17','MUMBAI',9955009900);
```

Q. 5(c) Consider following schema :

EMP (empno, deptno, ename, salary, designation, join_date, DOB, dept_location). Write down SQL queries for following:

(i) Display employees name & number in decreasing order of salary.

(ii) Display employee name & employee number whose designation is Manager.

(iii) Display age of employees with ename.

(iv) Display total salary of all employees.

(v) Display employee names having deptno as 20 and dept_location is Mumbai.

(vi) Display name of employee who earned lowest salary. **(4 Marks)**

Ans. :

(i) Display employees name & number in decreasing order of salary.

```
SELECT    ename, empno
FROM      Emp
ORDER By  salary desc;
```

(ii) Display employee name & employee number whose designation is Manager.

```
SELECT    ename, empno
FROM      Emp
WHERE     designation = 'Manager'
```

(iii) Display age of employees with ename.

```
SELECT    date diff (yy, DoB, getdate())
FROM      emp
WHERE     ename is not null;
```

(iv) Display total salary of all employees.

```
SELECT  sum (salary)
FROM      Emp;
```

(v) Display employee names having deptno as 20 and dept_location is Mumbai.

```
SELECT    ename
FROM      Emp
WHERE     dept no. = 20
AND       dept_location = 'Mumbai';
```

(vi) Display name of employee who earned lowest salary.

```
SELECT    ename
FROM      emp
WHERE     salary = (select min (salary) from emp);
```

Q. 6(a) Consider the structure for book table as Book-Master (bookid, bookname, author, no_of copies,price)
Write down SQL queries for following

(i) Write a command to create Book_master table.

(ii) Get authorwise list of all books.

(iii) Display all books whose price is between ` 500 & ` 800.

(iv) Display all books with details whose name start with 'D'.

(v) Display all books whose price is above ` 700.

(vi) Display all books whose number of copies are less than 10. **(4 Marks)**

Ans. :

(i) Write a command to create Book_master table.

```
Create bookmaster
(bookid varchar(10) primary key, bookname varchar (100), author varchar (100),  no-of-copy int, price number (10, 2));
```

(ii) Get authorwise list of all books.

```
SELECT *
FROM      bookmaster
ORDER BY author;
```

(iii) Display all books whose price is between ` 500 & ` 800.

```
SELECT *
FROM      bookmaster
WHERE     price between 500 and 800;
```

(iv) Display all books with details whose name start with 'D'.

```
SELECT *
FROM      bookmaster
WHERE     bookname LIKE 'D %';
```

(v) Display all books whose price is above ` 700.

```
SELECT *
FROM      bookmaster
WHERE     price >700;
```

(vi) Display all books whose number of copies are less than 10.

```
SELECT    *
FROM      bookmaster
WHERE     no_ of_ copies < 10 ;
```

Q. 6(b) Write a PL/SQL program to print n even numbers using For Loop. **(4 Marks)**

Ans. :

```
declare
 x number : = 0;
begin
for x in 1..100 loop
if mod(x,2)=0 then
dbms_output.put_line ( x);
```

```
end if;
end loop;
end;
declare
 x number := 0;
begin
for x in 1..N loop
if mod(x,2)=0 then
dbms_output.put_line ( x);
end if;
 end loop;
end;
declare
 x number := 0;
 N number := &N;
begin
for x in 1..N loop
        if mod(x,2)=0 then
            dbms_output.put_line ( x);
        end if;
end loop;
end;
declare
 x number := 0;
 N number := &N;
begin
for x in 1..N loop
        if mod(x,2)=0 then
            dbms_output.put_line ( x);
        end if;
end loop;
end;
/
```

Q. 6(c) Describe database privileges. Write down the procedure for granting & revoking privileges in database objects to the users. **(Section 7.14)** **(4 Marks)**

❑❑❑

Note

Note