Module I

Chapter 1 : Introduction Database Concepts 1-1 to 1-28

✓ **Syllabus Topic :** Introduction 1-1
1.1 Introduction to DBMS 1-1
✓ **Syllabus Topic :** Characteristics of Databases 1-3
1.2 Characteristics of DBMS 1-3
✓ **Syllabus Topic :** File System V/s Database System 1-5
1.3 File System v/s Database System 1-5
✓ **Syllabus Topic :** Users of a Database system 1-6
1.4 Database Users 1-6
✓ **Syllabus Topic :** Data Models 1-7
1.5 Data Model 1-7
1.6 Benefits of Data Modeling 1-8
1.7 Basic Building Block of Data Model 1-9
1.8 Object Based Logical Models 1-10
1.9 Record Based Logical Models 1-11
1.10 Hierarchical Model / Tree Model / XML Based Data Model 1-12
1.11 Network Model 1-13
1.12 Comparison of All Data Models 1-14
✓ **Syllabus Topic :** Schemas and Instances 1-15
1.13 Schema and Instances 1-15
✓ **Syllabus Topic :** Three-Schema Architecture 1-16
1.14 Three-Levels Schema Architecture 1-16
✓ **Syllabus Topic :** Data Independence 1-19
1.15 Data Independence 1-19
✓ **Syllabus Topic :** Database Administrator (DBA) 1-20
1.16 Database Administrator (DBA) 1-20

✓ **Syllabus Topic :** Role of a DBA 1-21
1.17 Role of Database Administrator (DBA) 1-21
1.17.1 Roles of DBA 1-21
1.17.2 Responsibilities of DBA 1-21
1.17.3 Skills required for DBA 1-22
1.18 Detailed DBMS Architecture 1-23
1.18.1 DBMS Architecture 1-23
1.18.2 Query Processor Components 1-23
1.18.3 Storage Manager / Storage Management 1-24
1.18.4 Transaction Management 1-25
1.19 Working of DBMS 1-25
1.20 University Questions and Answers 1-27

Module II

Chapter 2 : Entity Relationship Data Model 2-1 to 2-32

✓ **Syllabus Topic :** Conceptual Modeling of a Database 2-1
2.1 Conceptual Modeling of a Database 2-1
✓ **Syllabus Topic :** Entity-Relationship (ER) Model 2-3
2.2 Entity-Relationship (ER) Model 2-3
✓ **Syllabus Topic :** Entity Types and Entity Set 2-3
2.3 Entity 2-3
✓ **Syllabus Topic :** Weak Entity Types 2-5
✓ **Syllabus Topic :** Attributes and Keys 2-5
2.4 Attributes 2-5
✓ **Syllabus Topic :** Relationship Sets 2-9
2.5 Relationships 2-9
✓ **Syllabus Topic :** Relationship Types 2-10
2.6 Relationship Types based on Constraints 2-10
✓ **Syllabus Topic :** Extended Entity-Relationship (EER) Model 2-13
2.7 Extended Entity-Relationship (ER) Model 2-13

Scanned by CamScanner

✓ **Syllabus Topic :** Specialization 2-13
2.7.1 Specialization 2-13
✓ **Syllabus Topic :** Generalization 2-14
2.7.2 Constraints and Characteristics of Specialization and Generalization 2-16
✓ **Syllabus Topic :** Aggregation 2-20
2.8 Aggregation 2-20
2.9 Solved ER Designing Examples 2-20
2.10 University Questions and Answers 2-30

Module III

Chapter 3 : Relational Data Model **3-1 to 3-24**
✓ **Syllabus Topic :** Introduction to Relational Model 3-1
3.1 Relational Model 3-1
✓ **Syllabus Topic :** Relational Database Schema 3-4
3.2 Relational Database Schema 3-4
✓ **Syllabus Topic :** Relational Model Constraints 3-6
3.3 Relational Model Constraints 3-6
3.4 Domain Relational Constraint 3-6
3.5 Entity Integrity Constraints 3-6
3.6 Referential Integrity / Foreign Key 3-8
✓ **Syllabus Topic :** Mapping the ER and EER Model to the Relational Model 3-9
3.7 Concept of Keys 3-13
✓ **Syllabus Topic :** Mapping the ER and EER Model to the Relational Model 3-13
3.8 Mapping Entities to Tables 3-14
3.9 Mapping Attributes to Columns of Table 3-14
3.10 Mapping Relationships 3-15
3.11 Mapping Inheritance constraints 3-17
3.12 Solved Examples 3-19
3.12 University Questions and Answers 3-20
..... 3-23

Scanned by CamScanner

CHAPTER 1

Module I

Introduction Database Concepts

Course Outcome

After learning this topic students will be able to, explain the features of database management systems.

Syllabus Covered

Introduction, Characteristics of databases, File system V/s Database system, Users of a Database system Data Models, Schemas and Instances, Three-Schema Architecture and Data Independence, Database Administrator (DBA), Role of a DBA.

Syllabus Topic : Introduction

1.1 Introduction to DBMS

Q. Explain the detailed concept of DBMS.

- All 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, the price of a book, a 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 the bulk amount of data. But it is easier for us to remember only some amount of data.

Example 1

- You may be in a position to tell accurately the age, height, income, educational qualification, residential address, etc. of your close friends.
- But it is very difficult for you to memorize all these information for a large number of individuals in your company.

1. Data

Q. Write a short note on : Data

Scanned by CamScanner

- The facts and figures that can be recorded in system and that have some special meaning assigned to it.
- The system can be a manual system (register) or it can be a computerized system.
- **Example :** Data of a customer like name, telephone number, address and 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

Q. Write a short note on : Database

- A database is a collection of data items stored in one place and having something common between them Like a college database contains teachers, students, books, canteen etc. college is common (Base) between all above data items.
- So Data with a common base (Background) is called as Database (Data + Base).
- **Example :** College database stores information about students, teachers, classes, subjects (data with college as base).
- The database acts as a logical collection of relevant data. It is designed to offer an organized mechanism for storing, managing and retrieving stored information.

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)

University Question

Q. Discuss Database system. **(May 2017, 3 Marks)**

- A **Database Management System (DBMS)** is a collection of software or programs which help user in creation and maintenance of a database (set of information). Hence it is also known as a computerized record-keeping system.

- DBMS is the software system that helps in the process of defining, constructing, manipulating the database.
- Database management system has become an integral part of the information systems of many organizations as it is used to handle a huge amount of data.
- Computer-based **Information Systems (IS)** is capable of serving to many complex tasks in a coordinated manner. Such systems handle large volumes 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 **Information Systems (IS)** have to handle huge amounts of data. This core module of an IS 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.

Syllabus Topic : Characteristics of Databases

1.2 Characteristics of DBMS

Q. Explain the features of DBMS.
Q. Explain various advantages of Databases.

- The database approach has many important characteristics due to which database has become an integral part of the software industry.
- **The various characteristics of the databases are as mentioned below :**

1. Data integrity
2. Data security
3. Data independence
4. Transaction control – Commit and 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 that do not result in a loss of data consistency and correctness.

Scanned by CamScanner

- 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 database is accurate.
- Some examples of incorrect data are as below :
  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 data.
- Authorized users are able to access data any time he wants.

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 to database can be reverted back with help of rollback command.
- The changes can be saved successful 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.

Syllabus Topic : File System V/s Database System

1.3 File System v/s Database System

MU - May 2012, Dec.2014,May 2015, Dec.2015

University Questions

Q. Discuss the advantages of Database system over Files system. **(May 2012, 5 Marks)**

Q. Explain advantages of DBMS over file system. **(Dec.2014, 10 Marks)**

Q. List four significant differences between file processing system and database management system. **(May 2015, Dec.2015, 5 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.

Scanned by CamScanner

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.

Syllabus Topic : Users of a Database system

1.4 Database Users

MU - May 2017

University Question

Q. Discuss different Users. (May 2017, 3 Marks)

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.

Syllabus Topic : Data Models

1.5 Data Model

MU - May 2014, May 2016

University Question

Q. Explain different data models with its advantages and disadvantages. (May 2014, May 2016, 10 Marks)

- The data model will give you the idea how your final system or software will look like after development is completed.
- This concept is exactly like real world modeling in which before constructing any project (Bridges, Buildings, Towers etc.) engineers create a model for it, this model gives you the 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 specified system.

Scanned by CamScanner

Definition

According to Hoberman (2009)

- "A data model is a way finding tool for both business and IT professionals, which uses a set of symbols and text to precisely explain a subset of real information to improve communication within the organization and thereby lead to a more flexible and stable application environment."
- Data model is a simple Abstraction of complex real world data gathering environment.

1.6 Benefits of Data Modeling

1. A data model is a set of concepts that can be used to **describe the structure of data** in a database.

   In Fig. 1.6.1 (Simple Logical Data model) we have described structure of student data and class data.

2. Data models are used to **support the development** of information systems by providing the definition and format of data to be involved in future system.

   Data model is acting like a guideline for development also gives idea about possible alternatives to achieve targeted solution.

3. A data model can be sometimes referred to as a **data structure**, especially in the context of programming languages.

   In Fig. 1.6.1 (Simple Logical Data model) student and class are data structure of type class.

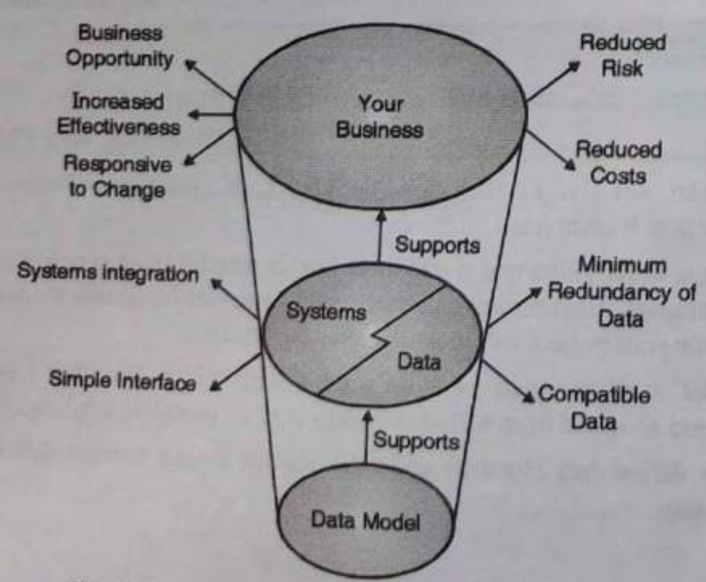


Fig. 1.6.1 : Important roles of Data Model in DBMS

a. Reduced risk

- Data model prevents system from **future risk and failure** by defining structure of data in advance.
- As we got idea of final system in the beginning of development itself so if need to have any revision or improvement we can do it in system, as actual system is not yet developed.

b. Reduced cost

As we got an idea of final system at the beginning of development itself, so we can reduce cost of project by proper planning and cost estimations as actual system is not yet developed.

c. Minimizes redundancy and data compatibility

Data repetition and data type compatibility can be checked and removed with help of data model.

d. Improves effectiveness of system

We can improve Graphical User Interface (GUI) of system by making its model and get it approved by its future user (user of that system) so it will be simple for them to operate system and make entire system effective.

1.7 Basic Building Block of Data Model

1. Introduction

The basic building block for any of model is Entities, Attributes, relationships and constraints. This Model uses collection of tables to represent relationships amongst the data.

2. Entity

A fundamental component of a model. An entity is having its own independent existence in real world.

Example, A Student, Faculty, Subject having independent existence.

An entity may be an object with a physical existence or it may have logical existence.

Example, Entities like Department, Section, adult (age>18) may have physical existence or it may have only logical existence.

3. Attributes

Each entity has its own properties which describes that entity such properties are known as **attributes.** In relational model, the column in relation (Table) or field of data is also called as Attribute.

Scanned by CamScanner

The single attribute will contains the similar type of data of all entities in relation.

Name
Mahesh
Suhas
Jay
Sachin

Example, The Name attribute in above student relation will contains the name of all student entities in student relation.

4. Relationships

A relationship is an association among several entities. E.g. Employee works for Department.

Degree : The degree of relationship type is number of participating entity types in a particular relation. Data model uses three types of relationships as below :

(a) One is to one

1. One entity is associated with at most one other entity.
2. **E.g.** One department can have only one manager.

(b) One is to Many

1. One entity is associated with any number of entities in other entity.
2. **E.g.** One teacher may teach to many students.

(c) Many is to Many

1. One entity is associated with any number of entities in other entity.
2. **E.g.** Books in library issued by students.

1.8 Object Based Logical Models

Q. Explain object based logical model.

- 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.

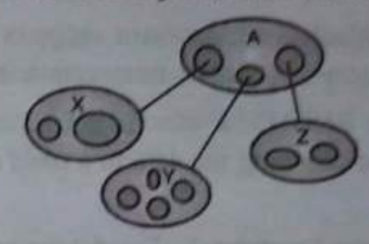


Fig. 1.8.1 : Object model

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.

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.

1.9 Record Based Logical Models

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.
- A relational database is a collection of 2-dimensional tables which consists of rows and columns.

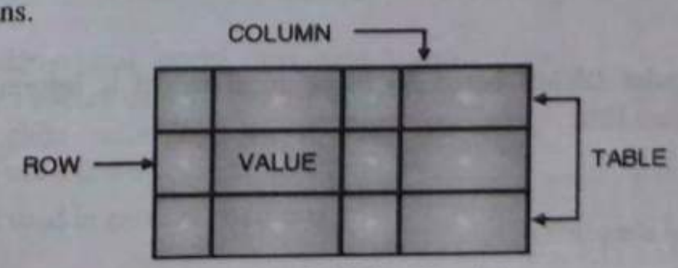


Fig. 1.9.1 : Relational model

(b) Example

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

Scanned by CamScanner

(c) 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.

1.10 Hierarchical Model / Tree Model / XML Based Data 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 in 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.

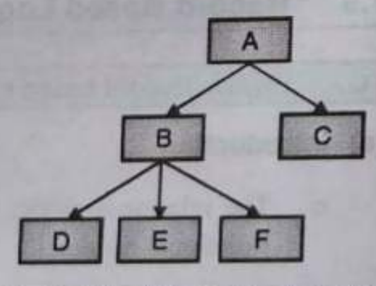


Fig. 1.10.1 : Hierarchical model

(b) Example

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

(c) 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.

(d) Disadvantages

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.

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.

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.

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

1.11 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 used in network databases.

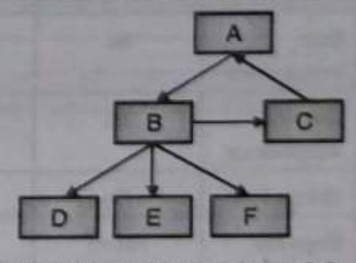


Fig. 1.11.1 : Network model

(b) Example

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

Scanned by CamScanner

(c) Advantages

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

(ii) Ability to handle many types of relationship

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

(iii) Ease of data access

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

(d) Disadvantages

(i) System complexity

(a) In a network model, data are accessed one record 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.

1.12 Comparison of All Data Models

Q. Compare various data models.

Sr. No.	Parameter	File system Model	Hierarchical Model	Network Model	Relational Model	ER Model	Object oriented Model
1.	Data Independence	No	Yes	Yes	Yes	Yes	Yes
2.	Structural Independence	No	No	No	Yes	Yes	Yes
3.	Storage Type	File	Segment	Record	Relation (Table)	Entity	Class
4.	Single row storage	Record	Segment occurrence	Current Record	Row(tuple)	Entity Occurrence	Object – instance of class
5.	Basic storage	Field	Segment field	Record field	Relation Attribute	Entity attribute	Object attribute
6.	Storage Identifier	Index	Sequence Field	Record key	Key	Entity key attribute	Object identifier

Sr. No.	Parameter	File system Model	Hierarchical Model	Network Model	Relational Model	ER Model	Object oriented Model
7.	Advantages	*Simple to implement, *Low cost implementation	*Promotes Data sharing, *conceptual simplicity, *Handle simple relationships(1:N) *Flexible data access	*conceptual simplicity, *Handle complex relationships (M:N)	*Tabular view, *Adhoc query capability, *improves management and implementation simplicity	*Very good *conceptual simplicity, *Effective *communication tool, *Integrated with dominant relational tools	*Semantic contents, *Promotes data integrity
8.	Disadvantages	*Limited implementations *Lack of standards	*Complex implementation, *Lack of standards *Limited implementations (No DML)	*Simplicity limits efficiency *Complex navigational system	*Hardware and software required	*Limited constraint representation *Limited relationship representation * No DML	* Slow development Of standards * complex navigation * Slow transactions if overload on system
9.	Examples	Operating system, Notepad, CSF(Comma Separated Files)	IMS (Information Management System)	IDS (Integrated Data Store)	Oracle, DB2, SQL SERVER etc.	ER diagrams	Database using C, java etc.

Syllabus Topic : Schemas and Instances

1.13 Schema and Instances

Q. Write a note on : Database Schema and Instances

1. Database Schema

- Database schema is a structure denotes the logical view of complete database.
- Schema consists of entities and relationship among these entities.
- It is similar to programming data types and variable to store data.
- The database schema is the information about database or table structure.

Student table

Sid	Name	Class	Subject

Fig. 1.13.1: Student Database Schema

Scanned by CamScanner

- Types of Database Schemas.
- Physical Database Schema
- This schema represents physical structure of data or actual storage of data.
- It is similar to actual variable in programming.
- It defines how the data will be stored in memory.

Example,

- The name is stored as a character data in storage.
- Logical Database Schema
- This schema shows the logical structure need to be applied on the data stored.
- Similar to as data type of variable.
- It can be defined as tables, views, and integrity constraints.

Example,

- The database consists of information about a set of students and departments in a college and the relationship between them

2. Database Instance

- The data content of the database at a particular point in time is also referred as a database instance.
- It is similar to the value of a variable,

Student table	A001	Dinesh	FYJC	DBMS

Fig. 1.13.2 : Student Database Instance

- It is always possible that this data will change with time. So it is very dynamic in nature.

Syllabus Topic : Three-Schema Architecture

1.14 Three-Levels Schema Architecture

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 (user applications interface) and the back end (physical database).
- 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 does not expected to change frequently.

2. Database architecture

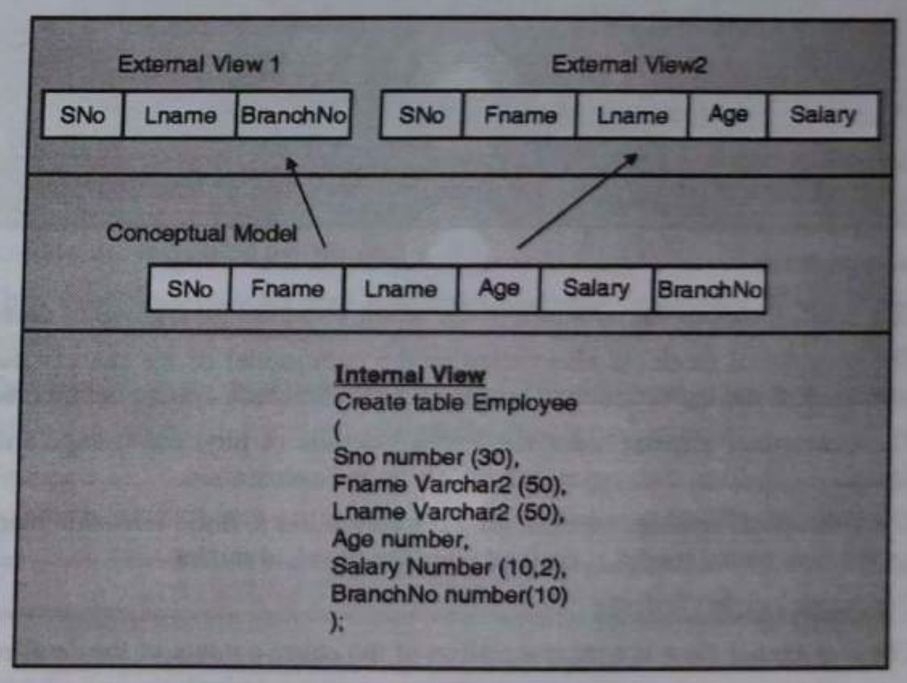


Fig. 1.14.1 : Database schema levels

(I) Internal level (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.

Scanned by CamScanner

- 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 specified what indexes 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 (view level)

- The external level is the one closest to the user, i.e., it is the 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.

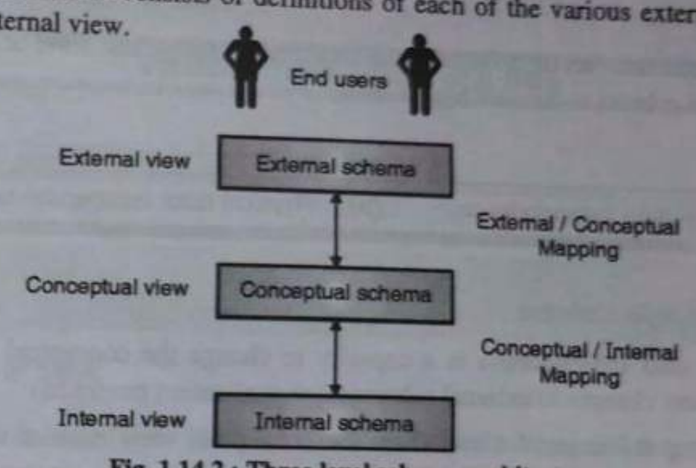


Fig. 1.14.2 : Three level schema architecture

(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 mapping :** The DBMS must transform a request on an external schema into a request against the conceptual schema.
- **Conceptual / internal mapping :** A certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

Syllabus Topic : Data Independence

1.15 Data Independence

MU - Dec. 2013, Dec. 2014, May 2016, Dec. 2016

University Questions

Q.	Define data Independence and explain types of data Independence.	(Dec. 2013, 5 Marks)
Q.	Define data Independence.	(Dec. 2014, 2 Marks)
Q.	Explain the term : Data Independence.	(May 2016, 2 Marks)
Q.	Write short notes on : Data Independence.	(Dec. 2016, 5 Marks)

Concept of data independence can be explained with help of 3 schema architecture. The

Scanned by CamScanner

three-schema architecture can make it easier to achieve true data independence.

1. 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. Types

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

(a) Logical data independence

- Logical data independence is a capacity to change the conceptual schema without having 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 having 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 same data as before remains in the database, we should not have to change the conceptual schema.

Syllabus Topic : Database Administrator (DBA)

1.16 Database Administrator (DBA)

Q. Write short notes on : Database Administrator.

- 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.

Syllabus Topic : Role of a DBA

1.17 Role of Database Administrator (DBA)

MU - May 2016

University Question

Q. Discuss the role of Database Administrator. **(May 2016, 5 Marks)**

1.17.1 Roles of DBA

- The DBA needs to performs many roles to keep the database up and running,
- System Administrator / Designer
- The database administrator need to manage DBMS software and server.
- He is also responsible for deciding on the storage and access methods.
- The DBA performs all data field updates or adding new fields into database.
- Database Developer / Programmer
- The DBA writes the programmes to design database and to design the means of reorganizing databases periodically.
- The DBA also determine and implement database searching strategies.
- System Analyst
- The DBA needs to analyses the system performance and fine tune the DBMS activities.
- DBA needs to take care of system crashes by planning proper recovery procedures.
- He will also specify techniques for monitoring database performance

1.17.2 Responsibilities of DBA

MU - May 2012

University Question

Q. Write short notes on : Responsibilities of Database Administrator. **(May 2012, 5 Marks)**

- The various responsibilities of DBA are as follows,
- Designing overall Database schema
- The DBA is responsible for designing overall database schema (tables and fields). Also responsible for deciding on the data storage and access methods.
- Selecting and installing database software and hardware

Scanned by CamScanner

- The DBA selects the suitable DBMS software like Oracle, SQL Server or MySQL. The Designing the means of reorganizing databases periodically.
- Designing Authorization/Access Control
- The DBA will decide the user access levels and security checks for access and data manipulations.
- Designing Recovery Procedures
- In order to take care of system crashes DBA needs to design the system recovery procedures and also specifying techniques for monitoring database performance.
- Operations Management
- 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.

1.17.3 Skills required for DBA

Q. List the important skills required to a Database Administrator (DBA).

- The various programming and soft skills are required to DBA are as follows,
  - Good communication skills
  - Excellent knowledge of databases architecture and design and RDBMS
  - 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.

1.18 Detailed DBMS Architecture

1.18.1 DBMS Architecture

MU - Dec. 2013, May 2015, May 2017

University Questions

Q. Draw and explain database system structure. **(Dec. 2013, May 2015, May 2017, 10 Marks)**

Q. Describe overall architecture of DBMS with diagram. **(May 2015, 10 Marks)**

- A database system can be separated into two different modules that deal with all operations of the overall system.
- Components of a database system,
  1. Query Processor Components
  2. Storage Manager/ Storage Management
  3. Transaction Management
- The storage manager is important because databases typically require a huge amount of storage space.

1.18.2 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 interprets DDL statements and fetch the definitions in the data dictionary.

(ii) DML compiler

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

Scanned by CamScanner

(iii) Query evaluation engine

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

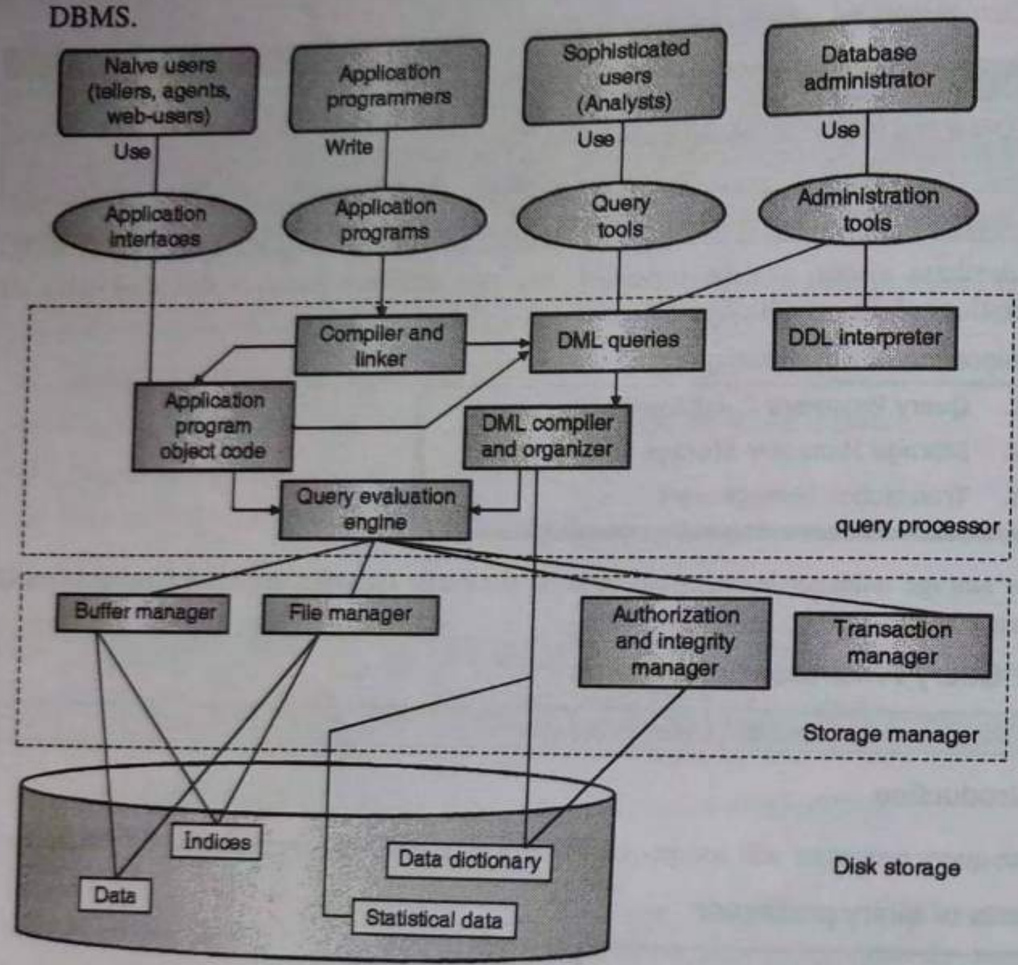


Fig. 1.18.1 : Components of DBMS

1.18.3 Storage Manager / Storage Management

Q. Write a short notes 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 :
  - **Authorization and integrity manager :** Checks for integrity constraints and authority of users to access data.
  - **Transaction manager**, which ensures that the database remains in a consistent (correct) state although there is system failures.
  - **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
  - **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,
  - **Data files :** Stored in the database itself.
  - **Data dictionary :** Stores metadata about the structure of the database.
  - **Indices :** Provide fast access to data items.

1.18.4 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.

1.19 Working of DBMS

(a) User requests data item from database.

(b) DBMS intercepts and interprets the request.

(c) Retrieves the data from the physical database.

(d) Constructs the record using physical/conceptual mapping.

(e) Records constructed using relevant conceptual/external mapping.

(f) Derives the required external record from conceptual record.

Scanned by CamScanner

Example

- Consider the situation in a library. Here, we have **data** corresponding to *books, authors, suppliers, borrowers*, etc. The total volume of data stored and handled in a library may be quite large.
- The Library DBMS may require several **operations,** such as *issue, return* or *purchase* of books; handle **queries** relating to *book information, borrowing information*, etc.
- Moreover, there are different types of **users** who operate various stages or activities. For instance, a *borrower* may merely view certain information, whereas an *issuer* may be allowed to update the status of a book during issue or return.
- The *Library staff* may, on the other hand, add new books, their supplier, price and other information to the database.
- Each user category has a different **access right** on both, the data, as well as the processing capabilities.
- Multiple users may concurrently operate the library DBMS performing several tasks at the same time.
- They may even try to access the same data simultaneously. It is the job of a DBMS to handle the data and it's processing in an integrated, coordinated and consistent manner.

Review Questions

Q. 1 Write advantages of DBMS over a file system.

Q. 2 State five main advantages of DBMS.

Q. 3 What are the different types of database system users?

Q. 4 List the functions of a Database Administrator (DBA).

Q. 5 Write short note on : Responsibilities of database administrator.

Q. 6 Describe the overall architecture of DBMS with diagram.

Q. 7 Explain the following term : Data independence and its types.

Q. 8 Explain the features of DBMS.

Q. 9 Explain three-level architecture of DBMS.

Q. 10 Explain need of data model in DBMS.

Q. 11 Explain the difference between external, internal and conceptual schemas. How these different layers are related to the concepts of logical and physical and physical data independence ?

Q. 12 Describe advantages of database management system over file-processing system.

Q. 13 What are the advantages of file processing system which were removed by DBMS ?

Q. 14 Why would you choose a database system instead of simply storing the data in operating system files ?

Q. 15 What is Data model ? Explain various types of data models.

Q. 16 Explain Hierarchical and network database model.

Q. 17 Compare various data models available.

Q. 18 Explain various data model with their advantages and disadvantages.

1.20 University Questions and Answers

May 2012

Q. 1 Discuss the advantages of Database system over Files system. *(Ans. : Refer section 1.3)* **(5 Marks)**

Q. 2 Write short notes on : Responsibilities of Database Administrator. *(Ans. : Refer section 1.17.2)* **(5 Marks)**

Dec. 2013

Q. 3 Define data Independence and explain types of data Independence. *(Ans. : Refer section 1.15)* **(5 Marks)**

Q. 4 Draw and explain database system structure. *(Ans. : Refer section 1.18.1)* **(10 Marks)**

May 2014

Q. 5 Explain different data models with its advantages and disadvantages. *(Ans. : Refer section 1.5)* **(10 Marks)**

Dec. 2014

Q. 6 Define data independence. *(Ans. : Refer section 1.15)* **(2 Marks)**

Q. 7 Explain advantages of DBMS over file system. *(Ans. : Refer section 1.3)* **(10 Marks)**

May 2015

Q. 8 Describe overall architecture of DBMS with diagram. *(Ans. : Refer section 1.18.1)* **(10 Marks)**

Scanned by CamScanner

Q. 9 List four significant differences between file processing system and database management system. *(Ans. : Refer section 1.3)* **(5 Marks)**

Q. 10 Draw and explain database system structure. *(Ans.: Refer Section 1.18.1)* **(10 Marks)**

Dec. 2015

Q. 11 List four significant differences between file processing system and database management system. *(Ans. : Refer section 1.3)* **(5 Marks)**

May 2016

Q. 3(a)(v) Explain the term : Data independence *(Ans.: Refer section 1.15)* **(2 Marks)**

Q.2(a) Explain different data models with its advantages and disadvantages. *(Ans. : Refer section 1.5)* **(10 Marks)**

May 2017

Q. 1(c) Discuss different Users and Database system. *(Ans.: Refer sections 1.1(3)and 1.4)* **(5 Marks)**

Q. 5(b) Draw and explain database system structure. *(Ans.: Refer Section 1.18.1)* **(10 Marks)**

❑❑❑

CHAPTER 2

Module II

Entity Relationship Data Model

Course Outcome

After learning this topic students will be able to, Design conceptual model of a database using ER modeling for any real life applications

Syllabus Covered

Conceptual Modeling of a database, The Entity-Relationship (ER) Model, Entity Types and Entity Sets, Attributes and Keys, Relationship Types, Relationship Sets, Weak Entity Types, Generalization, Specialization and Aggregation, Extended Entity-Relationship (EER) model

Syllabus Topic : Conceptual Modeling of a Database

2.1 Conceptual Modeling of a Database

The database design and implementation process need a systematic approach to design the logical and physical structure of one or more databases to satisfy all information needs of the users in an organization for any given applications.

1. Gathering Requirements from users

Before designing any database application one must know about goals of system, expectations of end users and uses of database application in detailed manner.

2. Conceptual Modeling / Logical Modeling

A conceptual schema is a high-level description of user requirements. It generally includes the main concepts and relationship between them. A conceptual data model is a representation of the concepts and their relationships.

Scanned by CamScanner

The second phase of database design involves two parallel activities,

(a) Conceptual Data Model - Check all data requirements produced by previous phase.

(b) Transaction Design - Design the characteristics of known database transactions.

3. **Physical Database Design / Physical Modeling**

Physical database design is the process of selecting data structure and access paths for database files to achieve good performance. Each DBMS offers many multiple options for file organization and access paths.

4. **Database Implementation and Tuning**

After the logical and physical designs are completed the final database schema is ready for final implementation.

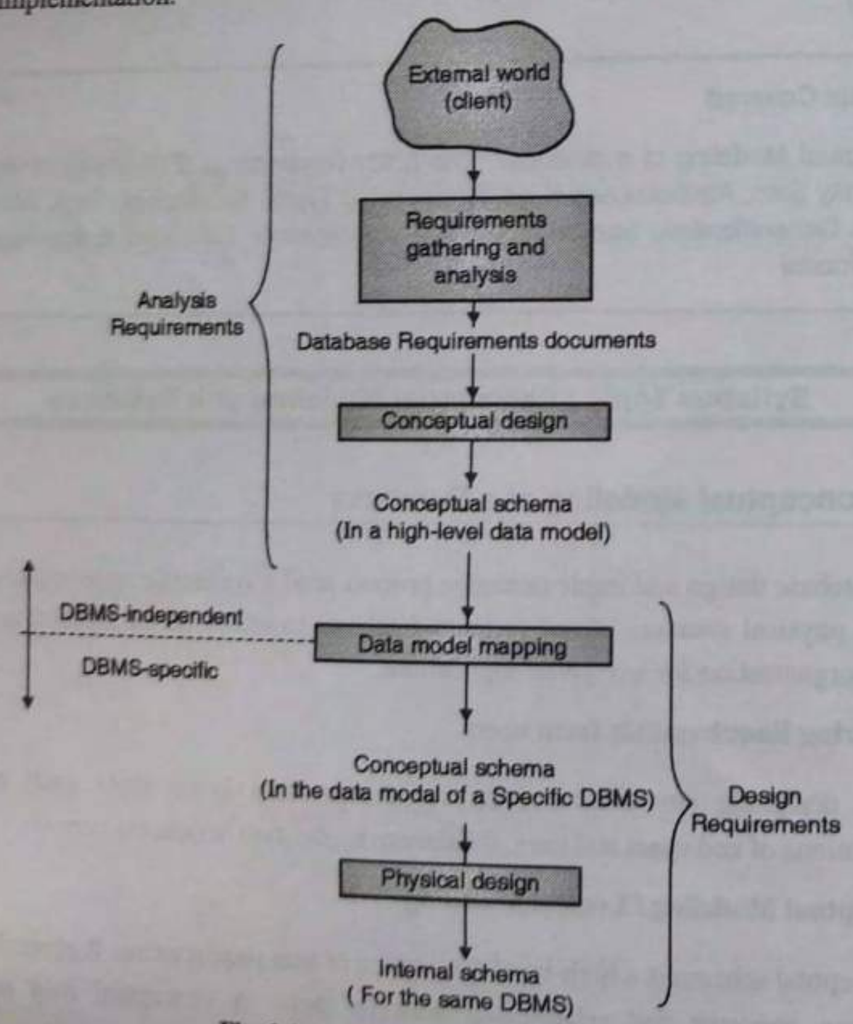


Fig. 2.1.1 : Database Design phases

Syllabus Topic : Entity-Relationship (ER) Model

2.2 Entity-Relationship (ER) Model

University Question MU - Dec. 2015

Q. Compare ER and EER models. **(Dec. 2015, 5 Marks)**

- 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.
- The ER data model is based on perception of real world data that consists of set of entities (data items) and relationship among these entities.
- ER Diagrams having components,
  - Entity
  - Attributes
  - Relationships

Syllabus Topic : Entity Types and Entity Set

2.3 Entity

University Question MU - May 2015

Q. We can convert any entity set to a strong entity set by simply adding appropriates attributes. Why then, do we have weak entity sets ? **(May 2015, 5 Marks)**

(1) Introduction

- Entity is anything in real world which may have physical or logical existence.
- An **entity** is anything in real world with its physical existence. Example, Student, faculty, subject having independent physical existence.
- An entity may be an object with a physical existence or it may have logical existence. Example, Department, Section, subject may have logical existence.
- Each entity has its own properties which describes that entity such properties are known as **attributes.**

Scanned by CamScanner

(2) Entity Set

- Entity set is collection of entities of same type.
- Example, Student entity set contains all students in college database.

(3) Entity Type

- Entity set is collection of entities with same attributes.
- As in Student table, each row is an entity and have same attributes. In other words we can say a student table is an entity type
- The types of entities are,

(a) Strong entity type
(b) Weak entity type

(a) Strong entity type

MU - May 2012, May 2014

University Question
Q. What is strong entity ? Explain with example. **(May 2012, 5 Marks)**
Q. Define Strong entity set. **(May 2014, 2 Marks)**

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

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 identified by unique roll number which is his primary key.
- Strong entity type is represented by single rectangle.

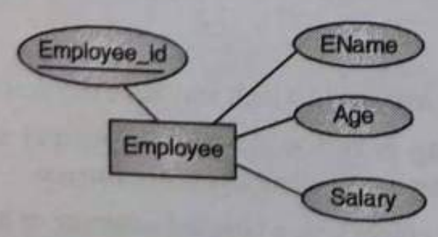


Fig. 2.3.1 : Employee entity

Syllabus Topic : Weak Entity Types

(b) Weak entity type

MU - May 2012

University Question
Q. What is weak entity ? Explain with example. **(May 2012, 5 Marks)**

- Entity type which cannot form distinct key from their attributes and takes help from corresponding strong entity is called as weak entity type.
- 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 depend on employee entity for primary key.

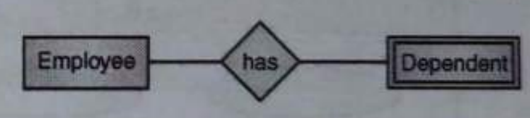


Fig. 2.3.2 : Weak entity "dependent"

Syllabus Topic : Attributes and Keys

2.4 Attributes

MU - Dec.2014,May 2017

University Question
Q. Explain different types of attributes in ER Model **(May 2017, 5 Marks)**

Introduction

- Each entity has its own properties which describes that entity such properties are known as attributes.
- The attribute value that describes each entity becomes a major part of data stored in database.
- Employee entity may be described by attributes name, age, phone etc.

Type	Notation
Attribute (Simple/Single valued/Stored)	

Scanned by CamScanner

- A particular entity will have some value for each of its attributes.

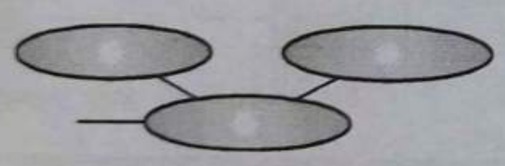
Example,

- For an employee of with Employee_id 30, the name attribute value is 'Jayendra'
- The various types of attribute are used in ER diagrams,
  - Composite Attributes
  - Multivalued Attributes
  - Derived Attributes
  - Null Attributes
  - Key Attributes

(a) Composite Attributes

Q. Write a note on composite attributes.

- The attributes which can be divided in multiple subparts.

Type	Notation
Composite attribute	

- The divisible attributes are composite attributes.

Example, The Name attribute of Student table can be divided into First_Name and Last_Name.

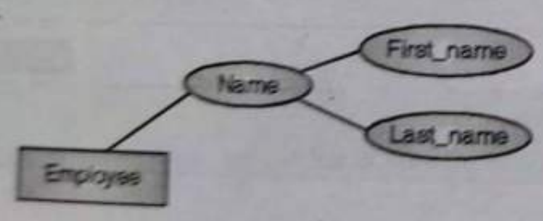


Fig. 2.4.1 Composite attributes

(b) Multivalued attributes

Q. Write a note on multivalued attributes.

- The attribute having more then one value for a same entity is called as multi-valued attribute.

Type	Notation
Multivalued Attribute	

- Example, A Single student can have multiple mobile numbers.

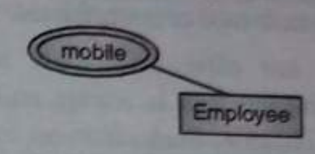


Fig. 2.4.2 Multi valued attributes

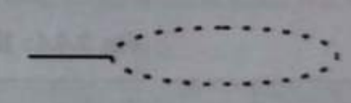
(c) Derived Attributes

MU - Dec. 2014

University Question

Q. Define Derived attribute. (Dec. 2014, 2 Marks)

- The value of some attribute can be derived from the value of related stored attribute such attributes are known as derived attributes.

Type	Notation
Derived attribute	

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

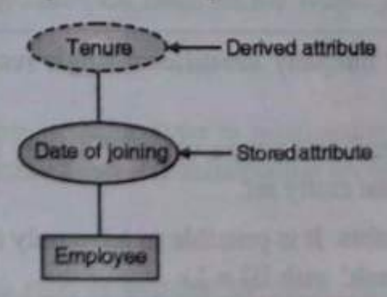


Fig. 2.4.3 : Derived attributes

(d) Null attribute

Q. Write a note on Null attributes

- 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.

Scanned by CamScanner

- Example, The 'Net_Banking_Active_Bin' attribute gives weather particular customer having net bankig facility 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 use or for unknown, unsigned, missing values of attribute.

(e) Key attributes

Q. Write a note on 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.

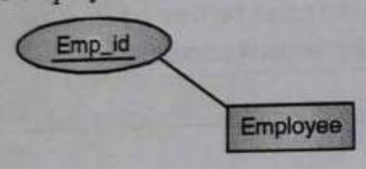


Fig. 2.4.4 : Key attributes

Type	Notation
Key attribute	

- The column value that uniquely identifies a single record in a table called as key of table.
- An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set.
- 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)

Key type	Definition
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row.
Foreign key	An attribute (or combination of attributes) in one table whose values must either match the primary key in another table or be null.

Syllabus Topic : Relationship Sets

2.5 Relationships

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

1. Introduction

- A relationship is an association among one or more than one entities.
- We use diamond shape to show relationship.

Type	Notation
Relationship	

- It is recommended to arrange relationship to be read it from left to right or up to down.
- **Example :** Employee works for Department.

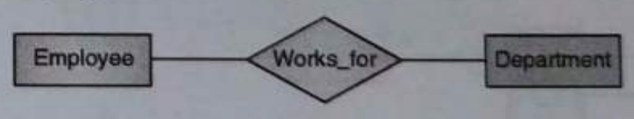


Fig. 2.5.1 : ER Diagram for Works_for

2. Relationship Set

Collection of all relationship of same type is relationship set. The many employees are working for different departments so it is relationship set of Works_For relationship.

3. Degree of Relationship

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

Types of Relationship based on degree are,

- Unary Relationship
- Binary Relationship
- Ternary Relationship

Scanned by CamScanner

Syllabus Topic : Relationship Types

2.6 Relationship Types based on Constraints

MU - May 2015, May 2016, Dec. 2016

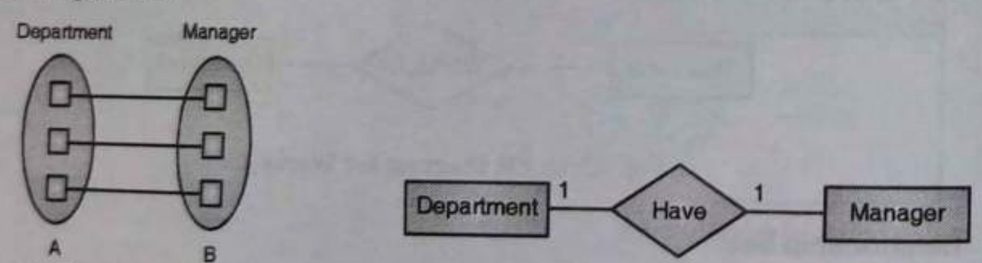
(A) Mapping Constraints / Cardinalities

- 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 is 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.

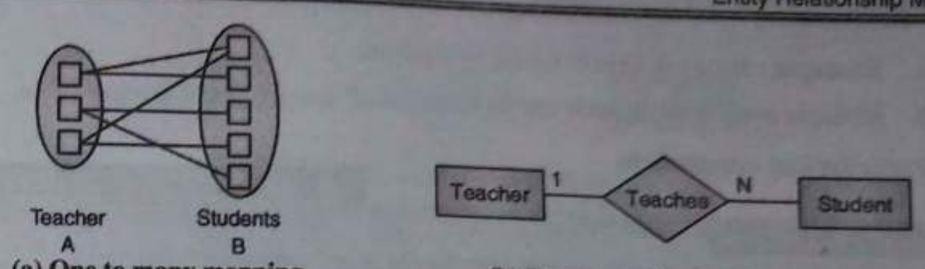


(a) One to one mapping (b) Representation in ER diagram

Fig. 2.6.1 : 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 associated with any number of entities in B.
- B 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.

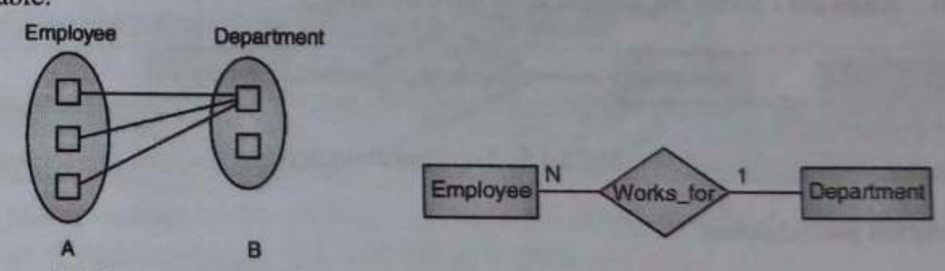


(a) One to many mapping (b) Representation in ER diagram

Fig. 2.6.2 : 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 associated with at most one entity in B.
- B 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.

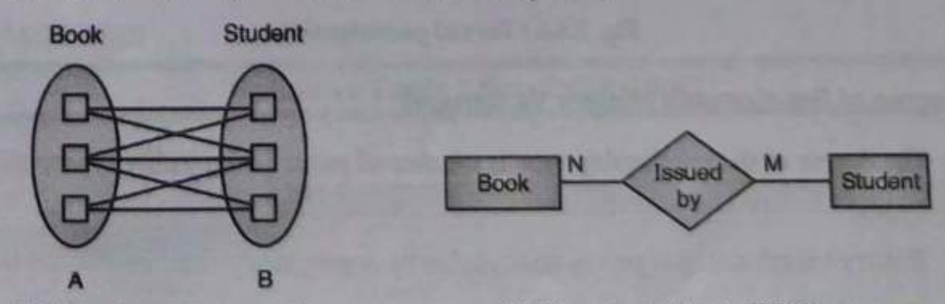


(a) Many to one mapping (b) Representation in ER diagram

Fig. 2.6.3

(iv) Many to many

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



(a) Many to many mapping (b) Representation in ER diagram

Fig. 2.6.4

Scanned by CamScanner

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

(B) Participation constraints

MU - May 2015, May 2016, Dec. 2016

University Questions

Q. Explain the terms total participation and Partial participation with example. **(May 2015, 5 Marks)**

Q. Explain the term : Total participation. **(May 2016, 2 Marks)**

Q. Explain Total participation and Partial participation. **(Dec. 2016, 5 Marks)**

(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.

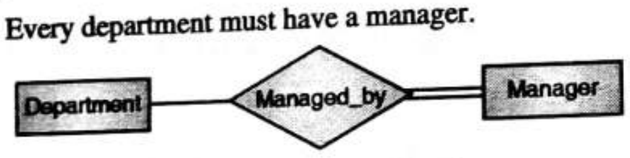


Fig. 2.6.5 : Total participation

(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. 2.6.6 : Partial participation

(C) Degree of Relationship (Binary Vs ternary)

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

(b) Types

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

  Example, Employees works for department.

Fig. 2.6.7 : Binary Relationship

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

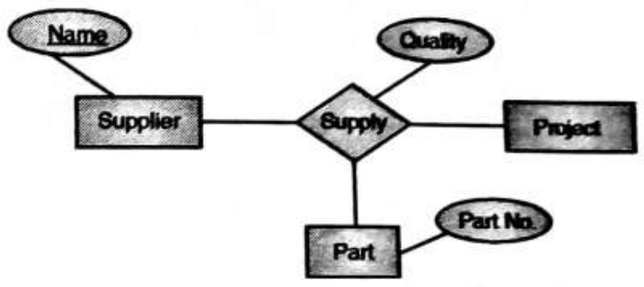


Fig. 2.6.8 : Ternary relationship

Syllabus Topic : Extended Entity-Relationship (EER) Model

2.7 Extended Entity-Relationship (ER) Model

MU - Dec. 2015

University Question

Q. Compare ER and EER models. **(Dec. 2015, 5 Marks)**

- EER model includes all the modeling concept of ER model. In addition it also includes the concept of aggregation, specialization and generalization.
- A diagrammatic technique for displaying these concepts when they arise in EER schema are the resulting schema diagrams called as EER diagrams.
- **EER Features**
  - Specialization
  - Generalization
  - Attribute Inheritance (Subclass and Superclass)
  - Aggregation

Syllabus Topic : Specialization

2.7.1 Specialization

MU - Dec. 2013, May 2014, Dec. 2014, May 2016, Dec. 2016

University Question

Q. Explain Specialization. **(Dec. 2013, Dec. 2014, May 2016, Dec. 2016, 5 Marks)**

Q. Explain Specialization with the help of an example. **(May 2014, 5 Marks)**

Scanned by CamScanner

- Top down approach of superclass / subclass relationship.
- Specialization is a process of defining a set of subclass of entity type, this entity type is called super **class of specialization.**
- The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of entity in super class.

Example

Set of subclass (Saving_Account, Current_Account) are Specialization of super class Account.

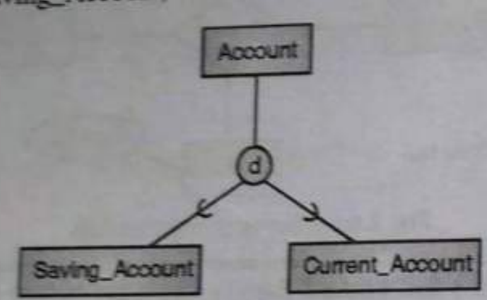


Fig. 2.7.1 : Specialization

Notation

- The subclass defined in a specialization is attached by lines to a circle which is connected to super class.
- The subset symbol on each line connecting a subclass to circle indicates the direction of super class / subclass relationship.

Specific attribute

- An attribute applied only to entities of particular subclass is called as specific attribute.

Summary of specialization
(i) Defines a set of sub class of an entity type.
(ii) Establish additional specific attributes with each subclass.
(iii) Establish additional specific relationship types between each subclass and other entity type or other subclass.

Syllabus Topic : Generalization

Generalization

MU - May 2012, Dec. 2013, May 2014, May 2016, Dec. 2016

University Question

Q. Write short note on : Generalization. **(May 2012, 2 Marks)**

Q. Explain Generalization with the help of an example. **(May 2014, 5 Marks)**

Q. Explain Generalization. **(Dec. 2013, Dec. 2014, May 2016, Dec. 2016, 2 Marks)**

- This is reverse process of specialization or this is bottom up approach of Superclass /subclass relationship.

Definition

- Generalization is a process in which we differentiate among several entity types identifying there common features and generalizing them to a single super class of which original entity type are special subclass.

Example

- Car and Bike all having several common attribute they can generalize to the super class vehicle.

Notation

- A diagrammatic notation to distinguish between generalization and specialization is used in some programming methodologies.
- Arrow pointing to generalized superclass represents generalization.
- Arrow pointing to generalized subclass represents specialization.

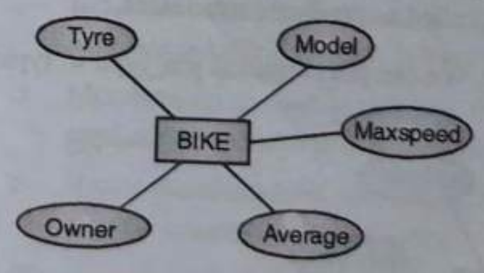


Fig. 2.7.2 : BIKE entity

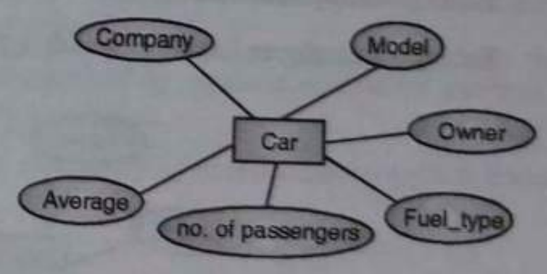


Fig. 2.7.3 : Car entity

(3) Attribute inheritance

- The attributes of higher and lower level entities created by specialization and generalizations are attributes inheritance.
- Abstraction through which relationship (aggregation) is treated as higher level entities.

Scanned by CamScanner

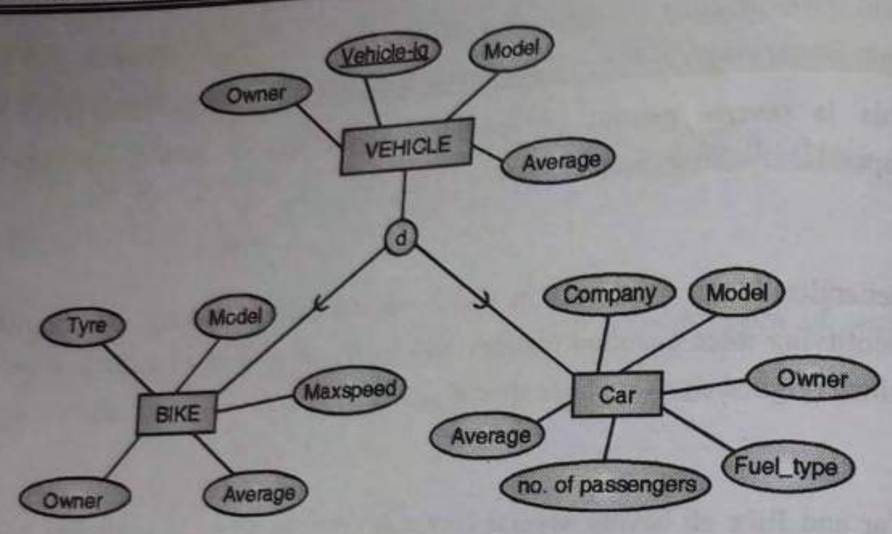


Fig. 2.7.4 : Generalized VEHICLE entity

2.7.2 Constraints and Characteristics of Specialization and Generalization

(1) Predicate defined subclass

- In specialization sometimes we can find exactly which entities will become member of specific subclass by placing a condition (or a predicate) on the value of some attributes of superclass, such subclasses are called as predicate subclasses.
- **Example :** employee has attribute job_type. We can put condition job_type = 'typist'

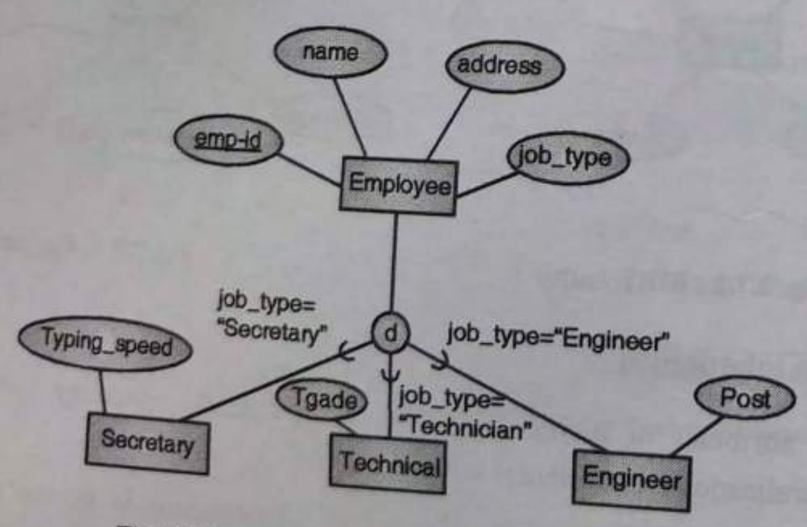


Fig. 2.7.5 : Predicate defined specialization

(2) Attribute defined specialization

- If all subclasses in a specialization have there membership condition on some attribute of superclass then the specialization itself is called as attribute defined specialization.
- This kind of attribute is called as defining attribute of specialization.

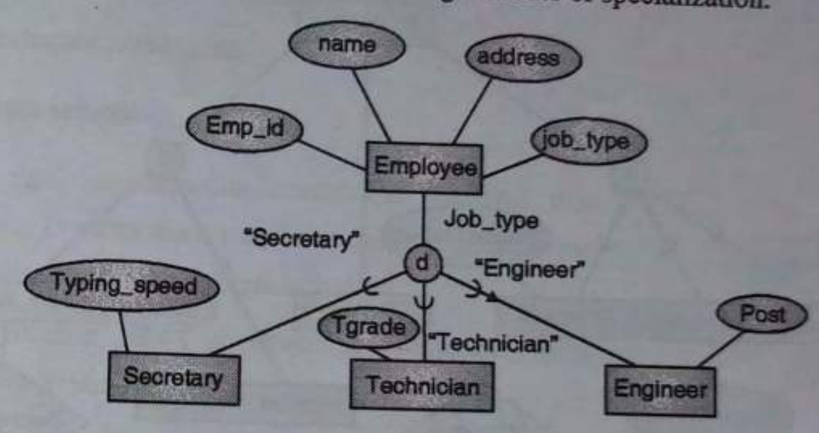


Fig. 2.7.6 : Attribute defined specialization

(3) User defined subclass

- When we do not have a predefined condition for determining membership to a subclass hence user needs to specify condition for such specialization then the subclass is known as user defined subclass.
- Membership in such a subclass is determined by database user, when database user applies the operations to add an entity to subclass.
- Hence membership is specialized for each entity by user and not by any condition that may be evaluated automatically.

(4) Other constraints specialization

(a) Disjointness

(i) Disjointness constraints

- Disjointness constraints specify that the subclasses of specialization must be disjoint that means entity can be a member of the most one subclass of specialization.
- **For Example :** New employee can become member of only one subclass like Assistant, Technician or Engineer only.
- Attribute defined Specialization determines the disjointness constraints.
- If attribute is used to define the membership then predicate should be single valued.

Scanned by CamScanner

- **Example :** In employee entity Job_Type predicate that must be single valued.
- Disjoint subclass indicated by encircled **d** as shown in Fig. 2.7.7

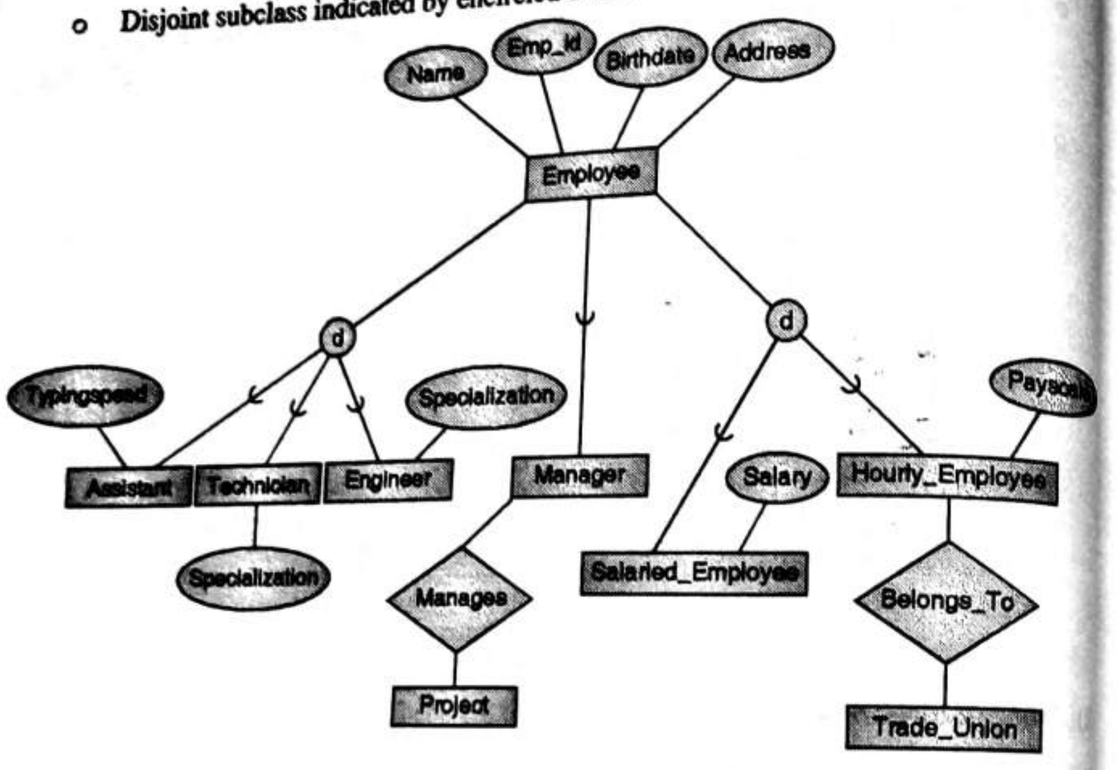


Fig. 2.7.7 : Disjoint constraints

(ii) Overlapping constraints

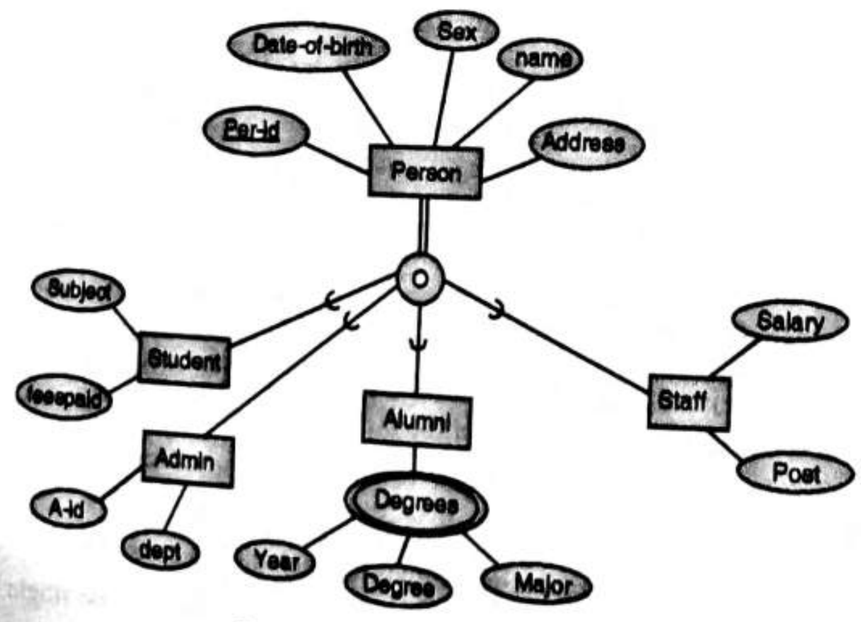


Fig. 2.7.8 : Overlapping constraint

- The subclass is not always required to be disjoint, then such set of entities can be overlaped i.e. entity may part of more the subclasses.
- **Example :** Person can become member of subclasses like student, Admin or Staff. It is possible to have person belongs to more than one subclasses.
- Disjoint subclass is indicated by encircled 'O'.

(b) Completeness constraints

(i) Total specialization

- A total specialization constraints specifies that entity in super class may be member of at least one of the subclass in the specialization.
- A double line is used to represent total specialization in EER diagram.
- **Example :** An employee must belong to salaried employees or hourly employee.

Fig. 2.7.9(a) : Total specialization

(ii) Partial specialization

- An entity can either belong to a subclass or not belongs to any subclass.
- A single line used to represent partial specialization in EER diagram.
- **Example :** An employee can be Engineer, Secretary or Technician.

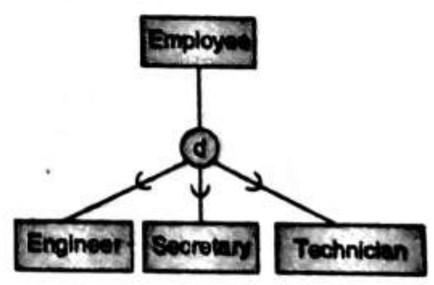


Fig. 2.7.9(b) : Partial specialization

(iii) Possible constraints

- Disjoint, total constraints
- Disjoint, partial constraints
- Overlapping, total constraints
- Overlapping, partial constraints

Scanned by CamScanner

Syllabus Topic : Aggregation

2.8 Aggregation

MU - May 2012, May 2014

University Questions

Q. Write short note on : Aggregation. **(May 2012, 5 Marks)**

Q. Explain aggregation with the help of an example. **(May 2014, 5 Marks)**

- Aggregation is meant to represent a relationship between a whole object and its component parts.
- It is used when we have to model a relationship involving entity sets and a relationship set.
- Aggregation allows us to treat a relationship set as an entity set for purpose of participation in (other) relationships.
- **Example :** A Project is sponsored by a department. This is a simple relationship.
- An Employee monitors this sponsorship (and not project or department). This is aggregation. Monitors are mapped to the table like any other relationship set.

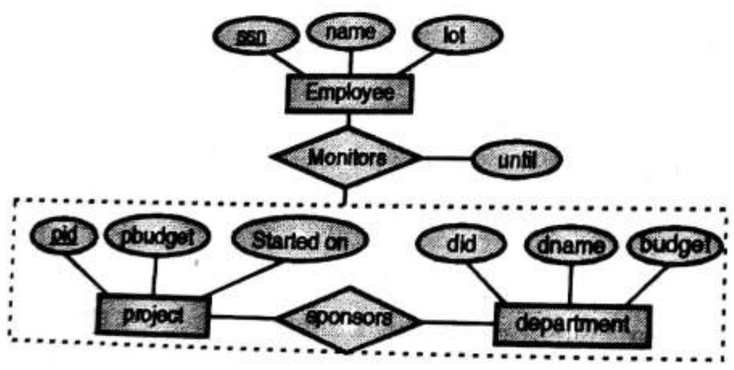


Fig. 2.8.1 : Aggregation

2.9 Solved ER Designing Examples

Example 2.9.1

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.

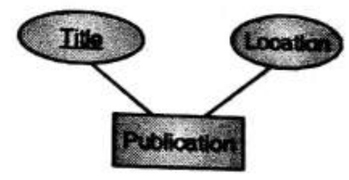
Solution :

Step 1 : Identify entities

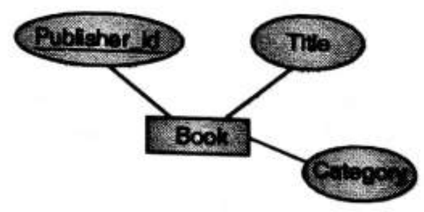
1. Publication
2. Book
3. Article
4. Journal
5. Subject
6. Author

Step 2 : Identify attributes

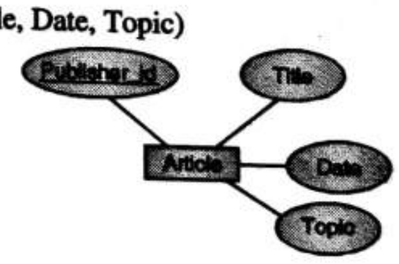
1. Publication (Title, Location)



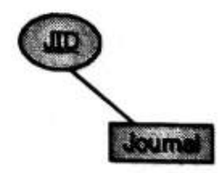
2. Book (Publisher_id, Title, Category)



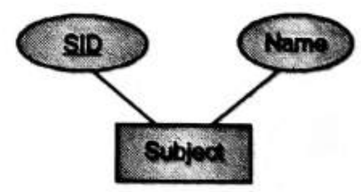
3. Article (Publisher_id, Title, Date, Topic)



4. Journal (JID)



5. Subject (SID, Name)



Scanned by CamScanner

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

Step 3 : Identify relationships

1. Articles are published in Journal.

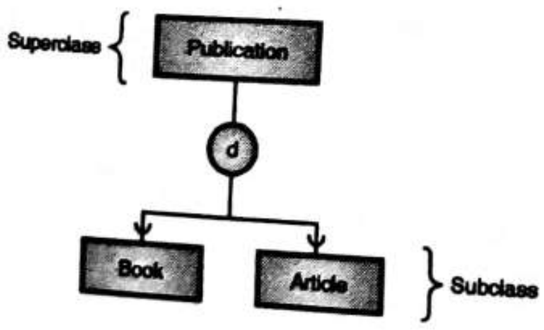
2. Publication is written by Author.

3. Publication belongs to a particular subject.

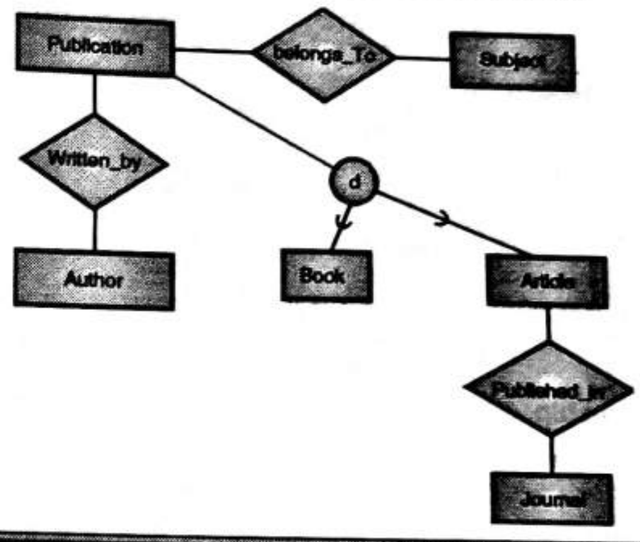
Step 4 : Identify inheritance relations

Publication can be

BOOK or ARTICLE.



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



Example 2.9.2 MU - May 2014, May 2015, 10 Marks

Construct an E-R diagram for a car-insurance company that has a set of customers, each of whom owns one or more cars. Each car has associated with it zero to any number of recorded accidents.

Solution :

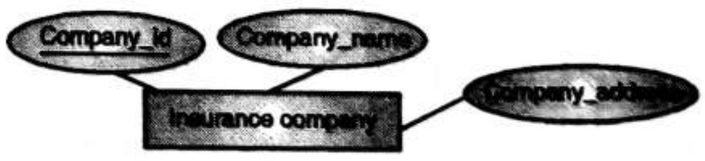
(1) Identify all entities

(a) Insurance company (b) Customer

(c) Car (d) Accidents

(2) Identify all attributes

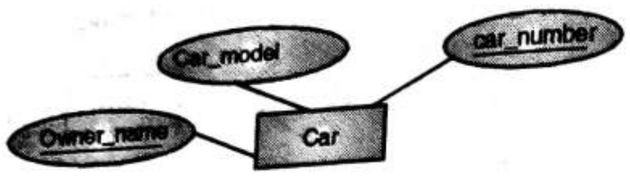
(a) Company entity



(b) Customer entity

Scanned by CamScanner

(c) Car entity



(d) Accidents

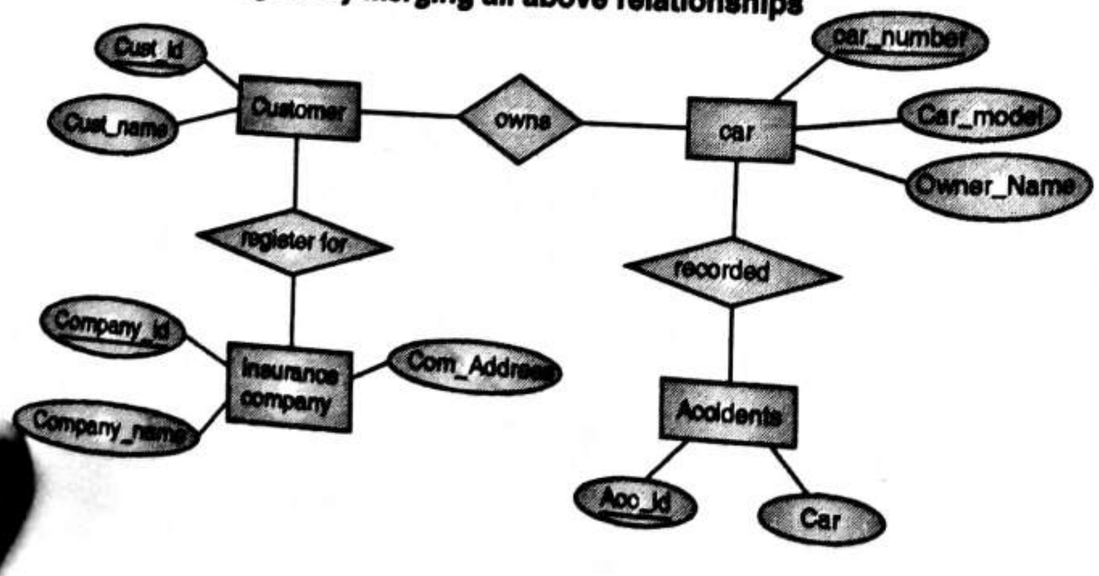
(3) Identify all relationship

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

(4) Construct ER diagram by merging all above relationships



Example 2.9.3

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.

Solution :

(1) Identify Entities

(i) Hospital

(ii) Patient

(iii) Doctors

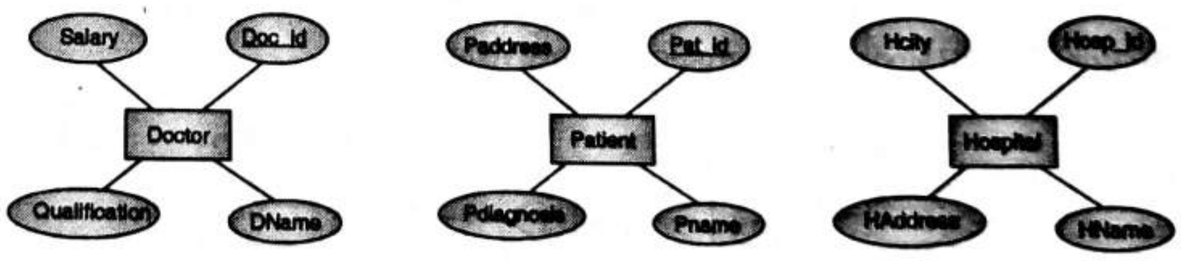
(iv) Medical-record (Record of various test and examination conducted)

(2) Identify Attributes

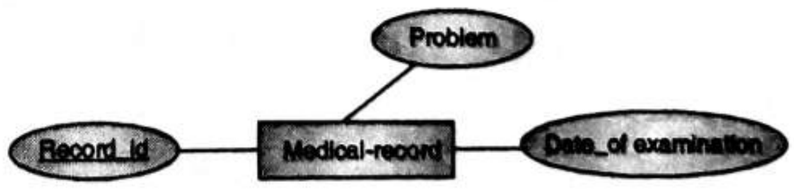
(i) Hospital (Hosp_id, HName, HAddress, Hcity)

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

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



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



Scanned by CamScanner

(3) Identify relationships

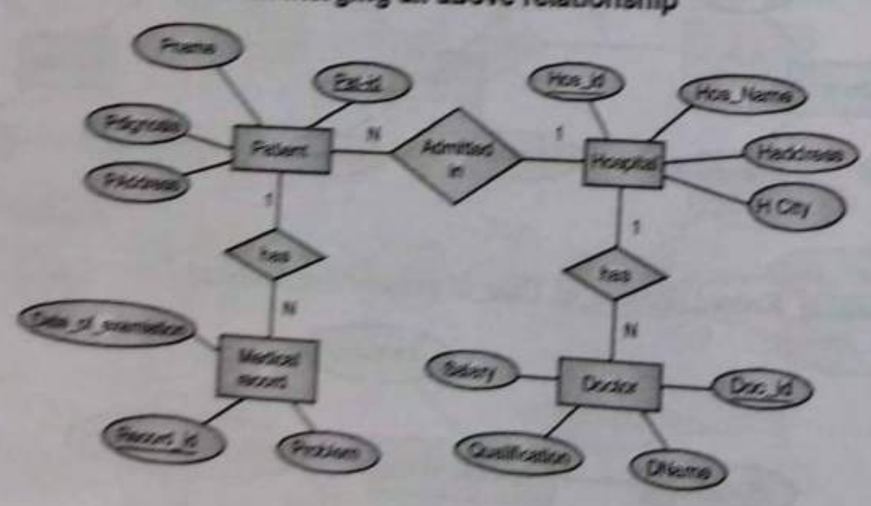
(a) Hospital has a set of patients

(b) Hospital has a set of doctors

(c) Doctors are associated with each patient

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

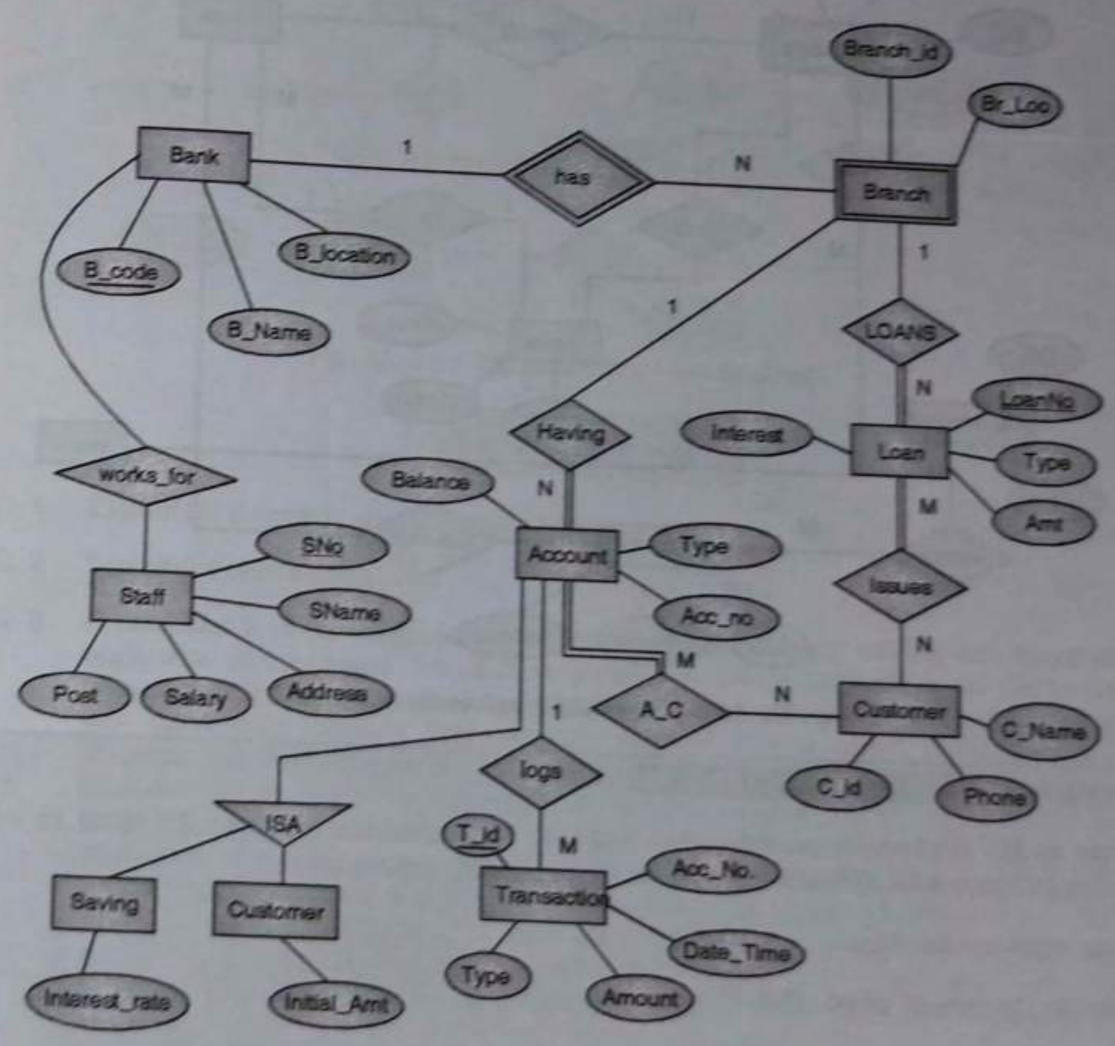
(4) Construct ER Model from merging all above relationship



Example 2.9.4

Draw ER Diagram for banking enterprise.

Solution :



Example 2.9.5 **MU - Dec 2013, Dec 2014, Dec.2016.10 Marks**

Draw ER Diagram for University database consisting four Entities Student, Department, Class and Faculty. Student has a unique id, the student can enroll for multiple classes and has a most one major. Faculty must belong to department and faculty can teach multiple classes. Each class is taught by only faculty. Every student will get grade for the class he/she has enrolled.

Scanned by CamScanner

Solution :

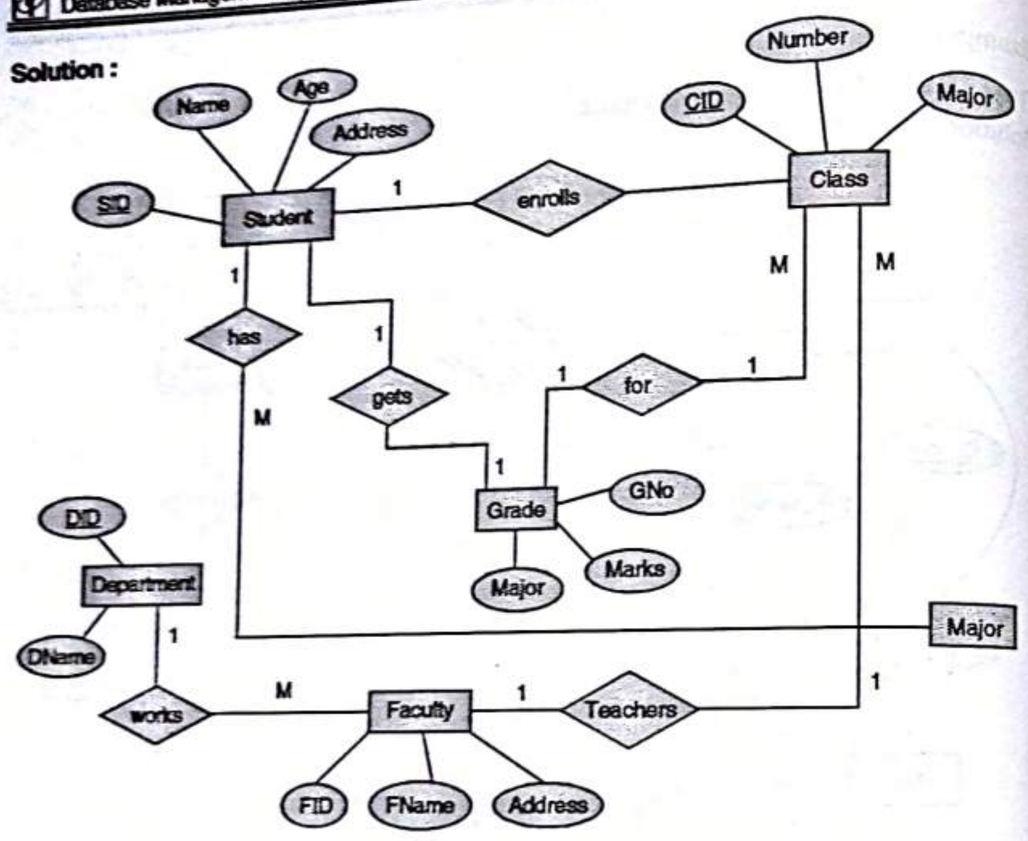


Fig : Ex. 2.9.5 ER model for university database

Example 2.9.6 **MU - Dec 2015.10 Marks**

Draw an ER diagram for the education database that contains the information about an in house company education training scheme. The relevant relations are :

Course (course-no, title)

Offering (course-no, off-no, off-date, location)

Teacher (course-no, off-no, emp-no)

Enrolment (course-no, off-no, stud-no, grade)

Employee (emp-no, emp-name, job)

Student (stud-no, stud-name, ph-no)

Solution. :

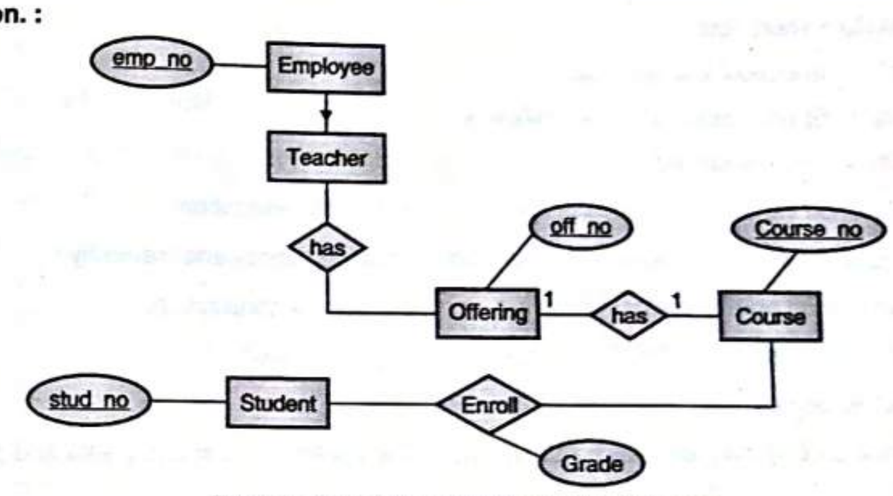


Fig : Ex. 2.9.6 ER model Employee database

Review Questions

Q. 1 Explain ER Diagrams and its components.

Q. 2 Explain the term aggregation.

Q. 3 A database is to be designed for a medium sized company dealing with industrial application of computers. The company delivers various products to its customers ranging from a single application program through to complete installation of hardware with customized software. The company employs various expert, consultants and supporting staff. All personal are employed on long term basic i.e. there are no short-term or temporary staffs. Although the company is somehow structured for administrative purposes (that is, it is divided into department headed by Department Managers) all projects are carried out in an inter-disciplinary way. For each project a project team is selected, grouping employees from different department, and a Project Manager (also an employee of the Company) is appointed who is entirely and exclusively responsible for the control of the project, quite independently of the Company hierarchy. Draw an ER Diagram. If required some more information can be assumed.

Q. 4 You have to design and implement a database that manages information about publishers, authors, and books. Some information includes : A publisher has a name and an address for the headquarters. Each publisher also has a set of branches, each branch having an address and two phone numbers. An author has a name and an address. A book is published by a publisher and has a list of authors associated with it. An author can publish several books and a book can be published by only one publisher.

(a) Draw an extended E-R diagram for the system.

(b) Take two typical queries and write them in SQL and relational algebra.

Scanned by CamScanner

Q. 5 Write a short note on :

(i) Subclass and superclass

(ii) Specialization and generalization

(iii) Type Inheritance

Q. 6 Describe various constraints of specialization and generalization.

Q. 7 Explain concept of specialization and generalization lattices and hierarchy.

Q. 8 Write short note on : Weak entity set, specialization, generalization.

Q. 9 Write short note on : Total participation, partial participation.

Q. 10 Define degree.

Q. 11 Draw an E-R diagram which models an online bookstore. List entity sets and primary keys.

Q. 12 Write short note on : Extended e-r features.

Q. 13 Write short note on : Generalization and aggregation.

Q. 14 What is strong entity and weak entity ? Explain with example.

2.10 University Questions and Answers

May 2012

Q. 1 Write short note on : Generalization and Aggregation.

(Ans. : Refer sections 2.7.1 and 2.8) **(5 Marks)**

Q. 2 What is strong entity and weak entity ? Explain with example.

(Ans. : Refer section 2.3) **(5 Marks)**

Dec. 2013

Q. 3 Explain Generalization and Specialization *(Ans. : Refer sections 2.7.1)* **(5 Marks)**

Q. 4 Draw ER Diagram for University database consisting four Entities Student, Department, Class and Faculty.

Student has a unique id, the student can enrol for multiple classes and has a most one major.

Faculty must belong to department and faculty can teach multiple classes.

Each class is taught by only faculty.

Every student will get grade for the class he/she has enrolled.

(Ans. : Refer Ex.2.9.5) **(10 Marks)**

May 2014

Q. 5 Define Strong entity set. *(Ans. : Refer section 2.3(a))* **(2 Marks)**

Q. 6 Explain generalization, specialization and aggregation with the help of an example.

(Ans. : Refer section 2.7.1 and 2.8) **(10 Marks)**

Q. 7 Construct on E-R diagram for a car-insurance company that has a set of customers each of whom owns one or more cars. Each car has associated with it zero to any number of recorded accidents. *(Ans. : Refer Ex.2.9.2)* **(10 Marks)**

Dec. 2014

Q. 8 Define Derived attribute. *(Ans. : Refer section 2.4(c))* **(2 Marks)**

Q. 9 Explain generalization and specialization.

(Ans. : Refer sections 2.7.1) **(5 Marks)**

Q. 10 Draw E-R diagram for university database consisting of four entities: student, department, class, faculty, 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 teach multiple classes. Each class is taught by only one faculty. Every student will get grade for the class he/she has enrolled. *(Ans. : Refer Ex.2.9.5)* **(10 Marks)**

May 2015

Q.11 Explain the terms total participation and Partial participation with example

(Ans. : Refer section 2.6(B)) **(5 Marks)**

Q. 12 Construct on E-R diagram for a car-insurance company that has a set of customers each of whom owns one or more cars. Each car has associated with it zero to any number of recorded accidents. *(Ans. : Refer Ex.2.9.2)* **(10 Marks)**

Q. 13 We can convert any entity set to a strong entity set by simply adding appropriates attributes. Why then, do we have weak entity sets ? *(Ans.: Refer section 2.3)* **(12 Marks)**

Dec. 2015

Q. 14 Compare ER and EER models. *(Ans.: Refer section 2.2 and 2.7)* **(5 Marks)**

Q. 15 Draw an ER diagram for the education database that contains the information about an in house company education training scheme. The relevant relations are :

Course (course-no, title)

Offering (course-no, off-no, off-date, location)

Scanned by CamScanner

Teacher (course-no, off-no, emp-no)

Enrolment (course-no, off-no, stud-no, grade)

Employee (emp-no, emp-name, job)

Student (stud-no, stud-name, ph-no) *(Ans. : Refer Ex.2.9.6)* **(10 Marks)**

May 2016

Q. 3(a)(iv) Explain the term : Total participation. *(Ans.: Refer section 2.6(B))* **(2 Marks)**

Q.1(b) Explain generalization and specialization. *(Ans.: Refer section 2.7.1)*

Dec. 2016

Q. 1(a) Explain generalization and specialization. *(Ans.: Refer section 2.7.1)*

Q. 1(c) Explain Total participation and Partial participation.

(Ans.: Refer section 2.6(B)) **(5 Marks)**

Q. 3(a) Draw E-R diagram for university database consisting of four entities: student, department, class, faculty, 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 teach multiple classes. Each class is taught by only one faculty. Every student will get grade for the class he/she has enrolled. *(Ans. : Refer Ex.2.9.5)* **(10 Marks)**

May 2017

Q. 1(d) Explain different types of attributes in ER Model *(Ans.: Refer Section 2.4)* **(5 Marks)**

□□□

CHAPTER 3

Module III

Relational Data Model

Course Outcome

After learning this topic students will be able to,

1. Understand Relational Model basic concepts.
2. Convert ER diagram into tables (Relations)

Syllabus

Introduction to Relational Model, Relational Model Constraints and Relational Database Schemas, Concept of Keys : Primary Key, Secondary key, Foreign Key, Mapping the ER and EER Model to the Relational Model.

Syllabus Topic : Introduction to Relational Model

3.1 Relational Model

1. 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**.

Scanned by CamScanner

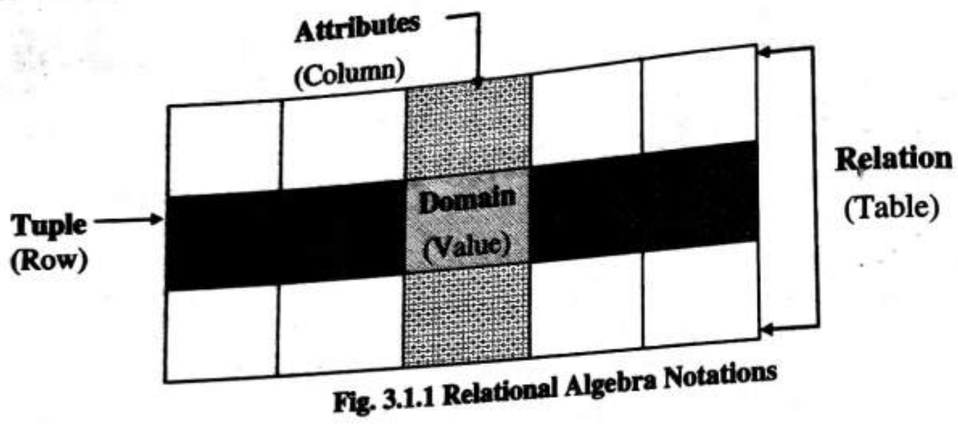


Fig. 3.1.1 Relational Algebra Notations

2. Relation (Table)

Q. Define Relation.

- 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.
- The table containing rows and columns represents entity in relational model, it is called as Relation.
- This concept doesn't represent how the data is stored in the physical memory of computer system.
- The relation can contain data about a single entity or relationship between two entities.

Characteristics of Relation

- A table composed of rows and columns.
- Each table in a database has its unique *table name*.
- Each table row (tuple) represents a single entity occurrence within the entity set.
- 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,

The student relation can be shown as given below,

Id	Name	Age	Class	Branch
105	Mahesh	25	BE	IT
106	Suhas	28	FE	CS
107	Jay	29	SE	CS
108	Sachin	30	TE	EXTC

3. Attributes (Column)

Q. Define Attributes

- Relation has its own properties which describes that relation (table) such properties are known as attributes.
- In relational model, the column in relation (table) or field of data is also called as Attribute.
- Every table must have at least one column in it.
- The single attribute will contains the similar type of data of all entities in relation.

Name
Mahesh
Suhas
Jay
Sachin

- It is not possible to have multiple columns with same column name in the same relation. But it is possible to have multiple columns with same column name in two different relations.
- The SQL standard does not specify any maximum number of columns in a table.

Example,

- The Name attribute in above student relation will contains the name of all student entities in student relation.

4. Tuple (Row / Records)

- A single row in relational table which contains all the information about a single entity is called as Tuple.
- The single row in relation (Table) is called as Tuple.
- Each horizontal row of the student table represents a Student tuple.
- A table can have any number of rows in it.

Stud_Id	Name	Age	Std	Div
105	Mahesh	25	BE	A

Example,

The above tuple contains all data about the Id 105, student entity in student relation.

5. Domain (Data Value)

- The intersection column and row in a relational table which represents data of entity is called as Domain.

Scanned by CamScanner

- Every column in a table has a set of data values that are allowed for that column which is called as Domain.
- In a relational table a domain can have a single value or no (Null) value.
- The single domain will contains the specific data of single entities in relation.

Name
Mahesh

Example,

- The Name attribute of tuple id 105 will contains the name of all student with id 105 in student relation.

Syllabus Topic : Relational Database Schema

3.2 Relational Database Schema

Q. Write a note on Relational Database Schema.
Q. Compare Relational Database Schema and Relational Schema with example.

(1) Introduction

- The term schema refers to the organization or structure of data in relational database.
- The relational schema describes structure of relation (i.e. table) and relational database schema explains the structure of relational database.

(2) Relational Schema

- Relation schema consists of a number of attributes associated with relation.

Example,

EMPLOYEE	(Ssn, Ename, Bdate, Address, Dnumber)
DEPARTMENT	(Dnumber, Dname, Dmgr_ssn)
DEPT_LOCATIONS	(Dnumber, Dlocation)
PROJECT	(Pnumber, Pname, Plocation, Dnum)
WORKS_FOR	(Ssn, Pnumber, Hours)

Fig. 3.2.1 Sample Relational Schema

(3) Relational Database Schema

- Relational database schema consists of a number of relation schemas associated with that database.

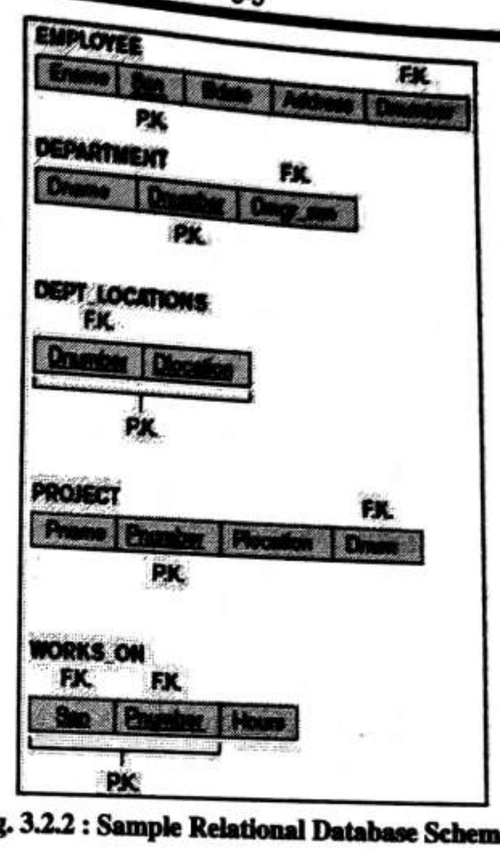


Fig. 3.2.2 : Sample Relational Database Schema

(4) Comparison

- The attributes are grouped to form a relation schema by mapping a conceptual data model i.e. ER or EER data model to relational schema.
- The relational mapping will identify entity types and relationship types and their respective attributes.
- The relation schema consists of a number of attributes in relation while the relational database schema consists of a number of relation schemas in the corresponding database.

Relational Schema (Table Structure)	**Relational Database Schema** (Database Structure)
$R_1 (A_1, A_2,.., A_n)$ $R_2 (B_1, B_2,.., B_n)$ ... $R_m (C_1, C_2,.., C_n)$	$R_1, R_2 ,..., R_n$

Where,

$R_1, R_2,... R_n$ = Relation or Tables

A, B, C = Attributes or Columns

Scanned by CamScanner

Syllabus Topic : Relational Model Constraints

3.3 Relational Model Constraints

MU - Dec. 2016

University Question

Q Explain different integrity constraints. **(Dec. 2016, 10 Marks)**

(1) Introduction

- 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 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 below,
  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.

(2) Types of Relational Constraints

- Domain Relational Constraints
- Entity Relational Constraints
- Referential Relational Constraints

3.4 Domain Relational Constraint

MU - Dec. 2016

University Question

Q. Explain different integrity constraints. **(Dec. 2016, 10 Marks)**

1. Introduction

- 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.

2. Types of Domain Constraints

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

A. Required Data Constraint / Nullness Constraint

- The database may have some attributes mandatory like user registration must have user 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.

```
Student_Name varchar(100) NOT NULL
```

Therefore now, the Student_Name column in the STUDENT table is a required data column. It is not possible to insert Null value in Student_Name column of Student table.

- The DBMS can prevent user from inserting NULL values in any column with help of such constraints.

B. Check Constraint

- The **check constraint** is used to ensure that attribute value satisfies specific condition as specified by data requirements or user.
- 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'.

```
Student_Gender varchar(1) CHECK (gender IN ('M','F'))
```

C. Default Keyword

- Default keyword is used to add a default specified value, if attribute value is not provided by user.
- It avoids the addition of NULL value to the database by inserting default value as specified by the developer while creating a table.

Example,

Table with customer entity having name and gender.

Scanned by CamScanner

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

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

3.5 Entity Integrity Constraints

MU - Dec. 2016

University Question

Q. Explain different integrity constraints. (Dec.2016, 10 Marks)

1. Introduction

- 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 primary key of table.

2. Types of Entity Constraints

- Unique Constraint
- Primary Key Constraint

A. Unique Constraint / Unique Key

- 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.

Example,

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

B. Primary Key Constraint

MU - May 2016

University Question

Q. Explain the term : Primary Key. (May 2016, 2 Marks)

- 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 (Unique constraint+ Not Null constraint).
- 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 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
```

3.6 Referential Integrity / Foreign Key

MU - Dec. 2015

University Question

Q. Discuss what is meant by term : Referential integrity. (Dec. 2015, 3 Marks)

1. Introduction

- A value appearing in a 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.

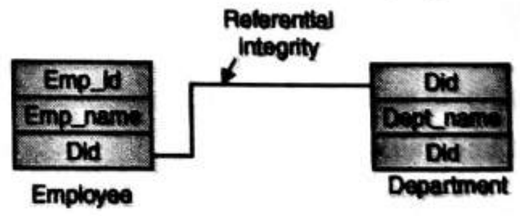


Fig. 3.6.1 : Referential integrity

Employee 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 Employee table has Did as foreign key reference to Did column in Department table this is called as referential integrity.

Scanned by CamScanner

- Here we are forcing the database to check the value of Did column from the department table while inserting any value in Employee table. This helps to maintain data consistency.

2. Foreign key violations in SQL

- If any row in EMP table is added with 'Did' value which is not there in department table the insert statement will give foreign key violation error.
- In above tables we will refer Department as parent table (as it is containing Primary key) and Employee table as Child table (as it is containing Foreign Key).
- There are 4 problems causes the foreign key violations,
- **Adding new tuple to Child Table (Add Child)**
- If we try to add an employee with Did 70 to employee table (Child Table), it will return foreign key violation error. As Did 70 is not there in Department table (Parent table).

```
INSERT INTO Employee
VALUES (11,'Devid', 70);
```

Output

ORA-02291 : integrity constraint (Employee.FK_Employee) violated - parent key not found

Adding new employee		
Emp_Id	Emp_name	Did
11	Devid	70

- This functionality helps to maintain data consistency in database.

Updating tuple from Child Table :

- If we try to update an employee Emp_Id = 2 with Did as 70 to employee table (Child Table), it will return foreign key violation error. As Did 70 is not there in Department table (Parent table).

```
UPDATE Employee
SET Did = 70
WHERE Emp_Id=2;
```

Output

ORA-02291 : Integrity constraint (Employee.FK_Employee) violated - parent key not found.

- This functionality helps to maintain data consistency in database.

Deleting tuple from Parent Table

- If we try to delete department of Did = 10 from Department table (Parent table), it will return foreign key violation error. As there are few employees working in department with Did =10.

```
DELETE Department
WHERE Did=10;
```

Output :

ORA-02292: integrity constraint (Employee.FK_Employee) violated - child record found

- This functionality will creates limitation for deletion of parent record if it has some associated child records.

Updating tuple from Child Table

- If we try to update department of Did = 10 with Did = 70, it will return foreign key violation error. As there are few employees still working in department with Did =10.

```
UPDATE Department
SET Did = 70
WHERE Did = 10;
```

Output

ORA-02292 : integrity constraint (Employee.FK_Employee) violated - child record found

- This functionality will creates limitation for updating parent record if it has some associated child records.

3. Delete-Update (DU) rules to solve problem of foreign key violation

- If any row in EMP table is added with 'Did' value which is not there in department table the insert statement will give foreign key violation error.
- This rule can be enforced as given below,

```
Create Table Employee
(
  Eid varchar (50) Primary Key,
  ....
        Did varchar (50) foreign key references department (Did)
        On delete CASCADE
        On update CASCADE
  )
```

Scanned by CamScanner

(a) NO ACTION / RESTRICT

- This clause will discards the delete or update operation on the parent table.
- In this case the database engine will not allow user to delete the row and using FK violation error.
- The RESTRICT rule will not allow you to delete a row from the parent table although as there is corresponding row present in child table.
- Foreign key (Did) references department :
  - On delete RESTRICT
  - On Update RESTRICT
- The database engine will give the error and the delete action on the row in the parent table is ignored.
- Deletion of department is not allowed as there is some employees are present in that department.

(b) CASCADE

- Corresponding rows are deleted from the referencing table (Child table), if that row is deleted from the parent table.
- Foreign key (Did) references department :
  - On delete CASCADE
  - On Update CASCADE
- If a department is deleted then all the employee records that refers to the deleted department are also been deleted.

(c) SET NULL

- Foreign key data value is set to NULL, if the corresponding row in the parent table is deleted.
- For this constraint to execute, the foreign key columns must be nullable.
- Foreign key (Did) references department :
  - On delete SET NULL
  - On Update SET NULL
- Insert Null value of did in the place of deleted did in employee table.

(d) SET DEFAULT

- Foreign key data values refers to non-existing foreign key are set to their default values.
- For this constraint to execute, all foreign key columns must have default definitions.

- Foreign key (Did) references department :
  - On delete SET DEFAULT
  - On Update SET DEFAULT
- If a column is null able, and there is no explicit default value set, NULL becomes the implicit default value of the column.
- Insert any default value of 'did' (which exists in the departing table) in the place of deleted 'Did'.

4. Self-referential relations

- A foreign key constraint can reference some other columns in same tables.
- It form self-referential relations.
- For Example,

  EMPLOYEE (employee_id, employee_name, manager_id)
- Because the manager is also employee, there is a foreign key relationship from the **manager_id** column to the **employee_id** column in same table.

Syllabus Topic : Mapping the ER and EER Model to the Relational Model

3.7 Concept of Keys

Q. Explain concept of keys relational Table.

- The column value that uniquely identifies a single record in a table called as key of table.
- An attribute or set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity set.
- 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)

Key type	Definition
Primary key	A candidate key selected to uniquely identify all other attribute values in any given row. (Explained in details in previous section Entity Integrity)
Secondary key	An attribute (or combination of attributes) used strictly for data retrieval purposes.

Scanned by CamScanner

Key type	Definition
Foreign key	An attribute (or combination of attributes) in one table whose values must either match the primary key in another table or be null. (Explained in details in previous section Referential Integrity)

Syllabus Topic : Mapping the ER and EER Model to the Relational Model

3.8 Mapping Entities to Tables

MU - Dec. 2013, May 2017

University Question

Q. Explain the steps of an algorithm for ER to relational mapping. (Dec, 2013, 10 Marks)

Q. Explain the algorithm to map ER and EER model to relational model in detail. (May 2017, 10 Marks)

(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.

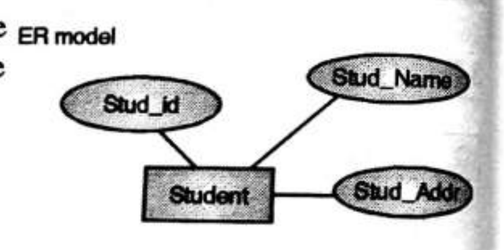


Fig. 3.8.1 : Regular entity

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).

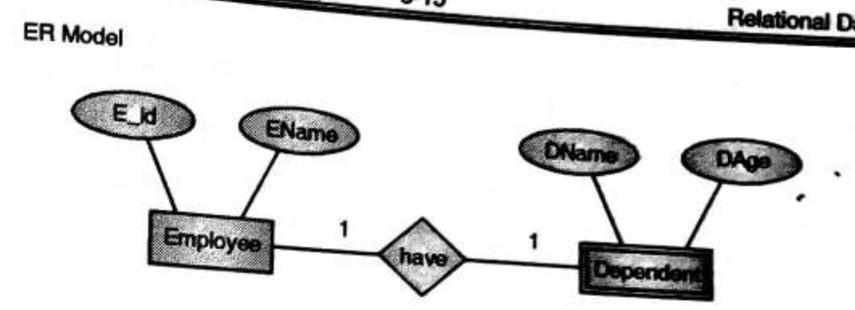


Fig. 3.8.2 : Weak entity

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

3.9 Mapping 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. 3.9.1 : 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 below example composite attribute 'Name' is converted to three columns in object model by avoiding actual attribute 'Name'.

Scanned by CamScanner

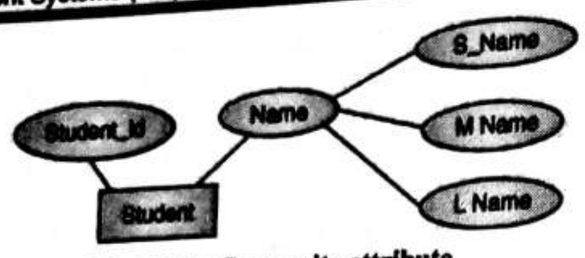


Fig. 3.9.2 : Composite attribute

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. 3.9.3.

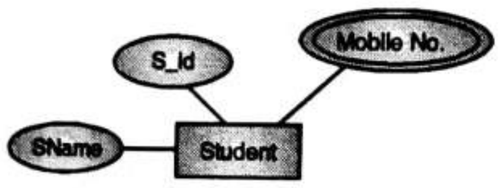
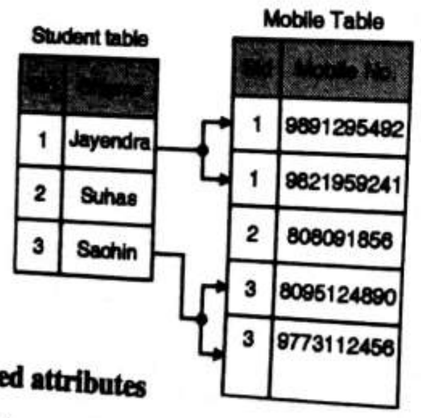


Fig. 3.9.3 : Multi valued attribute



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.

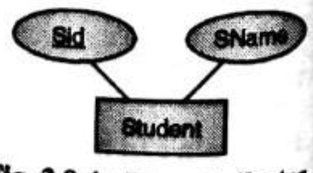


Fig. 3.9.4 : Key attributes

Student table

Sid	SName
1	Deepak
2	Vaibhav
3	Yogita
4	Bency

3.10 Mapping Relationships

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.

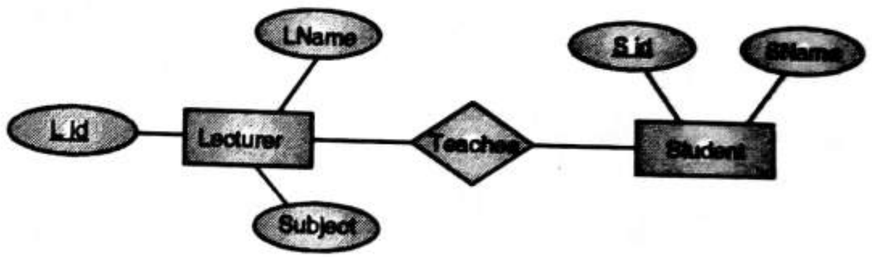


Fig. 3.10.1 : Foreign key approach

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

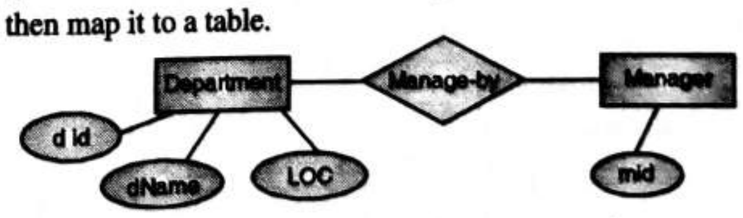


Fig. 3.10.2 : Merged relationship approach

Scanned by CamScanner

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.

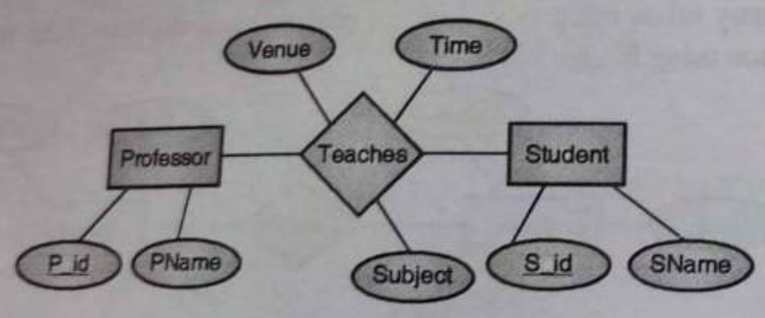


Fig. 3.10.3 : Cross reference approach

Professor table	
Pid	PName
1	Om
2	Nitin

Student table	
Sid	SName
1	Snehal
2	Tanmay

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

3.11 Mapping Inheritance constraints

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

- **Tables :** Each super class and subclass entity sets represents table in relational model.
- **Columns :** Attributes of entity set is converted to the columns of the tables in relational model.
- **Primary key :** The primary key column of super class is also added to all subclasses and treated as a primary key column for all tables in relational model.

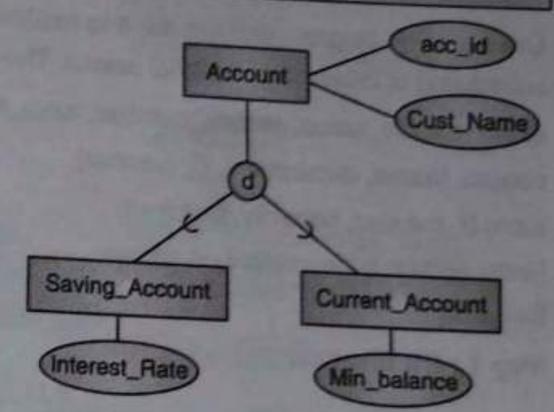


Fig. 3.11.1 : Inheritance relation

Account table	
acc_id	Cust_Name
1	Snehal
2	Tanmay
3	Nikhil

Saving_Account table	
acc_id	Interest_Rate
1	10
2	12
3	15

Current_Account	
acc_id	Min balance
1	1000
2	2000
3	500

Scanned by CamScanner

3.12 Solved Examples

Example 3.12.1

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.

Solution :

Step 1 : ER diagram

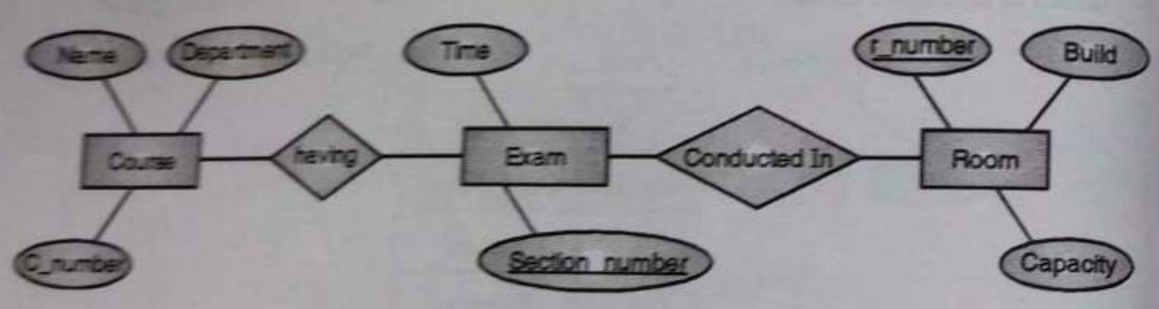


Fig. Ex. 3.12.1

Step 2 : Mapping to Tables

Course (C_number, name, department)

Exam (Section_number, time, C_number,)

Room (R_number, building, capacity, Section_number)

Example 3.12.2

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.

Solution :

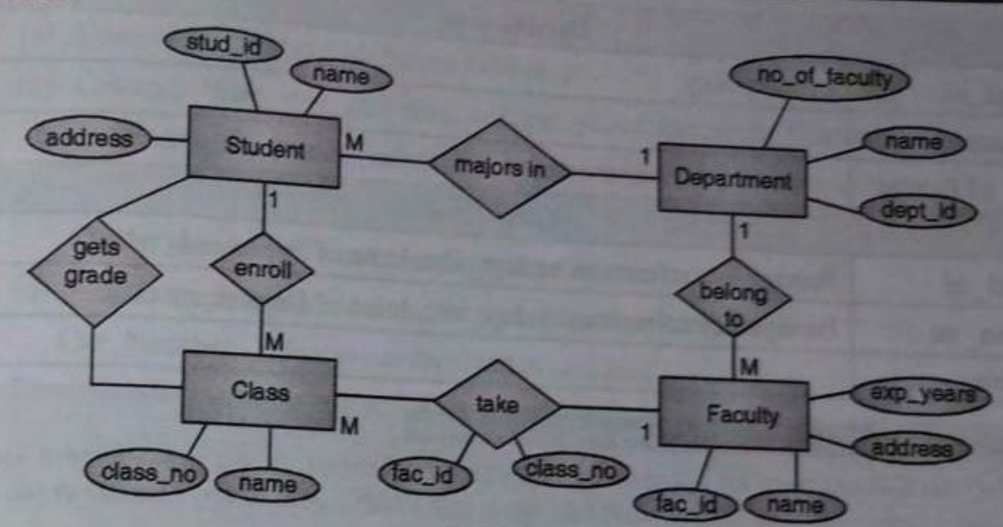


Fig.Ex.3.12.2

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

Scanned by CamScanner

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	

Example 3.12.3 MU - May 2014, May 2016, 10 Marks

Construct on E-R diagram for a car-insurance company that has a set of customers each of whom owns one or more cars. Each car has associated with it zero to any number of recorded accidents.

Solution :

(1) Identify all entities

(a) Insurance company (b) Customer

(c) Car (d) Accidents

(2) Construct ER diagram by merging all above relationships

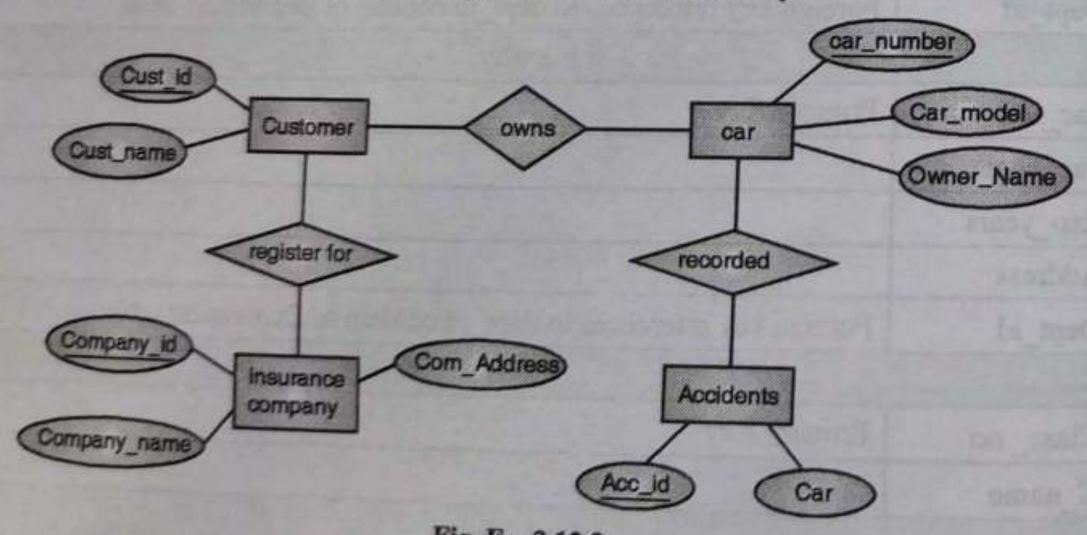


Fig. Ex. 3.12.3

(1) Mapping Entities

(a) Company (Company_id, Name, Address)

(b) Customer (Customer_id, Name, Address, phone)

(c) Car (Car_Number, Car_Model, Owner)

(d) Accidents (Accident_Id, Location, date, time)

(2) Mapping Relations

(a) Company (Company_id, Name, Address)

(b) Customer (Customer_id, Name, Address, phone, Insurance_Company)

Insurance_Company - refers to customers registered insurance company

(c) Car (Car_Number, Car_Model, Owner_Id)

Owner_Id - refers to customer id owns that car.

(d) Accidents (Accident_Id, Car_Number, Location, date, time)

Car_Number - refers to car involved in accident.

(3) Final Relational Schema

(a) Company (Company_id, Name, Address)

(b) Customer (Customer_id, Name, Address, phone, Insurance_Company)

(c) Car (Car_Number, Car_Model, Owner_Id)

(d) Accidents (Accident_Id, Car_Number, Location, date, time)

Review Questions

Q. 1 Explain all types of integrity with help of examples.

Q. 2 Write in brief about Referential Integrity problems.

Q. 3 Write short note about Delete update rules in Referential integrity.

Q. 4 Explain the term 'Referential Integrity' and its relation with foreign key.

Q. 5 Write short notes on : Integrity constraints in RDBMS.

Q. 6 Explain four update situations that can corrupt the referential integrity.

Q. 7 Explain terms primary key and foreign key with example.

Q. 8 Explain column check constraint in SQL. How it can be applied to columns of a table.

Q. 9 Why are entity integrity and referential integrity important in a database ?

3.12 University Questions and Answers

Dec. 2013

Q. 1 Explain the steps of an algorithm for ER to relational mapping.

(Ans. : Refer section 3.8) **(10 Marks)**

Scanned by CamScanner

May 2014

Q. 2 Construct on E-R diagram for a car-insurance company that has a set of customers each of whom owns one or more cars. Each car has associated with it zero to any number of recorded accidents. *(Ans.: Refer Ex. 3.12.3)* **(10 Marks)**

Dec. 2015

Q. 3 Discuss what is meant by term : Referential integrity. *(Ans. : Refer section 3.6)* **(3 Marks)**

May 2016

Q. 3(a)(i) Explain the term : Primary Key. *(Ans. : Refer section 3.5(B))* **(2 Marks)**

Q. 2(b) Draw E-R diagram for a car-insurance company that has a set of customers. Each of customer has one or more cars. Each car is zero or more accident record. *(Ans. : Refer Ex. 3.12.3)* **(10 Marks)**

Dec. 2016

Q. 2(b) Explain different integrity constraints. *(Ans.: Refer Sections 3.3, 3.4 and 3.5)* **(10 Marks)**

May 2017

Q. 2(a) Explain the algorithm to map ER and EER model to relational model in detail. *(Ans. : Refer Section 3.8)* **(10 Marks)**

□□□

CHAPTER 4

Module III

Relational Algebra

Course Outcome

After learning this topic students will be able to,

1. Construct Relational algebra queries.

Syllabus

- Introduction to Relational Algebra
- Relational Algebra expressions for,
  - o Unary Relational Operations
  - o Set Theory operations
  - o Binary Relational operation
- Relational Algebra Queries

Syllabus Topic : Introduction to Relational Algebra

4.1 Relational Algebra

MU - Dec. 2014, Dec. 2016, May 2017

University Questions

Q. Explain any four relational algebra operations with proper examples. **(Dec. 2014, Dec. 2016, 10 Marks)**

Q. Discuss fundamental operations in relational algebra. **(May 2017, 10 Marks)**

- Relational algebra becomes popular after the publication of E.F. Codd's relational model of data in 1970.

Scanned by CamScanner

Chapter 4 : Relational Algebra 4-1 to 4-22

✓ **Syllabus Topic :** Introduction to Relational Algebra 1

4.1 Relational Algebra 1

✓ **Syllabus Topic :** Unary Relational Operations 2

4.2 Selection Operation (σ) 2

4.3 Projection Operation (π) 4

4.4 Rename Operation (ρ) 5

✓ **Syllabus Topic :** SET Theory Operations 6

4.5 SET Operation 6

4.5.1 Union Operator 7

4.5.2 Intersect Operator 8

4.5.3 Difference Operator 10

✓ **Syllabus Topic :** Binary Relational Operation 11

4.6 Cross Product / Cartesian product 11

4.7 Join Operation (⋈θ) 12

4.8 Relational Division Operator 16

4.9 Operator Precedence 18

✓ **Syllabus Topic :** Relational Algebra Queries 18

4.10 Relational Algebra Queries - Solved Examples 18

4.11 University Questions and Answers 21

Scanned by CamScanner

CHAPTER 4

Module III

Relational Algebra

Course Outcome

After learning this topic students will be able to,

1. Construct Relational algebra queries.

Syllabus

- Introduction to Relational Algebra
- Relational Algebra expressions for,
  - Unary Relational Operations
  - Set Theory operations
  - Binary Relational operation
- Relational Algebra Queries

Syllabus Topic : Introduction to Relational Algebra

4.1 Relational Algebra

MU - Dec. 2014, Dec. 2016, May 2017

University Questions

Q. Explain any four relational algebra operations with proper examples. **(Dec. 2014, Dec. 2016, 10 Marks)**

Q. Discuss fundamental operations in relational algebra. **(May 2017, 10 Marks)**

– Relational algebra becomes popular after the publication of E.F. Codd's relational model of data in 1970.

Scanned by CamScanner

- It is procedural language useful for representing query execution plan and relatively close to SQL.
- Relational algebra is set of operations which accept one relation and produces new relation as a result.
- This query is applied to tables/relations and output is also a table/relation.
- Fundamental operations of Relational Algebra,
  - Unary Relational Operations
    - Project operation (π)
    - Select operation (σ)
    - Rename operation (ρ)
  - SET Theory Operations
    - Union operation ($\cup$)
    - Difference operation (-)
    - Intersection operation ($\cap$)
  - Binary Operations
    - Join operations ($\bowtie$)
    - Cartesian product operations (X)
    - Division operation (%)

Syllabus Topic : Unary Relational Operations

4.2 Selection Operation (σ)

University Question MU - Dec. 2015

Q. Describe the following relational algebra operation : Select. **(Dec. 2015, 3 Marks)**

Overview

- This operator is used to select some rows from table which satisfy particular selection condition given in selection operation.
- Selection operator selects a set of tuples that satisfy a selection predicate or condition.
- Output of query is exactly same as input schema of table.

This is unary relational operator having only one input table.

b) Syntax

$\sigma_{\text{<attribute_name><comparison_operator><constant_value>}}$ (Input_Table_Name)

Where,

Attribute_name : Name of column in table

Comparison_operator : =, <, < =, >, > =, < >

c) Example

Eid	Ename	Age	Salary
1	Suhas	24	50000
2	Jayendra	24	15000
3	Sachin	25	52000
4	Mahesh	23	41000
5	Satish	34	25000
6	Suma	54	50000
7	Raj	69	45000
8	Anu	74	50000

Query : Select all employees having age below 30 years.

Solution : $\sigma_{age < 30}$ **(Employee)**

Eid	Ename	Age	Salary
1	Suhas	24	50000
2	Jayendra	24	15000
3	Sachin	25	52000
4	Mahesh	23	41000

d) Combining multiple conditions

We can have more than one predicate by using logical connectives like AND ($\wedge$ OR ($\vee$).

Query : Select all employees having salary above 5000 and age above 65.

Solution : $\sigma_{salary > 5000 \wedge age > 65}$ **(Employee)**

Scanned by CamScanner

Eid	Ename	Age	Salary
7	Raj	69	45000
8	Anu	74	50000

Query : Select all employees with either salary above 50000 or age above 65.

Solution : $\sigma_{salary > 50000 \vee age > 65}$ **(Employee)**

Eid	Ename	Age	Salary
3	Sachin	25	52000
7	Raj	69	45000
8	Anu	74	50000

.3 Projection Operation (π)

MU - May 2012, Dec. 2013, Dec. 2015

University Questions

. Explain Project Relational algebra operators with suitable examples. **(May 2012, Dec 2013, 2 Marks)**

. Describe the following Relational algebra operation : Project **(Dec.2015, 3 Marks)**

Overview

- This operator is used for selecting some of many columns in table to display in result set.
- Projection operator can select a column or set of columns of table to be display in output of query.
- We can select only few columns or all columns of a table as per requirements.
- This is unary relational operator having only one input table.

yntax

$$\pi_{<column_list>} (Input_Table_Name)$$

xample

uery : Find salary and age of all Employees.

olution : $\pi_{age, salary}$ **(Employee)**

Age	Salary
24	50000
24	15000
25	52000
23	41000
34	25000
54	50000
69	45000
74	50000

Query : Find salary of all employees having age less than 25.

Solution : $\pi_{age, salary}$ **(Employee)**

Age	Salary
24	50000
24	15000
23	41000

4.4 Rename Operation (ρ)

MU - May 2012, Dec. 2013, May 2014

University Question

Q. Explain Rename Relational algebra operators with suitable examples. **(May 2012, Dec. 2013, May 2014, 2 Marks)**

a) Overview

- We can give alternative name to any column (attribute) or any table of query expressions using operator called as RENAME operator.
- This operator is specially introduced to select specific column from joined table (set of two or more tables) containing multiple columns of same column name.
- Rename operator, denoted by the lowercase Greek letter rho (ρ).

Scanned by CamScanner

b) Syntax

$\rho_{<New_Name>}$ (Input_Table_Name)

c) Example

Query : Find salary and age of all Employees.

Solution : $\pi_{e.age,\ e.salary}$ (ρ_e (Employee))

e.Age	e.Salary
24	50000
24	15000
25	52000
23	41000
34	25000
54	50000
69	45000
74	50000

Syllabus Topic : SET Theory Operations

4.5 SET Operation

Q. Explain various SET operators in relational algebra.

1. Introduction

- SQL SET operators allow combining results from two or more SELECT statements or combines result set of multiple queries.
- 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
```

2. SET Compatibility

- In order to apply SET operations the source tables must possess following Requirements,
- SELECT statement of both queries must retrieve the same number of columns.
- SELECT columns of both queries must be of same Data types.

4.5.1 Union Operator

MU - May 2014

University Question

Q. Explain Set Union Relational algebra operators with suitable examples. **(May 2014, 2 Marks)**

a) Overview

- This operator finds out all combined rows in table 1 and table 2.
- Union effectively appends the result of first query to the result of second query.
- It does not eliminate all duplicate rows and they are printed in result expression.

b) Syntax

(Query Expression 1) ∪ (Query Expression 2)

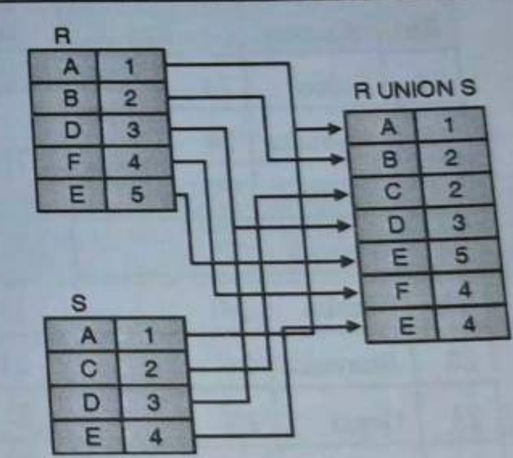


Fig. 4.5.1: SET Operation

c) Example

(i) IT Employee table

Table Name : IT_Employee		
Eid	**Ename**	**Age**
11	Suhas	24
12	Jayendra	24
13	Sachin	25
14	Mahesh	23

Scanned by CamScanner

(ii) Computer department Employee table

Table Name : COMP_Employee		
Eid	**Ename**	**Age**
21	Varsha	24
22	Bhavna	24
23	Geeta	25
24	Amrita	23

Query : Find all Employees in computer and IT departments.

Solution : (IT_Employee) ∪ (COMP_Employee)

Eid	Ename	Age
11	Suhas	24
12	Jayendra	24
13	Sachin	25
14	Mahesh	23
21	Varsha	24
22	Bhavna	24
23	Geeta	25
24	Amruta	23

4.5.2 Intersect Operator

MU - May 2014, May 2015

University Question

Q. Explain Set Intersection operation with suitable examples.

(May 2014, May 2015, 2 Marks)

a) Overview

- This operator finds out all rows that are common in table 1 and table 2.
- If Intersect operator is applied on two queries then it will return all rows that are common in the result of Query 1 and Query 2.

b) Syntax

(Query Expression 1) ∩ (Query Expression 2)

R	
A	1
B	2
D	3
F	4
E	5

R INTERSECTION S	
A	1
D	3

S	
A	1
C	2
D	3
E	4

c) Example

(i) All employees in IT department.

Table Name : IT_Employee		
Eid	**Ename**	**Age**
11	Suhas	24
12	Jayendra	24
13	Sachin	25
14	Mahesh	23

(ii) All employees in Vidya Engineering College.

Table Name : Vidya_Employee		
Eid	**Ename**	**Age**
11	Suhas	24
12	Jayendra	24
23	Geeta	25
24	Amruta	23
35	Sangita	21

Query : Find all Employees in IT department of Vidya Engineering College.

Solution : (IT_Employee) ∪ (Vidya_Employee)

Scanned by CamScanner

Eid	Ename	Age
11	Suhas	24
12	Jayendra	24

4.5.3 Difference Operator

Q. Explain Set Difference Relational algebra operators with suitable examples. **MU - May 2012** **(May 2012, 2 Marks)**

a) Overview

- This operator finds out all rows that are present in table 1 and not in table 2.
- If Intersect operator is applied on two queries then it will return all rows that are present in the result of Query 1 and not in Query 2.

b) Syntax

(Query Expression 1) - (Query Expression 2)

R	
A	1
B	2
D	3
F	4
E	5

R DIFFERENCE S	
B	2
F	4
E	5

S	
A	1
C	2
D	3
E	4

S DIFFERENCE R	
C	2
E	4

c) Example

(i) All faculties in IT department of Vidya Engineering College.

Table Name : Vidya_Employee		
Eid	Ename	Age
11	Suhas	24
12	Jayendra	24
13	Sachin	25
14	Mahesh	23

(ii) All faculties in IT department of all colleges.

Table Name : IT_Employee		
Eid	Ename	Age
11	Suhas	24
12	Jayendra	24
13	Sachin	25
14	Mahesh	23
23	Geeta	25
24	Amruta	23
35	Sangita	21

Query : Find all Employees in IT department but not in Vidya Engineering College.

Solution : (IT_Employee) - (Vidya_Employee)

Eid	Ename	Age
23	Geeta	25
24	Amruta	23
35	Sangita	21

Syllabus Topic : Binary Relational Operation

4.6 Cross Product / Cartesian product

a) Overview

- A cross join performs relational product or Cartesian product of two tables specified in query.
- In this case every row in first table will be joined with every row in second table. So finally number of rows in result table will be equals to product of number of rows in table 1 and number of rows in table 2.
- That means all rows in the first table are joined to all rows in the second table.

b) Syntax

(Query Expression 1) × (Query Expression 2)

Scanned by CamScanner

c) Example

Employee		
Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	100

Department	
Did	Dname
100	HR
200	TIS

Query : Find combination all Employees and departments.

Solution : (Employee) X (Department)

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

Cartesian product

- 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.

To avoid a Cartesian product, always include a valid join condition in a WHERE clause.

4.7 Join Operation ($\bowtie_\theta$)

University Question

Q. Express Join in terms of basic relational algebra operations.

a) Overview

- Join operator helps us to retrieve data from multiple tables or relations.

- Most common type of join is **Natural join** ($\bowtie$) in which column having same name in two table will be taken for joining tables.

b) Syntax

(< table_name>) $\bowtie_{<\text{join_condition}>}$ (table_name)

c) There are various types of joins possible in relational algebra.

1. Type 1 : Natural joins
2. Type 2 : Inner joins
3. Type 3 : Outer joins

Type 1 : Natural join ($\bowtie$)

MU - May 2012, Dec. 2013, May 2014, May 2015

University Question

Q. Explain Natural Join Relational algebra operators with suitable examples. **(May 2012, Dec. 2013, May 2014, May 2015, 2 Marks)**

- Natural join can join tables based on the common columns in the tables being joined.
- 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.
  - **Prerequisites for Natural Join**

    Both table must have at list one common column with same column name and same data type.

Steps of Working

- The two table are joined using Cross join
- DBMS will look for a common column with same name and data type
- Tuples having exactly same values in common columns are kept in result.

Example

Employee		
Eid	Ename	Did
1	Amit	10
2	Nitin	30
3	Yogesh	50

Department	
Did	Dname
10	IT
30	HR
40	TIS

Query : Find all Employees and their respective departments.

Solution : (Employee) $\bowtie$ (Department)

Scanned by CamScanner

Eid	Ename	Did	Did	Dname
1	Amit	10	10	IT
2	Nitin	30	30	HR
Employee Data			Department Data	

- In above example the employee data having same did as department data are kept in result set.
- All non-matching tuples of cross join are ignored.

Type 2 : Inner joins / Theta Join ($\bowtie_\theta$)

Q. Explain Inner Join Relational algebra operators with suitable examples.

- Theta join will combines tuples from multiple relations if they satisfy the specified join condition.
- This join condition is also called as Theta and denoted by the symbol θ.
- The tables are joined according to join conditions.
- The only rows with matching values are combined using inner join.
- Inner join will ignore all tuple does not find matching tuple in other table.

Example

Query : Find all Employees and their respective departments.

Solution : Employee $\bowtie_{\text{employee.did= Department.did}}$ Department

Eid	Ename	Did	Did	Dname
1	Amit	10	10	IT
2	Nitin	30	30	HR
Employee Data			Department Data	

- In above example the employee data having did exactly same as department data are kept in result set.
- All non-matching tuples of cross join are ignored.

Type 3 : Outer joins

University Question MU - Dec. 2013

Q. Explain Outer Join Relational algebra operators with suitable examples.

(Dec. 2013, 2 Marks)

- 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.
- Outer joins are useful when you are trying to determine which values in related tables cause referential integrity problem.
- Such problems are created when foreign key values do not match the primary key values in related table.

(I) Left outer join

- Table on left side of operator may contain null values.
- Left outer join takes all tuples in the left relation that did not match with any tuple in the right relation.

Example

Query : Find all Employees and their respective department data.

Solution : Employee $\bowtie=_{\text{employee.did= Department.did}}$ Department

Eid	Ename	Did	Did	Dname
1	Amit	10	10	IT
2	Nitin	30	30	HR
3	Yogesh	50	Null	Null
Employee Data			Department Data	

- In above example the employee data having did exactly same as department data a kept in result set.
- All left side non-matching tuples of cross join are also considered.

(II) Right outer join

- Table on right side of operator may contain null values.
- Right outer join takes all tuples in the right relation that did not match with any in the left relation.

Example

Query : Find all departments with employee data.

Solution : Employee $=\bowtie_{\text{employee.did= Department.did}}$ Department

Scanned by CamScanner

Eid	Ename	Did	Did	Dname
1	Amit	10	10	IT
2	Nitin	30	30	HR
Null	Null	Null	40	TIS
Employee Data			Department Data	

- In above example the employee data having did exactly same as department data are kept in result set.
- All right side non-matching tuples of cross join are also considered.

(ii) Full outer join

- Any table on both sides of operator may contain null values.

Example

Query : Find all Employees and departments.

Solution : Employee $=\bowtie=_{\text{employee.did= Department.did}}$ Department

Eid	Ename	Did	Did	Dname
1	Amit	10	10	IT
2	Nitin	30	30	HR
3	Yogesh	50	Null	Null
Null	Null	Null	40	TIS
Employee Data			Department Data	

- In above example the employee data having did exactly same as department data are kept in result set.
- All right side non-matching tuples of cross join are also considered.

.8 Relational Division Operator

University Question MU - May 2012, Dec. 2013, May 2015

. Explain Division Relational algebra operators with suitable examples. **(May 2012, Dec. 2013, May 2015, 2 Marks)**

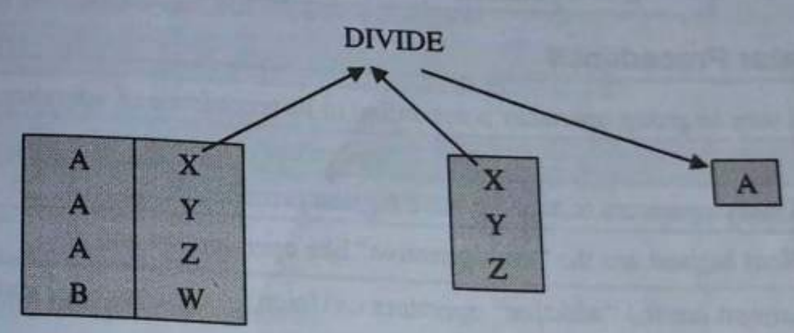
Overview

- The divide operator operates on two tables that must have common columns between them.

- The relational divide operator (so called to distinguish it from mathematical division) returns the records in one record set that have values that match all the corresponding values in the second record set.
- The output of divide operation is one column in which values of common column in both tables match.

b) Syntax

(Query Expression 1) ÷ (Query Expression 2)



DIVIDE Operator

c) Example

Student Table			
Stud_ID	Sname	Course_ID	Gender
1	Mahesh	100	M
2	Manish	100	M
3	Amruta	100	F
3	Amruta	200	F
6	Neesha	100	F
3	Amruta	300	F
6	Neesha	300	F
6	Neesha	200	F

Course Table contains all courses that trainer 401 is taking.

Course_ID	Trainer_ID
100	401
200	401
300	401

Scanned by CamScanner

Find female students take ALL the courses that 401 are taking.

Student ÷ Course

OR

$\pi_{s.Stud_ID,\ s.Sname,\ s.Gender,\ s.Course_ID}\ (\rho_s(Student) \div \rho_c(Course))$

s.Stud_ID	s.Sname	s.Gender	s.Course_ID
3	Amruta	F	200
6	Neesha	F	100

4.9 Operator Precedence

The normal way to group operators is according to its precedence of operation.

Precedence	Operators
1.	Unary operators σ, π and ρ have highest precedence.
2.	Next highest are the "multiplicative" like operators ⋈ and X.
3.	Lowest are the "additive" operators ∪(Union), ∩(Intersection) and − (minus) operators.

But there is no universal agreement, so we always put parentheses around the argument of a unary operator and it is a good idea to group all binary operators with parentheses enclosing their arguments.

Syllabus Topic : Relational Algebra Queries

4.10 Relational Algebra Queries - Solved Examples

Example 4.10.1

Consider the following relations for database that keeps track of student enrollment in courses and books issued for each course.

STUDENT (Ssn, Name, Subject, DOB)
COURSE (Course_id, Name, Dept)
ENROLL (Ssn, Course_id, Semester, Grade)
Book_issued (Course_id, Semester, ISBN)
TEXT (ISBN, Title, Publisher, Author)
Write any 5 Queries in relational algebra.

Solution :

(1) Write a query to select all courses available in institute.

$\Pi_{course_id,\ CName,\ Dept}$ (COURSE)

(2) Find all student details registered for course id 10.

$\Pi_{Ssn,\ Name}\ (\sigma_{course_id\ =\ 10}(ENROLL \bowtie STUDENT))$

(3) Find various book titles and authors for semester higher than 3.

$\Pi_{ISBN,\ Title,\ Author}\ (\sigma_{semester\ >\ 3}(Book_Issed \bowtie TEXT))$

(4) Find all students belongs to IT Depatment (without join)

a) To find course_id of 'IT' Department

$T_1 \leftarrow \Pi_{course_id,}\ (\sigma_{Dept\ =\ 'IT'}\ (COURSE))$

b) To find all students enrolled for above course id.

$T_2 \leftarrow \Pi_{ssn\ ,}\ (ENROLL \bowtie T_1)$

c) To find student details having above Ssn.

$Ans \leftarrow \Pi_{Ssn,\ Name,\ DOB}\ (STUDENT \bowtie T_2)$

Example 4.10.2

Consider the relations given below :

Dealer (Dealer-no, DealerName, address)
Part (Part-no, Part-name, color)
Assigned-to (Dealer-no, Part-no, cost)

Give an expression in relational algebra the following queries :

(i) Find the name of all dealers who supply 'Red' Parts.
(ii) Find the name of the dealers who supply both Yellow and Green Parts.
(iii) Find the name of the dealers who supply all the Parts.
(iv) List all dealer names.

Solution :

(i) The name of all dealers who supply 'Red' Parts.

$\Pi_{Dealername}\ (\sigma_{course='red'}\ (Dealer \bowtie Part))$

(ii) The name of the dealers who supply both Yellow and Green Parts

$\Pi_{Dealername}\ (\sigma_{course='red'\ \&\ course='yellow'}\ (Dealer \bowtie Part))$

(iii) The name of the dealers who supply all the Parts.

$\Pi_{Dealername}\ (Dealer \bowtie Part))$

(iv) The list of all dealer names

$\Pi_{Dealername}\ (Dealer)$

Scanned by CamScanner

Review Questions

Q. 1 Explain various operators in relational algebra.

Q. 2 Write a short note :

(a) Selection operation (b) Projection Operator

Q. 3 Explain concept of division operation.

Q. 4 Explain concept of JOIN operation in relational algebra.

Q. 5 Explain various SET operators in relational algebra.

Q. 6 Explain concept of product operation in relational algebra.

Q. 7 Explain following Relational algebra operators with suitable examples :

(i) Select (ii) Project (iii) Join

(iv) Division (v) Cartesian product

Q. 8 Explain following relational algebra operators with example :

(i) Set-difference (ii) Outer join

Q. 9 Explain following relational algebra operators with example :

(i) Project (iii) Division

(iv) Natural Join (v) Set Difference.

Q. 10 Explain following relational algebra with suitable example : Natural Join.

Q. 11 Express Natural Join and Division operator in terms of basic relational algebra operations.

Q. 12 Given the following relations :

Vehicle (reg_no, make, colour)

Person (eno, name, address)

Owner (eno, reg_no)

Write expressions in relational algebra to answer the following queries :

(i) List the names of persons who do not own any car.

(ii) List the names of persons who own only Maruti Cars.

Q. 13 Consider the relations given below :

Dealer (Dealer-no, DealerName, address)

Part (Part-no, Part-name, color)

Assigned-to (Dealer-no, Part-no, cost)

Give an expression in relational algebra the following queries :

(i) Find the name of all dealers who supply 'Red' Parts.

(ii) Find the name of the dealers who supply both Yellow and Green Parts.

(iii) Find the name of the dealers who supply all the Parts.

4.11 University Questions and Answers

May 2012

Q. 1 Explain following relational algebra operators with example :

(i) Project (ii) Rename (iii) Division

(iv) Natural Join (v) Set Difference.

(Ans. : Refer sections 4.3, 4.4, 4.5.3, 4.7 (Type I), 4.8) **(10 Marks)**

Dec. 2013

Q. 2 Explain following relational algebra operations with proper examples.

(i) Project (ii) Left outer join, (iii) Division

(iv) Rename (v) Natural join

(Ans. : Refer sections 4.3, 4.4, 4.7 (Type I, 4.7 (Type III), 4.8) **(10 Marks)**

May 2014

Q. 3 Explain the following relational algebra operations with proper examples.

(i) Natural join (ii) Rename

(iii) Set Intersection operation (iv) Union

(Ans. : Refer sections 4.4, 4.5.1, 4.5.2, 4.7 (Type I) **(10 Marks)**

Dec. 2014

Q. 4 Explain any four relational algebra operations with proper examples.

(Ans. : Refer section 4.1) **(10 Marks)**

May 2015

Q. 5 Explain following relational algebra operations with proper examples :

(i) Set Intersection operation (ii) Natural join

(iii) Division

(Ans. : Refer sections 4.5.2, 4.7(Type I,4.8)) **(3 Marks)**

Scanned by CamScanner

Dec.2015

Q.6 Describe the following relational algebra operations :

(i) Project (ii) Natural join

(iii) Set Intersection

(Ans. : Refer sections 4.3, 4.7(Type I), 4.5.2)

Q.7 Describe the following relational algebra operation : Select **(7 Marks)**

(Ans.: Refer section 4.2)

Dec. 2016

Q.5(a) Explain any four relational algebra operations with proper examples. **(3 Marks)**

(Ans. : Refer section 4.1)

May 2017

Q.4(b) Discuss fundamental operations in relational algebra. **(10 Marks)**

(Ans. : Refer section 4.1)

(10 Marks)

❑❑❑

Module IV

CHAPTER 5

Structured Query Language

Course Outcome

Create and populate a RDBMS for a real life application

Syllabus Covered

- Overview of SQL
- Referential Integrity Constraint in SQL
- Data Definition Commands
- Data Manipulation commands
- Data Control commands

Syllabus Topic : Overview of SQL

5.1 Overview of SQL

Q. Explain role SQL with example.

a) **SQL (Structured Query Language)** is a computer language aimed to store, manipulate, and retrieve data stored in relational databases.

b) It was developed by IBM Research in the mid 70's and standardized by ANSI in 1998.

c) The first commercial relational database was released by Relational Software (later called as Oracle).

d) SQL is a keyword based language and each statement begins with a unique keyword.

e) SQL syntax is not case sensitive.

Scanned by CamScanner

4.11 University Questions and ...

Module 4

Chapter 5 : Relational Database Languages ..... **5-1 to 5-26**

✓ **Syllabus Topic :** Overview of SQL 5-1

5.1 Overview of SQL 5-1

5.1.1 Role of SQL 5-2

5.2 SQL Data Types 5-2

✓ **Syllabus Topic :** Data Definition Commands (DDL) 5-5

5.3 Data Definition Language (DDL) 5-5

Scanned by CamScanner

5.4 CREATE Statement / CREATE Table 5-5
5.5 Create Table with Constraints 5-7
5.5.1 Domain Integrity Constraint 5-7
5.5.2 Entity Integrity Constraint 5-8
✓ **Syllabus Topic :** Referential Integrity Constraint in SQL 5-9
5.5.3 Referential Integrity Constraint in SQL 5-9
5.6 Alter Table 5-9
5.7 Rename Table 5-11
5.8 Truncate Table 5-11
5.9 Drop Command / DROP Table 5-11
✓ **Syllabus Topic :** Data Manipulation Commands (DML) 5-12
5.10 Data Manipulation Language (DML) 5-12
5.10.1 INSERT Statement 5-13
5.10.2 DELETE Statement 5-14
5.10.3 UPDATE Statement 5-14
✓ **Syllabus Topic :** Data Control Commands (DCC) 5-15
5.11 Data Control Language (DCL) 5-15
5.12 Privileges 5-16
5.13 Granting Privileges 5-17
5.14 Revoking of Privileges 5-19
5.15 Solved Designing Problem 5-21
5.16 University Questions and Answers 5-26

Chapter 6 : Display Data using SQL **6-1 to 6-39**

6.1 SELECT Statement 6-1
6.2 Ordering Query Results 6-4
6.3 Displaying Selected Rows / Selection Operation in SQL 6-7
6.4 Limit Clause – Restricting Rows in Result 6-12
✓ **Syllabus Topic :** Aggregate Functions 6-13
6.5 Aggregate Functions 6-13
6.5.1 Summary of Aggregate Functions 6-18

Scanned by CamScanner

✓ **Syllabus Topic : Complex Retrieval Queries using Group By** 6-19

6.6 GROUP BY Clause - Grouping Query Results 6-19

6.7 HAVING Clause – Apply Condition on groups 6-21

6.8 GROUP BY and Aggregate Function 6-23

6.9 Pattern Matching - String Operations 6-24

6.10 SET Operations 6-29

6.10.1 Union Operator 6-29

6.10.2 Intersect Operator 6-31

6.10.3 Difference Operator 6-32

6.11 NULL Values 6-34

6.12 Solved Designing Problem 6-38

6.13 University Questions and Answers 6-39

Chapter 7 : Complex Queries **7-1 to 7-40**

Scanned by CamScanner

CHAPTER 5

Module IV

Structured Query Language

Course Outcome

Create and populate a RDBMS for a real life application

Syllabus Covered

- Overview of SQL
- Referential Integrity Constraint in SQL
- Data Definition Commands
- Data Manipulation commands
- Data Control commands

Syllabus Topic : Overview of SQL

5.1 Overview of SQL

Q. Explain role SQL with example.

a) **SQL (Structured Query Language)** is a computer language aimed to store, manipulate, and retrieve data stored in relational databases.

b) It was developed by IBM Research in the mid 70's and standardized by ANSI in 1998.

c) The first commercial relational database was released by Relational Software (later called as Oracle).

d) SQL is a keyword based language and each statement begins with a unique keyword.

e) SQL syntax is not case sensitive.

Scanned by CamScanner

- As per size of number we can use following types of integers.
  - (i) INTEGER (p)
  - (ii) INTEGER or INT
  - (iii) SMALLINT
  - (iv) BIGINT

(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

(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)

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)

Scanned by CamScanner

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 [.*s*F] 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 / date time

- The TIMESTAMP data type includes both date and time fields, plus a minimum of six positions and for decimal fractions of second and optimal with TIMEZONE Qualifier.
- Represented using the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND in the format YYYY-MM-DD HH:MM:SS[.*s*F] 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 time zone

(d) Interval

- 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.

- Generally not supported by SQL server but supported by Oracle.

Syllabus Topic : Data Definition Commands (DDL)

5.3 Data Definition Language (DDL)

University Question **MU - May 2017**

Q. Explain DDL commands with example. **(May 2017, 10 Marks)**

- To create database schema and database objects like table **Data Definition Language (DDL)** can be used.
- DDL statements are used to build and modify the structure of your tables and other objects in the database.
- The set of DDL commands are as below,
  1. CREATE Statement : To create Database objects
  2. ALTER Statement : To modify structure of database objects
  3. DROP Statement : To remove database objects
  4. RENAME Statement : To Rename Database objects
  5. TRUNCATE Statement : To empty the database table
- 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.
- **E.g.** Table, View, Sequence etc.

5.4 CREATE Statement / CREATE Table

Q. Explain CREATE command with example.

- CREATE statement is used to create new database objects like table, index and others.
- CREATE TABLE is the command in database system is used to create a new table with unique name or identifier.
- This statement used to create database object.

Scanned by CamScanner

Syntax

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

Example

```
SQL> CREATE TABLE Employee
    (   Eid  INT,
   Name     VARCHAR (20),
   Age  INT,
   Address   CHAR (25),
   Salary    DECIMAL (18, 2)
    );
Query OK, 0 rows affected (0.01 sec)
```

- To view the structure of newly created table.

```
SQL> DESC Employee;
+---------+----------------+------+-----+---------+-------+
| Field   | Type           | Null | Key | Default | Extra |
+---------+----------------+------+-----+---------+-------+
| EID     | int(10)        | YES  |     | NULL    |       |
| NAME    | varchar(20)    | YES  |     | NULL    |       |
| AGE     | int(11)        | YES  |     | NULL    |       |
| ADDRESS | char(25)       | YES  |     | NULL    |       |
| SALARY  | decimal(18,2)  | YES  |     | NULL    |       |
+---------+----------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```


5.5 Create Table with Constraints

5.5.1 Domain Integrity Constraint

Domain constraints are used to test the values inserted into the table is correct or not.

a) Required Data Constraint / Not Null Constraint

- 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.

```
SQL> CREATE TABLE Student
    ( Eid INT NOT NULL);
Query OK, 0 rows affected (0.01 sec)
```

b) Check Constraint

- 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'.

```
SQL> CREATE TABLE customer
   (Name    CHAR (25) NOT NULL,
    Gender  CHAR (1),
        CHECK (Gender IN ('M', 'F'))
   );
Query OK, 0 rows affected (0.01 sec)
```

c) Default Keyword

- 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'.

```
SQL> Create table customer
    (    Name    char (25) DEFAULT 'UNKNOWN');
Query OK, 0 rows affected (0.01 sec)
```

Scanned by CamScanner

5.5.2 Entity Integrity Constraint

Entity constraints are used to test the values inserted into the database are correct or not with respect to other tuples in same table.

a) Primary Key Constraint

- Primary key attribute is same as unique key constraint with Not NULL constraints.
- Primary key attribute values needs to be unique as well as null values are not allowed in primary key attributes.
- 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.
- **Example :** Table with customer entity having name, cid and gender in which cid is primary key.

```
SQL> CREATE TABLE customer
   (Name   CHAR (25),
   Cid CHAR (10) PRIMARY KEY
    );
Query OK, 0 rows affected (0.01 sec)
```

b) Unique Constraint

- 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.

Example

```
SQL> CREATE TABLE customer
   (Name   CHAR (25),
   Cid CHAR (10),
   Email   CHAR (50) UNIQUE
    );
Query OK, 0 rows affected (0.01 sec)
```


Syllabus Topic : Referential Integrity Constraint in SQL

5.5.3 Referential Integrity Constraint in SQL

- A value appearing in a 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 defined between two tables to maintain the consistency among tuples in the two relations.

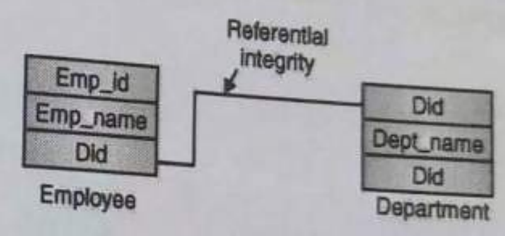


Fig. 5.5.1

Example

```
SQL> Create table Department
   (Did INT,
    Dept_name VARCHAR (100) NOT NULL,
    PRIMARY KEY (did)
   );
Query OK, 0 rows affected (0.01 sec)
SQL> Create table Emp
   (     Emp_id INT,
    Emp_name VARCHAR (100) NOT NULL,
    Did       INT REFERENCES department (did)
   );
Query OK, 0 rows affected (0.01 sec)
```

5.6 Alter Table

Q. Explain ALTER command with example.

Scanned by CamScanner

- Once database object is created in database, we may require to the ALTER command is used to update structure of database object.
- The ALTER TABLE statement can be used to add, delete, or modify columns in an existing table.
- The ALTER TABLE command can also be used to add and drop various constraints on an existing table.

Syntax

```
ALTER TABLE <Table_Name>
 ADD Column_1 datatype;

ALTER TABLE <Table_Name>
 Modify Column_1 New_datatype;

ALTER TABLE <Table_Name>
 DROP Column_1;
```

Example

```
SQL> ALTER TABLE Employee
   ADD Address VARCHAR (100);
Query OK, 0 rows affected (0.01 sec)
```

To view the changed structure of table

```
SQL> DESCRIBE TABLE Employee;
+---------+---------------+------+-----+---------+-------+
| Field   | Type          | Null | Key | Default | Extra |
+---------+---------------+------+-----+---------+-------+
| EID     | int(10)       | YES  |     | NULL    |       |
| NAME    | varchar(20)   | YES  |     | NULL    |       |
| AGE     | int(11)       | YES  |     | NULL    |       |
| ADDRESS | varchar(100)  | YES  |     | NULL    |       |
| SALARY  | decimal(18,2) | YES  |     | NULL    |       |
+---------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```


5.7 Rename Table

Q. Explain RENAME command with example.

- It is possible to change name of table with or without data in it using simple RENAME command.
- We can rename any table object at any point of time.

Syntax

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

Example

```
SQL> RENAME TABLE Employee To EMP;
```

5.8 Truncate Table

Q. Explain TRUNCATE command with example.

- The SQL TRUNCATE TABLE command is used to delete all data from an existing table.
- It is possible to da same action with DROP TABLE command but it would remove complete table structure form the database.
- A DELETE command will also remove all data from table but with DELETE data deletion can be rolled back and truncate acts as permanent data deletion with no roll back possible.
- If any delete triggers are defined on the table, then the triggers are not fired on truncate table.
- Truncate will de-allocates memory space. So that the free space can be used by other tables unlike DELETE command.

Syntax

```
TRUNCATE TABLE <Table_Name>;
```

Example

```
SQL> TRUNCATE TABLE EMP;
```

5.9 Drop Command / DROP Table

Q. Explain DROP command with example.

- Drop command can be used to remove database any objects from user database.
- The SQL DROP TABLE statement is used to remove a table definition and all related data like indexes, triggers, constraints and permission specifications for that table.

Scanned by CamScanner

- The developer must be careful while running this command because once a table is dropped then all the information available in that table will also be lost forever and no roll back can be done.

Syntax

```
DROP TABLE <Table_Name>;
```

Example

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

```
SQL> DROP TABLE Employee;
Query OK, 0 rows affected (0.01 sec)
```

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

```
SQL> DESC TABLE Employee;
ERROR 1146 (42S02): Table 'TEST.Employee' doesn't exist
```

Syllabus Topic : Data Manipulation Commands (DML)

5.10 Data Manipulation Language (DML)

MU - May 2017

University Question

Q. Explain DML commands with syntax. **(May 2017, 10 Marks)**

- **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. Or in other words the DML statements do not implicitly commit the current transaction.
- DML is set of commands used to,
  - Insert data into table
  - Delete data from table
  - Update data of table

5.10.1 INSERT Statement

- Insert statement used to add records to the existing table.
- To insert data into a table, SQL INSERT INTO command can be used.
- To insert few values in table as per columns names we can use generic syntax as below,

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

- If all values for all the columns of the table are to be added then also no need to specify the column names in the SQL query.
- But, we need to make sure the order of the values is in the same order as the columns in the table.

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

Example

```
SQL> INSERT INTO Employee VALUES (1001, 'Mahesh');
SQL> INSERT INTO Employee VALUES (NULL, 'Jayendra');

SQL> INSERT INTO Employee (Name, Eid) VALUES ('Sachin' , 1002);
SQL> INSERT INTO Employee (Name, Eid) VALUES ('Suhas' , NULL);
```

The data added in table can be displayed as follows,

```
SQL> SELECT * FROM Employee;
+----+----------+
|EID | NAME     |
+----+----------+
|1001| Ramesh   |
|1002| Khilan   |
|NULL| Kaushik  |
|NULL| Chaitali |
+----+----------+
```

Scanned by CamScanner

5.10.2 DELETE Statement

- Delete statement is used to delete some or all records from the existing table.
- To delete data into a table, SQL DELETE command can be used.
- To delete all rows in table we can use generic syntax as below,

Syntax

```
DELETE
FROM <Table_Name>;
```

To delete selected rows from table we can specify the WHERE condition.

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

Example

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

To check rows deleted by above delete query is as given.

```
SELECT   *
FROM     Employee;
SQL> SELECT * FROM Employee;
+----+----------+
|EID | NAME     |
+----+----------+
|1001| Ramesh |
|1002| Khilan   |
+----+----------+
```

5.10.3 UPDATE Statement

- The UPDATE statement is used to modify the existing data present in a table.
- To update data in a table, SQL UPDATE command can be used.

- To update all rows in table we can use generic syntax as below,

```
UPDATE <Table_Name>
SET    column1 = new_value;
```

- To update selected rows from table we can specify the WHERE condition in Update statement.

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

- **Example**

```
SQL>  UPDATE   Employee
      SET      Eid=1002
      WHERE    name = 'Suhas';
```

To check rows updated in table using select query.

```
SQL>  SELECT *
      FROM   Employee;
```

```
SQL> SELECT * FROM Employee;
+----+----------+
|EID | NAME     |
+----+----------+
|1001| Ramesh |
|1002| Suhas    |
+----+----------+
```

Syllabus Topic : Data Control Commands (DCC)

5.11 Data Control Language (DCL)

Q. Write a note on DCL.

- **Data Control Language (DCL)** is used to control various user actions (or privileges) in Database.

Scanned by CamScanner

- To perform any operation in the database user needs **privileges** like creating tables, sequences 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 from user.
- Privileges can be of many types,
  - **System Privileges** : creating a table is types of system privilege.
  - **Object Privileges** : To execute query on tables object privilege can be used.
  - **Ownership Privileges** : To execute query on tables created by same user.

5.12 Privileges

Q. Explain all types of privileges.

Q. Enlist the 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 below,

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 has different privileges assigned for him,
- Few object privileges are as below,

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.

5.13 Granting Privileges

Q. Write syntax for GRANT privileges.

Scanned by CamScanner

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]
```

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';
```

(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';
```

5.14 Revoking of Privileges

Q. Write a short note on Revoking of privileges.

Scanned by CamScanner

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 behavior.


```
REVOKE SELECT
ON Emp_Salary
FROM 'mahesh'@'somehost'
RESTRICT;
```

5.15 Solved Designing Problem

Example 5.15.1

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)

Solution :

(i) Modify the database so that 'Jack' now lives in 'Newyork'.

```
MySQL> UPDATE Employee
   SET City = 'Newyork'
   WHERE Name = 'Jack';
```

(ii) Give all employees of 'ANZ corporation' a 10% raise in salary.

```
MySQL> UPDATE Works
   SET Salary = (salary+(0.1*salary))
   WHERE CID IN ( SELECT Cid
                  FROM Company
                  WHERE Company_name = 'ANZ corporation');
```

Example 5.15.2

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)

Scanned by CamScanner

Solution :

(i) Update the damage amount for car with licence number "Mum2022" in the accident with report number "AR2197" to Rs. 5000.

```
MySQL> Update  Participated
    SET  Damage_amount = 500
    WHERE   Report_number LIKE 'AR2197' AND Car = 'Mum2022';
```

Example 5.15.3 :

For given database, write SQL queries.

Employee (EID, Name, Street, City)

Works (EID, CID, Salary)

Manager (EID, Manager_Name)

Company (CID, Company_name, City)

Solution :

(i) Modify the database so that 'TRATHAM' now lives in USA

```
MySQL>  UPDATE Employee
    SET  City = 'USA'
    WHERE   Name = 'TRATHAM';
```

(ii) Give all employees of 'SHARAYU Steel' a 10% raise in salary.

```
MySQL>  UPDATE works
    SET  salary = (salary + (0.1 * salary))
    WHERE   cid IN ( SELECT   cid
        FROM   company
        WHERE  company_name = 'SHARAYU Steel');
```

Example 5.15.4 **MU - Dec. 2014, 10 Marks**

Consider insurance database given below and answer the following queries in SQL.

Person (driver_id, name, address)

Car (license, model, year)

Accident (report_no, adate, location)

Owns (driver_id, license)

Participated (driver_id, license, report_no, damage_amount)

Solution :

(i) Add new accident to database.

```
MySQL> INSERT INTO Accident (report_no, adate, location)
    VALUES ('111', '01/01/2014','Pune');
```

(ii) Delete `Santro' belonging to 'John Smith'.

```
MySQL>  DELETE
    FROM CAR
    WHERE    Model = 'SANTRO'
     AND
     License IN ( SELECT license
            FROM   Owns
            WHERE  Driver_id IN ( SELECT driver_id
                    FROM person
                   WHERE name = 'John Smith')
            );
```

Example 5.15.5 **MU - Dec 2013, 5 Marks**

Consider the following employee database.

Employee (empname, street, city, date_of_joining)

Works (empname, company _name, salary)

Company (company_name, city)

Manages (empname, manager_name).

Write SQL queries for the following statements :

(i) Modify the database so that 'John' now lives in 'Mumbai'.

(ii) Give all employees of ABC Corporation' a 10% raise

Solution :

(i) Modify the database so that 'John' now lives in 'Mumbai'.

```
MySQL>  UPDATE Employee
    SET  City= 'Mumbai'
    WHERE   Empname= 'JOHN';
```

Scanned by CamScanner

(ii) Give all employees of ABC Corporation' a 10% raise.

```
MySQL> UPDATE Works
    SET  Salary=0.1*salary
    WHERE  Company_name= 'ABC Corporation';
```

Example 5.15.6 **MU - May 2017. 10 Marks**

Employees (Empid, Fname, Lname, Email, Phoneno, Hiredate, Jobid, Salary, Mid, Did)

Departments (Did, Dname, Managerid, Locationid)

Locations (Locationid, Streetadd, Postalcode, City)

Write the SQL queries for the following.

1. List the employees have a manager who works for a department based in the U.S.
2. Write a query to display the details of all employees in the Finance department.
3. Give 10% hike to all the employees working in Did 20.
4. Write a query to display all the information of the employees whose salary is within the range 1000 and 3000.
5. Display the information of all the employees whose first name starts with 'R' in descending order of their salary.

Solution :

1. List the employees have a manager who works for a department based in the U.S.

```
SELECT *
FROM Employees e
        INNER JOIN Departments d ON e.did = d.did
        INNER JOIN Locations l ON l.locationid=d.locationid
WHERE City ='US';
```

2. Write a query to display the details of all employees in the Finance department.

```
SELECT *
FROM Employees e
        INNER JOIN Departments d ON e.did = d.did
WHERE Dname='Finance';
```

3. Give 10% hike to all the employees working in Did 20.

```
UPDATE employees
SET Salary = 1.1* Salary
WHERE did =20;
```


4. Write a query to display all the information of the employees whose salary is within the range 1000 and 3000.

```
SELECT *
FROM employees
WHERE Salary BETWEEN 1000 AND 3000;
```

5. Display the information of all the employees whose first name starts with R' in descending order of their salary.

```
SELECT *
FROM employees
WHERE Fname LIKE 'R%'
ORDER BY Salary DESC;
```

Review Questions

Q. 1 Explain various data definition statements in SQL.

Q. 2 Explain various data types used in SQL.

Q. 3 How to create table employee (Eid, Ename, age, DOB, address).

Q. 4 Explain Describe table command with example.

Q. 5 Explain various data definition statements in SQL.

Q. 6 Explain various data types used in SQL.

Q. 7 What is SQL ? Explain the following structures of SQL queries with appropriate example

(i) Select clause

(ii) Where clause

(iii) From clause

Q. 8 Write short note on : DDL and DML.

Q. 9 Write syntax for GRANT privileges.

Q. 10 Write a short note on Revoking of privileges.

Scanned by CamScanner

5.16 University Questions and Answers

Dec. 2013

Q. 1 Consider the following employee database.
Employee (empname, street, city, date_of_joining)
Works (empname, company _name, salary)
Company (company_name, city)
Manages (empname, manager_name). Write SQL queries for the following statement
(Ans.: Refer Example 5.15.5) **(5 Marks)**

Dec. 2014

Q. 2 Considers insurance database given below and answer the following queries in SQL.
Person (driver_id, name, address)
Car (license, model, year)
Accident (report_no, adate, location)
Owns (driver_id, license)
Participated (driver_id, license, report_no, damage_amount)
(Ans.: Refer Example 5.15.4) **(10 Marks)**

May 2017

Q.3(b) Explain DML and DDL commands with syntax. *(Ans.: Refer section 5.10 and 5.3)* **(10 Marks)**

Q. 5(a) Employees (Empid, Fname, Lname, Email, Phoneno, Hiredate, Jobid, Salary, Mid, Did) **(10 Marks)**
Departments (Did, Dname, Managerid, Locationid)
Locations (Locationid, Streetadd, Postalcode, City)
Write the SQL queries for the following.

1. List the employees have a manager who works for a department based in the U.S.
2. Write a query to display the details of all employees in the Finance department.
3. Give 10% hike to all the employees working in Did 20.
4. Write a query to display all the information of the employees whose salary is within the range 1000 and 3000.
5. Display the information of all the employees whose first name starts with 'R' in descending order of their salary. *(Ans.: Refer Example 5.15.6)*

□□□

CHAPTER 6

Module IV

Display Data using SQL

Course Outcome

Retrieve any type of information from a data base using complex queries in SQL.

Syllabus

- Aggregate Function
- Complex Retrieval Queries using Group By
- Set operations
- Null Values

6.1 SELECT Statement

Q. Explain use of SELECT in SQL. Give its syntax.
Q. What is SQL ? Explain the SELECT and FROM clause of SQL queries with appropriate example.

- The SELECT statement in SQL is used to select data and display from a table in database.
- The SELECT can also include SQL built in function to manipulate the data in database just for display purpose like aggregate functions.
- The SELECT can be used for displaying various attributes or columns of a table.

```
SELECT  [DISTINCT] [ * /Column1,Column2...]
FROM    <Table_Name>;
```

1. Display all data in table

- 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.

Scanned by CamScanner

Syntax

```
SQL>SELECT *
    FROM    <Table_Name>;
```

Example

Select all details of employees in company.

```
MYSQL> SELECT *
    FROM    Employee;

+----+---------+----+---------+---------+
| ID | NAME    | AGE | ADDRESS | SALARY |
+----+---------+----+---------+---------+
| 1 | Yogesh  | 32 | Mumbai  | 12000.00 |
| 2 | Karan   | 25 | Nagar   | 2500.00 |
| 3 | Akshay  | 43 | Delhi   | 4000.00 |
| 4 | Chetan  | 25 | Mumbai  | 6800.00 |
| 5 | Hardik  | 37 | Mumbai  | 8300.00 |
| 6 | Kamal   | 32 | Pune    | 4100.00 |
| 7 | Mahesh  | 24 | Pune    | 15000.00 |
+----+---------+----+---------+---------+
7 rows in set (0.00 sec)
```

2. Display limited column data in table

- 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  Column1, Column2
FROM    <Table_Name>;
```

Example

Select Id and names of employees in company.

```
MYSQL> SELECT ID, NAME
    FROM    Employee;

+----+---------+
| ID | NAME    |
+----+---------+
| 1 | Yogesh  |
```



```
| 2 | Karan   |
| 3 | Akshay  |
| 4 | Chetan  |
| 5 | Hardik  |
| 6 | Kamal   |
| 7 | Mahesh  |
+----+---------+
7 rows in set (0.01 sec)
```

3. Display unique records in table

- 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 DISTINCT 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  DISTINCT [*/Column1,Column2...]
FROM    <Table_Name>;
```

Example

Select different locations of above employees.

```
MYSQL> SELECT DISTINCT ADDRESS
    FROM    Employee;

+---------+
| ADDRESS |
+---------+
| Mumbai  |
| Nagar   |
| Delhi   |
| Pune    |
+---------+
4 rows in set (0.04 sec)
```

Scanned by CamScanner

- The DISTINCT keyword can be used with the selection of all columns. In this case, it will print all tuples by ignoring repeated tuples in the relation. If no values repeated in a row then DISTINCT will have no effect on results.
- Select all details of employees in company (Avoid repeated data).

```
MYSQL> SELECT DISTINCT *
       FROM   Employee;

+----+---------+-----+---------+---------+
| ID | NAME    | AGE | ADDRESS | SALARY  |
+----+---------+-----+---------+---------+
| 1 | Yogesh  | 32 | Mumbai   | 12000.00 |
| 2 | Karan   | 25 | Nagar    | 2500.00 |
| 3 | Akshay  | 43 | Delhi    | 4000.00 |
| 4 | Chetan  | 25 | Mumbai   | 6800.00 |
| 5 | Hardik  | 37 | Mumbai   | 8300.00 |
| 6 | Kamal   | 32 | Pune     | 4100.00 |
| 7 | Mahesh  | 24 | Pune     | 15000.00 |
+----+---------+-----+---------+---------+
7 rows in set (0.00 sec)
```

6.2 Ordering Query Results

Q. Explain 'ORDER BY' clause.

(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

Select details of employees in company in alphabetical order of names.

```
MYSQL> SELECT*
       FROM      Employee
       ORDER BY  NAME;
```



```
+----+---------+-----+---------+---------+
| ID | NAME    | AGE | ADDRESS | SALARY  |
+----+---------+-----+---------+---------+
| 3 | Akshay  | 43 | Delhi    | 4000.00 |
| 4 | Chetan  | 25 | Mumbai   | 6800.00 |
| 5 | Hardik  | 37 | Mumbai   | 8300.00 |
| 6 | Kamal   | 32 | Pune     | 4100.00 |
| 2 | Karan   | 25 | Nagar    | 2500.00 |
| 7 | Mahesh  | 24 | Pune     | 15000.00 |
| 1 | Yogesh  | 32 | Mumbai   | 12000.00 |
+----+---------+-----+---------+---------+
7 rows in set (0.01 sec)
```

- Now we want to select all employees from the table above, however, we want to sort the employees by reverse order of their names.
- In second query ORDER BY 2 indicates order by second column of table.

```
MYSQL> SELECT *
       FROM      Employee
       ORDER BY  NAME DESC;
OR
MYSQL> SELECT *
       FROM      Employee
       ORDER BY  2 DESC;

+-----+---------+-----+---------+---------+
| ID  | Name    | Age | Address | Salary  |
+-----+---------+-----+---------+---------+
|  1 | Yogesh  |  32 | Mumbai  | 12000.00 |
|  7 | Mahesh  |  24 | Pune    | 15000.00 |
|  2 | Karan   |  25 | Nagar   | 2500.00 |
|  6 | Kamal   |  32 | Pune    | 4100.00 |
|  5 | Hardik  |  37 | Mumbai  | 8300.00 |
|  4 | Chetan  |  25 | Mumbai  | 6800.00 |
|  3 | Akshay  |  43 | Delhi   | 4000.00 |
+-----+---------+-----+---------+---------+
7 rows in set (0.00 sec)
```

- The ordering can be done on numeric column like salary by default ascending OR descending ordering.

Scanned by CamScanner

```
MYSQL> SELECT *
    FROM    Employee
    ORDER BY    SALARY DESC;
OR
MYSQL> SELECT *
    FROM    Employee
    ORDER BY    5 DESC;
+----+---------+-----+---------+----------+
| ID | Name    | Age | Address | Salary   |
+----+---------+-----+---------+----------+
|  7 | Mahesh  |  24 | Pune    | 15000.00 |
|  1 | Yogesh  |  32 | Mumbai  | 12000.00 |
|  5 | Hardik  |  37 | Mumbai  |  8300.00 |
|  4 | Chetan  |  25 | Mumbai  |  6800.00 |
|  6 | Kamal   |  32 | Pune    |  4100.00 |
|  3 | Akshay  |  43 | Delhi   |  4000.00 |
|  2 | Karan   |  25 | Nagar   |  2500.00 |
+----+---------+-----+---------+----------+
7 rows in set (0.00 sec)
```

- The ordering can be done on multiple columns also by ascending or descending ordering. In such case ordering is first done by first column on list if this column has same value then only second column is considered for further ordering.

```
MYSQL> SELECT *
    FROM    Employee
    ORDER BY    ADDRESS ASC, SALARY DESC;

+----+---------+-----+---------+----------+
| ID | NAME    | AGE | ADDRESS | SALARY   |
+----+---------+-----+---------+----------+
| 3 | Akshay  | 43 | Delhi   | 4000.00 |
| 1 | Yogesh  | 32 | Mumbai  | 12000.00 |
| 5 | Hardik  | 37 | Mumbai  | 8300.00 |
| 4 | Chetan  | 25 | Mumbai  | 6800.00 |
| 2 | Karan   | 25 | Nagar   | 2500.00 |
| 7 | Mahesh  | 24 | Pune    | 15000.00 |
| 6 | Kamal   | 32 | Pune    | 4100.00 |
+----+---------+-----+---------+----------+
7 rows in set (0.01 sec)
```


- If values in first column contains all unique values then second column will not be considered.

```
MYSQL> SELECT *
    FROM    Employee
    ORDER BY    NAME DESC, SALARY DESC;

+----+---------+-----+---------+----------+
| ID | Name    | Age | Address | Salary   |
+----+---------+-----+---------+----------+
|  1 | Yogesh  |  32 | Mumbai  | 12000.00 |
|  7 | Mahesh  |  24 | Pune    | 15000.00 |
|  2 | Karan   |  25 | Nagar   |  2500.00 |
|  6 | Kamal   |  32 | Pune    |  4100.00 |
|  5 | Hardik  |  37 | Mumbai  |  8300.00 |
|  4 | Chetan  |  25 | Mumbai  |  6800.00 |
|  3 | Akshay  |  43 | Delhi   |  4000.00 |
+----+---------+-----+---------+----------+
7 rows in set (0.00 sec)
```

6.3 Displaying Selected Rows / Selection Operation in SQL

Q. What is SQL ? Explain the Where clause of SQL queries with appropriate example.

a) Introduction

- The SQL can print selected rows by applying selection operation in SQL with help of 'WHERE' Clause.
- WHERE statement is an optional statement.
- The condition written in WHERE clause is a row qualifier condition that row must satisfy to print that row in final result of query.
- The query without WHERE condition will print all rows by default.

b) Syntax

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

c) Examples

Select details of employees in company having salary above 5000.

```
MYSQL> SELECT *
    FROM    Employee
```

Scanned by CamScanner

```
    WHERE   SALARY > 5000
    Order By ID;
+----+----------+-----+----------+----------+
| ID | NAME     | AGE | ADDRESS  | SALARY   |
+----+----------+-----+----------+----------+
| 1 | Yogesh   | 32 | Mumbai   | 12000.00 |
| 4 | Chetan   | 25 | Mumbai   | 6800.00 |
| 5 | Hardik   | 37 | Mumbai   | 8300.00 |
| 7 | Mahesh   | 24 | Pune     | 15000.00 |
+----+----------+-----+----------+----------+
4 rows in set (0.01 sec)
```

d) Basic query execution

(i) In above given query the first executable statement is **FROM**, this statement will load the all rows in EMPLOYEE table from DBMS server to local memory of PC executing above query.

(ii) **WHERE** clause acts like a **row filter** using condition (Salary>5000); it will only select the rows of EMPLOYEE table having Salary over 5000 and other remaining employee tuples are deleted from memory.

(iii) **ORDER BY** statement will arrange output in proper order of ID column in ascending order by default.

(iv) At last the **SELECT** statement either prints all or specific columns onto client computer.

Fig. 6.3.1 : SQL Query Execution Hierarchy

Points to remember

- 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.

e) Arithmetic Comparison (=, <, <=, >, >=, <>)

- The comparison of attribute value with some constant values can be done with help of arithmetic comparisons.
- Find all employees not belongs to Mumbai.

```
MYSQL> SELECT *
    FROM    Employee
    WHERE   ADDRESS <> 'MUMBAI'

+----+----------+-----+----------+----------+
| ID | NAME     | AGE | ADDRESS  | SALARY   |
+----+----------+-----+----------+----------+
| 2 | Karan    | 25 | Nagar    | 2500.00 |
| 3 | Akshay   | 43 | Delhi    | 4000.00 |
| 6 | Kamal    | 32 | Pune     | 4100.00 |
| 7 | Mahesh   | 24 | Pune     | 15000.00 |
+----+----------+-----+----------+----------+
4 rows in set (0.01 sec)
```

f) BETWEEN Clause

- 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.
- Find all employees with salary between 2000 and 10000.

```
MYSQL> SELECT *
    FROM    Employee
    WHERE   SALARY BETWEEN 2000 AND 10000;

+------+--------+-----+--------+--------+
| ID   | Name   | Age | Address | Salary |
+------+--------+-----+--------+--------+
|  2 | Karan |  25 | Nagar  | 2500.00 |
|  3 | Akshay |  43 | Delhi  | 4000.00 |
|  4 | Chetan |  25 | Mumbai | 6800.00 |
|  5 | Hardik |  37 | Mumbai | 8300.00 |
|  6 | Kamal  |  32 | Pune   | 4100.00 |
+------+--------+-----+--------+--------+
5 rows in set (0.03 sec)
```

Scanned by CamScanner

g) IN Clause

- IN clause will acts like set comparison operator.
- It checks attribute value for equality with at least one value from given set of comparison values specified in IN clause.
- Find all employees located at Mumbai or Pune.

```
MYSQL> SELECT *
    FROM    Employee
    WHERE   ADDRESS IN ('Mumbai', 'Pune');

+------+--------+-----+--------+----------+
| ID   | Name   | Age | Address | Salary   |
+------+--------+-----+--------+----------+
|    1 | Yogesh |  32 | Mumbai | 12000.00 |
|    4 | Chetan |  25 | Mumbai |  6800.00 |
|    5 | Hardik |  37 | Mumbai |  8300.00 |
|    6 | Kamal  |  32 | Pune   |  4100.00 |
|    7 | Mahesh |  24 | Pune   | 15000.00 |
+------+--------+-----+--------+----------+
5 rows in set (0.00 sec)
```

h) Applying multiple conditions using Logical operators

- The Logical operators are those operators which returns true or false.
- They return a true or false values to combine one or more true or false values returned by evaluation of more than one expression.

AND Operator

- Logical AND operator compares between multiple Booleans values returned by expressions and returns true when all expressions returns to true.
- Example
- Find all employees located at Mumbai or Pune and salary above 2000.

```
MYSQL> SELECT *
    FROM    Employee
    WHERE   ADDRESS IN ('Mumbai', 'Pune')
     AND SALARY > 2000;
+----+---------+----+---------+----------+
| ID | NAME    | AGE | ADDRESS | SALARY   |
+----+---------+----+---------+----------+
| 1 | Yogesh  | 32 | Mumbai  | 12000.00 |
```



```
| 4 | Chetan  | 25 | Mumbai  |  6800.00 |
| 5 | Hardik  | 37 | Mumbai  |  8300.00 |
| 6 | Kamal   | 32 | Pune    |  4100.00 |
| 7 | Mahesh  | 24 | Pune    | 15000.00 |
+----+---------+----+---------+----------+
5 rows in set (0.01 sec)
```

OR Operator

- Logical AND operator compares between multiple Booleans values returned by expressions and returns true when at least one expression returns to true.
- Example
- Find all employees located at Mumbai or Pune or salary above 2000.

```
MYSQL> SELECT *
    FROM    Employee
    WHERE   ADDRESS IN ('Mumbai', 'Pune')
     OR SALARY > 2000;
+----+---------+----+---------+----------+
| ID | NAME    | AGE | ADDRESS | SALARY   |
+----+---------+----+---------+----------+
| 1 | Yogesh  | 32 | Mumbai  | 12000.00 |
| 2 | Karan   | 25 | Nagar   |  2500.00 |
| 3 | Akshay  | 43 | Delhi   |  4000.00 |
| 4 | Chetan  | 25 | Mumbai  |  6800.00 |
| 5 | Hardik  | 37 | Mumbai  |  8300.00 |
| 6 | Kamal   | 32 | Pune    |  4100.00 |
| 7 | Mahesh  | 24 | Pune    | 15000.00 |
+----+---------+----+---------+----------+
7 rows in set (0.01 sec)
```

NOT Operator

- Not logical operator takes a single Boolean value of expression and changes its from false to true or from true to false
- Example
- Find all employees located at Mumbai or Pune.

```
MYSQL> SELECT *
    FROM    Employee
    WHERE   ADDRESS NOT IN ('Mumbai', 'Pune');
```

Scanned by CamScanner

```
+----+----------+----+----------+----------+
| ID | NAME     | AGE | ADDRESS  | SALARY   |
+----+----------+----+----------+----------+
| 2 | Karan     | 25 | Nagar    | 2500.00 |
| 3 | Akshay    | 43 | Delhi    | 4000.00 |
+----+----------+----+----------+----------+
```

- Find all employees not between range of 2000 and 10000.

```
MYSQL> SELECT *
    FROM    Employee
    WHERE   SALARY NOT BETWEEN 2000 AND 10000;

+----+----------+----+----------+----------+
| ID | NAME     | AGE | ADDRESS  | SALARY   |
+----+----------+----+----------+----------+
| 1 | Yogesh    | 32 | Mumbai   | 12000.00 |
| 7 | Mahesh    | 24 | Pune     | 15000.00 |
+----+----------+----+----------+----------+
```

6.4 Limit Clause – Restricting Rows in Result

Q. Explain Limit clause.

a) Introduction

- 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.

b) Syntax

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

c) Example

Select top 3 salaried employees from company.

```
MYSQL> SELECT *
    FROM    Employee
    Order by SALARY DESC
    LIMIT       3;
```



```
+----+----------+----+----------+----------+
| ID | NAME     | AGE | ADDRESS  | SALARY   |
+----+----------+----+----------+----------+
| 7 | Mahesh    | 24 | Pune     | 15000.00 |
| 1 | Yogesh    | 32 | Mumbai   | 12000.00 |
| 5 | Hardik    | 37 | Mumbai   | 8300.00 |
+----+----------+----+----------+----------+
```

Select 2nd and 3rd largest salary in company.

```
MYSQL> SELECT *
    FROM    Employee
    Order by SALARY DESC
    LIMIT       2, 2;

+----+----------+----+----------+----------+
| ID | NAME     | AGE | ADDRESS  | SALARY   |
+----+----------+----+----------+----------+
| 1 | Yogesh    | 32 | Mumbai   | 12000.00 |
| 5 | Hardik    | 37 | Mumbai   | 8300.00 |
+----+----------+----+----------+----------+
```

Syllabus Topic : Aggregate Functions

6.5 Aggregate Functions

MU - May 2016, Dec. 2016

University Question

Q. Explain Aggregate function with example. (May 2016, Dec. 2016, 5 Marks)

1. Introduction

- The management report require lot of decision making data to get summarized information from table like average, sum, minimum.
- 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.
- The aggregate data functions ignore all null value in column on which aggregate data function if applied.

Scanned by CamScanner

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

Consider the student table indicate assessment marks of every student. The blanks in mark indicates absent student in exam. Assessment is conducted out of 100 marks. If school management want to analyse performance of class using summarized data.

ID	Name	Marks	Department	Subject	Grant
1	Mahesh	90	IT	DB	5000
2	Suhas	60	CS	DB	
3	Jayendra	89	IT	DB	3000
4	Sachin	99	CS	DB	7000
5	Vishal	78	IT	DB	
6	Payal	90	MECH	CPP	3000
7	Kasturi		MECH	CPP	

a) COUNT ()

- This function is used to calculate number of rows (or records) in a table selected by query.
- Column in the query must be numeric.
- COUNT returns the number of rows in the table when the column value is not NULL.
- Count (*) will returns that number of unique tuples present in table.

Find number of students in class.

```
MYSQL> SELECT count (*) Tot_Employees
   FROM      Student;
OR
MYSQL> SELECT count (ID) Tot_Employees
   FROM      Student;

+-------------+
| Tot_Employees |
+-------------+
|          7 |
+-------------+
1 row in set (0.00 sec)
```


- The aggregate data functions ignore all null value in column on which aggregate data function if applied.

Find number of students appeared for exam.

```
MYSQL> SELECT count (Marks)
   FROM      Student;

+-------------+
| count(Marks) |
+-------------+
|          6 |
+-------------+
1 row in set (0.00 sec)
```

- The COUNT aggregate data function can be used with DISTINCT to ignore all repeated value in column on which aggregate data function if applied.

Find number of departments of participated in exam.

```
MYSQL> SELECT count (DISTINCT Department)
   FROM      Student;

+-------------------------+
| count(DISTINCT Department) |
+-------------------------+
|                       3 |
+-------------------------+
1 row in set (0.00 sec)
```

- The COUNT aggregate data function can be used with WHERE to select few rows in table on which aggregate data function if applied.

Find number of departments of participated in exam with at least one student scored above 60.

```
MYSQL> SELECT count(DISTINCT Department)
   FROM      Student
   WHERE        Marks>60;

+-------------------------+
| count(DISTINCT Department) |
+-------------------------+
|                       2 |
+-------------------------+
1 row in set (0.00 sec)
```

Scanned by CamScanner

(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.

Find total of grants given by non IT departments.

```
MYSQL> SELECT SUM(Grants)
    FROM      Student
    WHERE     Department <> 'IT';

+-------------+
| SUM(Grants) |
+-------------+
|   10000.00 |
+-------------+
1 row in set (0.00 sec)
```

Find total of grants given by non IT departments to students scoring above 90.

```
MYSQL> SELECT SUM(Grants)
    FROM      Student
    WHERE     Department <> 'IT'
        AND Marks > 90;
+-------------+
| SUM(Grants) |
+-------------+
|    7000.00 |
+-------------+
1 row in set (0.00 sec)
```

(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.

Find average marks scored by non IT departments.

```
MYSQL> SELECT AVG(Marks)
    FROM      Student
    WHERE     Department <> 'IT';
```



```
+------------+
| AVG(Marks) |
+------------+
|   83.0000 |
+------------+
1 row in set (0.00 sec)
```

Find average marks scored by non IT students in CPP subject.

```
MYSQL> SELECT AVG(Marks)
    FROM      Student
    WHERE     Department <> 'IT'
        AND Subject = 'CPP';
+------------+
| AVG(Marks) |
+------------+
|   90.0000 |
+------------+
1 row in set (0.01 sec)
```

(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.

Find minimum marks scored by non IT departments.

```
MYSQL> SELECT MIN(Marks)
    FROM      Student
    WHERE     Department <> 'IT';
+------------+
| MIN(Marks) |
+------------+
|       60 |
+------------+
1 row in set (0.00 sec)
```

Find minimum marks scored by non IT students in CPP subject.

```
MYSQL> SELECT MIN(Marks)
    FROM      Student
    WHERE     Department <> 'IT'
        AND Subject = 'CPP';
```

Scanned by CamScanner

```
+----------+
| MIN(Marks) |
+----------+
|       90 |
+----------+
1 row in set (0.00 sec)
```

(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.

Find maximum marks scored by non IT departments.

```
MYSQL> SELECT MAX(Marks)
    FROM      Student
    WHERE     Department <> 'IT';
+----------+
| MAX(Marks) |
+----------+
|       99 |
+----------+
1 row in set (0.00 sec)
```

Find maximum marks scored by non IT students in CPP subject.

```
MYSQL> SELECT MAX(Marks)
    FROM      Student
    WHERE     Department <> 'IT'
        AND Subject = 'CPP';
+----------+
| MAX(Marks) |
+----------+
|       90 |
+----------+
1 row in set (0.00 sec)
```

6.5.1 Summary of Aggregate Functions

Q. Compare various Aggregate functions.

Function	Description
AVG ([DISTINCT | ALL] n)	Average value of n, ignoring null values.

Function	Description
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.

Syllabus Topic : Complex Retrieval Queries using Group By

6.6 GROUP BY Clause - Grouping Query Results

MU - May 2016

University Question

Q. Explain the term : Group by clause. **(May 2016, 2 Marks)**

(1) Introduction

- The SQL GROUP BY clause is used in with the SQL SELECT statement to arrange data into groups.
- Related rows can be grouped together by **GROUP BY** clause based on distinct values that exist for specified grouping columns.
- The GROUP BY statement is used with the SQL aggregate functions to group the data using one or more columns or expression.
- 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
[WHERE    Condition]
GROUP BY  Column_names
[ORDER BY Columns];
```

(3) Example

- Consider the student table indicate assessment marks of every student. The blanks in mark indicates absent student in exam. Assessment is conducted out of 100 marks.
- If school management want to analyse performance of class using summarized data.

Scanned by CamScanner

ID	Name	Marks	Department	Subject	Grant
1	Mahesh	90	IT	DB	5000
2	Suhas	60	CS	DB	
3	Jayendra	89	IT	DB	3000
4	Sachin	99	CS	DB	7000
5	Vishal	78	IT	DB	
6	Payal	90	MECH	CPP	3000
7	Kasturi		MECH	CPP	

Retrieve all departments in college.

```
MySQL> SELECT Department
    FROM        Student
    GROUP BY        Department;

+-----------------+
| Department      |
+-----------------+
| CS              |
| IT              |
| MECH            |
+-----------------+
3 rows in set (0.00 sec)
```

Retrieve all departments offers DB subject.

```
MySQL> SELECT Department
    FROM        Student
    WHERE       subject= 'DB'
    GROUP BY        Department;

+-----------------+
| Department      |
+-----------------+
| CS              |
| IT              |
+-----------------+
2 rows in set (0.00 sec)
```


Retrieve all different departments offers DB subject. (Sort in reverse alphabetical order)

```
MySQL> SELECT Department
    FROM        Student
    WHERE       subject= 'DB'
    GROUP BY        Department
    ORDER BY        Department;

+-----------------+
| Department      |
+-----------------+
| IT              |
| CS              |
+-----------------+
2 rows in set (0.00 sec)
```

6.7 HAVING Clause – Apply Condition on groups

(1) WHERE clause

- A Grouping Query can also group rows based on common values in a grouping columns.
- To select few rows with specified condition before grouping WHERE clause can be specified. If WHERE clause is not used in query then all rows in table are retrieved by default.
- WHERE clause defines a row qualifier condition. It means a condition row must satisfy to be a part of result set.

Retrieve all departments offers DB subject.

```
MySQL> SELECT Department
    FROM        Student
    WHERE       subject= 'DB'
    GROUP BY        Department;

+-----------------+
| Department      |
+-----------------+
| CS              |
| IT              |
+-----------------+
2 rows in set (0.00 sec)
```

Scanned by CamScanner

(2) HAVING clause

- A Grouping Query will group table rows based on common values in a grouping columns after that if we need to select few groups meeting certain conditions HAVING clause can be used.
- To select few groups with specified condition after grouping HAVING clause can be specified. If it is not used in query then all groups will be printed by default.
- 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.

Retrieve departments offered total grants above 5000 for DB Subject.

```
MySQL> SELECT Department
   FROM    Student
   WHERE   subject= 'DB'
   GROUP BY   Department
   HAVING  SUM(Grants) > 5000;

+-----------------+
| Department      |
+-----------------+
| CS              |
+-----------------+
1 rows in set (0.00 sec)
```

- Difference in operation of WHERE and GROUP BY clause
- 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.
- Only the groups that meet the HAVING conditions appear in the query output.
- It is possible to apply a HAVING clause only to columns
- vthat also appear in the GROUP BY clause or in z aggregate function. Retrieve departments with name start from letter 'C' and offers DB subject.

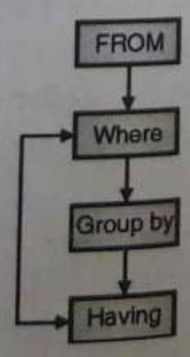


Fig.6.7.1

```
MySQL> SELECT Department
   FROM    Student
   WHERE   subject= 'DB'
```



```
   GROUP BY    Department
   HAVING  Department LIKE 'C%';
+-----------------+
| Department      |
+-----------------+
| CS              |
+-----------------+
1 rows in set (0.00 sec)
```

(3) Single row query Evaluation

(i) In above given query the first executable statement is **FROM**, this statement will load the all rows in STUDENT table on to server to local memory.

(ii) **WHERE** clause will select student rows with subject 'DB' and other remaining rows are deleted from memory.

(iii) **GROUP BY clause** will form groups of entire table using Department column then we will get number of groups equals to total number of departments.

(iv) **HAVING clause** will select few department groups with help of condition Department name start with letter C and remaining groups are deleted from memory.

(v) **ORDER BY** Clause will arrange output in proper order that can be either ascending or descending order.

(vi) At last **SELECT** Clause will either all (*) or specified columns will be printed on client computer.

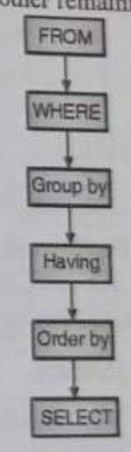


Fig. 6.7.2

6.8 GROUP BY and Aggregate Function

(1) Introduction

- **GROUP BY** clauses is very useful in combination with aggregate functions for generating management summary report which is more helpful for taking decisions.
- Generally, aggregate functions will return a single result for a table.
- **GROUP BY** statement can return single function value for each individual group of column values.

(2) Example

Retrieve maximum grant amount offered by any department.

```
MySQL> SELECT Max(Grants)
   FROM    Student;
+-----------------+
| Max(Grant)      |
```

Scanned by CamScanner

```
+-----------------+
| 7000            |
+-----------------+
1 rows in set (0.00 sec)
```

Retrieve maximum grant amount offered by each department.

```
MySQL> SELECT Department, Max(Grants)
    FROM    Student
    GROUP BY    Department;

+-------------+-------------+
| Department | Max(Grants) |
+-------------+-------------+
| CS         |   7000.00 |
| IT         |   5000.00 |
| MECH       |   3000.00 |
+-------------+-------------+
3 rows in set (0.00 sec)
```

6.9 Pattern Matching - String Operations

Q. Where to use LIKE clause ? Give its syntax.

(1) Introduction

- 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.

(2) Syntax

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

{PATTERN} often consists of wildcards.

(3) Example

- Consider the student table indicate assessment marks of every student. The blanks in mark indicates absent student in exam. Assessment is conducted out of 100 marks. If school management want to analyse performance of class using summarized data.

ID	Name	Marks	Department	Subject	Grant
1	Mahesh	90	IT	DB	5000
2	Suhas	60	CS	DB	
3	Jayendra	89	IT	DB	3000
4	Sachin	99	CS	DB	7000
5	Vishal	78	IT	DB	
6	Payal	90	MECH	CPP	3000
7	Kasturi		MECH	CPP	

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

Find all students whose name starts with 'S'.

```
MySQL> SELECT *
    FROM    Student
    WHERE   Name LIKE 'S%';
+-----+--------+-------+------------+---------+---------+
| ID  | Name   | Marks | Department | Subject | Grants  |
+-----+--------+-------+------------+---------+---------+
|  2  | Suhas  |  60   | CS         | DB      |  NULL   |
|  4  | Sachin |  99   | CS         | DB      | 7000.00 |
+-----+--------+-------+------------+---------+---------+
2 rows in set (0.00 sec)
```

Pattern 2 : Determine a string starts with a given string.

Find all students whose name starts with 'Om'.

```
Mysql> SELECT    *
    FROM    Student
    WHERE   Name LIKE 'Pay%';
+-----+-------+-------+------------+---------+---------+
| ID  | Name  | Marks | Department | Subject | Grants  |
+-----+-------+-------+------------+---------+---------+
|  6  | Payal |  90   | MECH       | CPP     | 3000.00 |
+-----+-------+-------+------------+---------+---------+
1 row in set (0.00 sec)
```

Scanned by CamScanner

Pattern 3 : Determine a string ends with a given letter.

Find all students whose name ends with 'h'.

```
Mysql> SELECT *
       FROM    Student
       WHERE   Name LIKE '%h';
+------+--------+------+------------+---------+---------+
| ID   | Name   | Marks | Department | Subject | Grants  |
+------+--------+------+------------+---------+---------+
|    1 | Mahesh |   90 | IT         | DB      | 5000.00 |
+------+--------+------+------------+---------+---------+
1 row in set (0.00 sec)
```

Pattern 4 : Determine a string contains given a letter.

Find all students whose name contains letter 'e'.

```
MySQL> SELECT *
       FROM    Student
       WHERE   Name LIKE '%e%';
+------+----------+------+------------+---------+---------+
| ID   | Name     | Marks | Department | Subject | Grants  |
+------+----------+------+------------+---------+---------+
|    1 | Mahesh   |   90 | IT         | DB      | 5000.00 |
|    3 | Jayendra |   89 | IT         | DB      | 3000.00 |
+------+----------+------+------------+---------+---------+
2 rows in set (0.00 sec)
```

Pattern 5 : Determine a string in which given pattern is combination of above forms.

Find all students whose names first letter is 'm' contains letter 'e' and ending with letter 'h'.

```
MySQL> SELECT *
       FROMStudent
       WHERE   Name LIKE 'M%e%h';
+------+--------+------+------------+---------+---------+
| ID   | Name   | Marks | Department | Subject | Grants  |
+------+--------+------+------------+---------+---------+
|    1 | Mahesh |   90 | IT         | DB      | 5000.00 |
+------+--------+------+------------+---------+---------+
1 row in set (0.00 sec)
```


Pattern 6 : Determine a String in which given a letter is present at particular position.

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

```
MySQL> SELECT *
       FROMStudent
       WHERE   Name LIKE '_a%';
+------+----------+------+------------+---------+---------+
| ID   | Name     | Marks | Department | Subject | Grants  |
+------+----------+------+------------+---------+---------+
|    1 | Mahesh   |   90 | IT         | DB      | 5000.00 |
|    3 | Jayendra |   89 | IT         | DB      | 3000.00 |
|    4 | Sachin   |   99 | CS         | DB      | 7000.00 |
|    6 | Payal    |   90 | MECH       | CPP     | 3000.00 |
|    7 | Kasturi  | NULL | MECH       | CPP     |    NULL |
+------+----------+------+------------+---------+---------+
5 rows in set (0.00 sec)
```

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

```
MySQL> SELECT *
       FROMStudent
       WHERE   Name LIKE '%s_';
+------+--------+------+------------+---------+---------+
| ID   | Name   | Marks | Department | Subject | Grants  |
+------+--------+------+------------+---------+---------+
|    1 | Mahesh |   90 | IT         | DB      | 5000.00 |
+------+--------+------+------------+---------+---------+
1 row in set (0.00 sec)
```

Pattern 7 : Determine a string not contains given a letter.

Find all students whose name do not contains 's'.

```
MySQL> SELECT *
       FROMStudent
       WHERE   Name NOT LIKE '%s%';
+------+----------+------+------------+---------+---------+
| ID   | Name     | Marks | Department | Subject | Grants  |
+------+----------+------+------------+---------+---------+
|    3 | Jayendra |   89 | IT         | DB      | 3000.00 |
|    6 | Payal    |   90 | MECH       | CPP     | 3000.00 |
+------+----------+------+------------+---------+---------+
2 rows in set (0.00 sec)
```

Scanned by CamScanner

(4) 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

Consider student table as given below,

```
MySQL> SELECT * FROM student;
+----+----------+-----+----------+-------+---------+----------------+
| ID | Name     | Marks | Department | Subject | Grants | email          |
+----+----------+-----+----------+-------+---------+----------------+
|  1 | Mahesh   |  90 | IT       | DB    | 5000.00 | a_bh@myc.edu   |
|  2 | Suhas    |  60 | CS       | DB    |    NULL | uh@myc.edu     |
|  3 | Jayendra |  89 | IT       | DB    | 3000.00 | a_by@myc.edu   |
|  4 | Sachin   |  99 | CS       | DB    | 7000.00 | a_bc@myc.edu   |
|  5 | Vishal   |  78 | IT       | DB    |    NULL | is@myc.edu     |
|  6 | Payal    |  90 | MECH     | CPP   | 3000.00 | a_by@myc.edu   |
|  7 | Kasturi  | NULL | MECH     | CPP   |    NULL | a_bs@myc.edu   |
+----+----------+-----+----------+-------+---------+----------------+
7 rows in set (0.00 sec)
```

Find all students whose email id does not contains underscore.

```
MySQL> SELECT *
       FROMStudent
       WHERE  Email NOT LIKE '%_%';

Empty set (0.00 sec)
```

The above query is not working as if finds all students with email ids with length not more than one letter.

Find all students whose email id does not contains underscore.

```
MySQL> SELECT *
       FROMStudent
       WHERE  Email NOT LIKE '%@_%' ESCAPE '@';
```



```
+----+----------+-----+----------+-------+---------+----------------+
| ID | Name     | Marks | Department | Subject | Grants | email          |
+----+----------+-----+----------+-------+---------+----------------+
|  2 | Suhas    |  60 | CS       | DB    |    NULL | uh@myc.edu     |
|  5 | Vishal   |  78 | IT       | DB    |    NULL | is@myc.edu     |
+----+----------+-----+----------+-------+---------+----------------+
2 rows in set (0.00 sec)
```

6.10 SET Operations

Q. What do you mean by set operation ?

1. Introduction

- SQL SET operators allow combining results from two or more SELECT statements or combines result set of multiple queries.
- 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
```

2. Requirements

- SELECT statement of both queries must retrieve the same number of columns.
- Data types of corresponding columns in each query must be of same type.

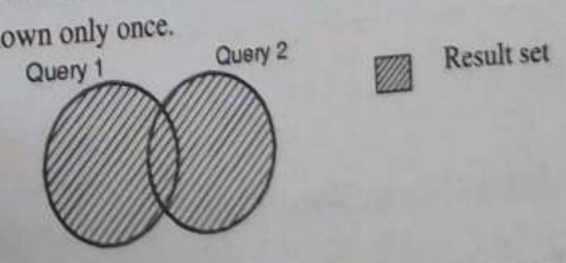
6.10.1 Union Operator

a) Overview

- Union effectively appends the result of first query to the result of second query.
- It does not eliminate all duplicate rows and they are printed in result expression.

DISTINCT : Duplicate row shown only once.

ALL : Duplicate row shown only once.



Find all students in all departments.

```
MySQL> SELECT *
       FROM   student
       WHERE  department='IT';
```

Scanned by CamScanner

```
+------+---------+-------+------------+---------+--------+--------------+
| ID   | Name    | Marks | Department | Subject | Grants | email        |
+------+---------+-------+------------+---------+--------+--------------+
|    1 | Mahesh  |    90 | IT         | DB      | 5000.00 | a_bh@myc.edu |
|    3 | Jayendra |   89 | IT         | DB      | 3000.00 | a_by@myc.edu |
|    5 | Vishal  |    78 | IT         | DB      |   NULL | is@myc.edu   |
+------+---------+-------+------------+---------+--------+--------------+
3 rows in set (0.00 sec)

MySQL> SELECT *
    FROM    student
    WHERE   department='CS';
+------+---------+-------+------------+---------+--------+--------------+
| ID   | Name    | Marks | Department | Subject | Grants | email        |
+------+---------+-------+------------+---------+--------+--------------+
|    2 | Suhas   |    60 | CS         | DB      |   NULL | uh@myc.edu   |
|    4 | Sachin  |    99 | CS         | DB      | 7000.00 | a_bc@myc.edu |
+------+---------+-------+------------+---------+--------+--------------+
2 rows in set (0.00 sec)

MySQL> SELECT *
    FROM student
    WHERE department='MECH';
+------+---------+-------+------------+---------+--------+--------------+
| ID   | Name    | Marks | Department | Subject | Grants | email        |
+------+---------+-------+------------+---------+--------+--------------+
|    6 | Payal   |    90 | MECH       | CPP     | 3000.00 | a_by@myc.edu |
|    7 | Kasturi | NULL  | MECH       | CPP     |   NULL | a_bs@myc.edu |
+------+---------+-------+------------+---------+--------+--------------+
2 rows in set (0.00 sec)


/* To Combine Result of above 3 Queries using UNION Operation */

MySQL> SELECT * FROM student WHERE department='IT'
    UNION
    SELECT * FROM student WHERE department='CS'
    UNION
```



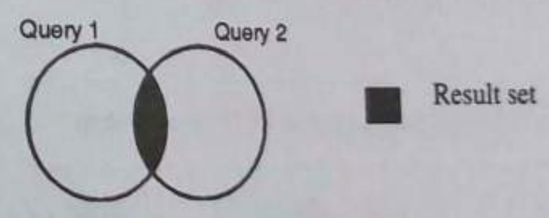
```
    SELECT * FROM student WHERE department='MECH';
+------+---------+-------+------------+---------+--------+--------------+
| ID   | Name    | Marks | Department | Subject | Grants | email        |
+------+---------+-------+------------+---------+--------+--------------+
|    1 | Mahesh  |    90 | IT         | DB      | 5000.00 | a_bh@myc.edu |
|    3 | Jayendra |   89 | IT         | DB      | 3000.00 | a_by@myc.edu |
|    5 | Vishal  |    78 | IT         | DB      |   NULL | is@myc.edu   |
|    2 | Suhas   |    60 | CS         | DB      |   NULL | uh@myc.edu   |
|    4 | Sachin  |    99 | CS         | DB      | 7000.00 | a_bc@myc.edu |
|    6 | Payal   |    90 | MECH       | CPP     | 3000.00 | a_by@myc.edu |
|    7 | Kasturi | NULL  | MECH       | CPP     |   NULL | a_bs@myc.edu |
+------+---------+-------+------------+---------+--------+--------------+
7 rows in set (0.06 sec)
```

6.10.2 Intersect Operator

a) Overview

- This operator finds out all rows that are common in both result of Query 1 and in the result of Query 2.
- It does not eliminate all duplicate rows and they are printed in result expression.

Example



Find all IT students offered grants above 2000.

```
/* Query 1: To print all IT students */
MySQL> SELECT *
    FROM    Student
    WHERE   Department='IT';
+------+---------+-------+------------+---------+--------+--------------+
| ID   | Name    | Marks | Department | Subject | Grants | email        |
+------+---------+-------+------------+---------+--------+--------------+
|    1 | Mahesh  |    90 | IT         | DB      | 5000.00 | a_bh@myc.edu |
|    3 | Jayendra |   89 | IT         | DB      | 3000.00 | a_by@myc.edu |
|    5 | Vishal  |    78 | IT         | DB      |   NULL | is@myc.edu   |
+------+---------+-------+------------+---------+--------+--------------+
```

Scanned by CamScanner

```
3 rows in set (0.00 sec)
/* Query 2: To print all students offered grants above 2000 */

MySQL> SELECT *
       FROM    Student
       WHERE   Grants>=2000;
+-----+----------+-------+-----------+-------+--------+--------------+
| ID  | Name     | Marks | Department | Subject | Grants | email        |
+-----+----------+-------+-----------+-------+--------+--------------+
|   1 | Mahesh   |    90 | IT        | DB    | 5000.00 | a_bh@myc.edu |
|   3 | Jayendra |    89 | IT        | DB    | 3000.00 | a_by@myc.edu |
|   4 | Sachin   |    99 | CS        | DB    | 7000.00 | a_bc@myc.edu |
|   6 | Payal    |    90 | MECH      | CPP   | 3000.00 | a_by@myc.edu |
+-----+----------+-------+-----------+-------+--------+--------------+
4 rows in set (0.00 sec)
```

/* Final Query: To print all IT students offered grants above 2000 */

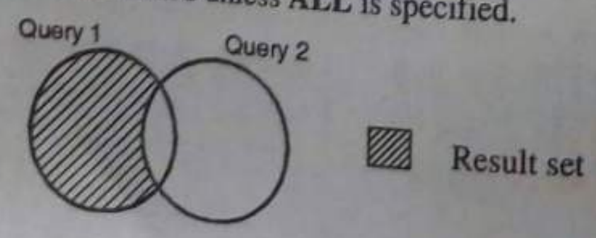
```
MySQL> SELECT * FROM Student WHERE Department='IT'
       INTERSECT
       SELECT * FROM Student WHERE Grants>=2000;

+-----+----------+-------+-----------+-------+--------+--------------+
| ID  | Name     | Marks | Department | Subject | Grants | email        |
+-----+----------+-------+-----------+-------+--------+--------------+
|   1 | Mahesh   |    90 | IT        | DB    | 5000.00 | a_bh@myc.edu |
|   3 | Jayendra |    89 | IT        | DB    | 3000.00 | a_by@myc.edu |
+-----+----------+-------+-----------+-------+--------+--------------+
2 rows in set (0.00 sec)
```

6.10.3 Difference Operator

a) Overview

- Returns all rows that are in the result of Query 1 but not in the result of Query 2.
- Again, duplicates are eliminated unless **ALL** is specified.



Find all non IT students offered grants above 2000.

/* Query 1: To print all students offered grants above 2000 */

```
MySQL> SELECT *
       FROM    Student
       WHERE   Grants>=2000;
+-----+----------+-------+-----------+-------+--------+--------------+
| ID  | Name     | Marks | Department | Subject | Grants | email        |
+-----+----------+-------+-----------+-------+--------+--------------+
|   1 | Mahesh   |    90 | IT        | DB    | 5000.00 | a_bh@myc.edu |
|   3 | Jayendra |    89 | IT        | DB    | 3000.00 | a_by@myc.edu |
|   4 | Sachin   |    99 | CS        | DB    | 7000.00 | a_bc@myc.edu |
|   6 | Payal    |    90 | MECH      | CPP   | 3000.00 | a_by@myc.edu |
+-----+----------+-------+-----------+-------+--------+--------------+
4 rows in set (0.00 sec)
```

/* Query 2 : To print all IT students */

```
MySQL> SELECT *
       FROM    Student
       WHERE   Department='IT';
+-----+----------+-------+-----------+-------+--------+--------------+
| ID  | Name     | Marks | Department | Subject | Grants | email        |
+-----+----------+-------+-----------+-------+--------+--------------+
|   1 | Mahesh   |    90 | IT        | DB    | 5000.00 | a_bh@myc.edu |
|   3 | Jayendra |    89 | IT        | DB    | 3000.00 | a_by@myc.edu |
|   5 | Vishal   |    78 | IT        | DB    |    NULL | is@myc.edu   |
+-----+----------+-------+-----------+-------+--------+--------------+
3 rows in set (0.00 sec)
```

/* Final Query : To print all IT students offered grants above 2000 */

```
MySQL> SELECT * FROM Student WHERE Grants>=2000
       MINUS
       SELECT * FROM Student WHERE Department='IT';
+-----+----------+-------+-----------+-------+--------+--------------+
| ID  | Name     | Marks | Department | Subject | Grants | email        |
+-----+----------+-------+-----------+-------+--------+--------------+
```

Scanned by CamScanner

```
|  4 | Sachin |  99 | CS      | DB   | 7000.00 | a_bc@myc.edu |
|  6 | Payal  |  90 | MECH    | CPP  | 3000.00 | a_by@myc.edu |
+----+--------+-----+---------+------+---------+--------------+
2 rows in set (0.00 sec)
```

6.11 NULL Values

1) Overview

- Concept of NULL values is introduced by father of relational database E. F. Codd.
- NULL values used to represent missing information and inapplicable information in RDBMS.
- The NULL value represents unknown, unassigned or empty values in any column of table in database.
- A value of NULL is different from an empty, White spaces (blank spaces) or zero value.
- No two NULL values are equal as both are unknown so they may not be equal.
- Null is not having any data type in SQL, meaning that it is not designated as an integer, character, or any other specific data type.
- NULL values require special handling in SQL queries.

2) Null as Third Logic (3 valued logic)

- Since Null is not a member of any columns in a table, it is not considered a value it is considered as placeholder indicating the absence of value.
- Because of this, comparisons with Null can never result in either True or False, but always in a third logical result, Unknown.

– **Not Logic table**

- As Null values do mean by "unknown" hence negation will produce same results,

P	Not P (~P)
True	False
False	True
Unknown	Unknown

– **Complete Logic table**

Comparisons between two null values, or between a NULL and any other value, return unknown because the value of each NULL is unknown.

P	Q	P OR Q	P AND Q	P = Q
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown

P	Q	P OR Q	P AND Q	P = Q
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

3) Locating Missing Values in table

Consider the student table indicate assessment marks of every student. The blanks in mark indicates absent student in exam. Assessment is conducted out of 100 marks.

```
MySQL> SELECT *
       FROM    Student;

+----+----------+-------+------------+---------+---------+-------------+
| ID | Name     | Marks | Department | Subject | Grants  | email       |
+----+----------+-------+------------+---------+---------+-------------+
|  1 | Mahesh   |  90   | IT         | DB      | 5000.00 | a_bh@myc.edu |
|  2 | Suhas    |  60   | CS         | DB      |  NULL   | uh@myc.edu  |
|  3 | Jayendra |  89   | IT         | DB      | 3000.00 | a_by@myc.edu |
|  4 | Sachin   |  99   | CS         | DB      | 7000.00 | a_bc@myc.edu |
|  5 | Vishal   |  78   | IT         | DB      |  NULL   | is@myc.edu  |
|  6 | Payal    |  90   | MECH       | CPP     | 3000.00 | a_by@myc.edu |
|  7 | Kasturi  | NULL  | MECH       | CPP     |  NULL   | a_bs@myc.edu |
+----+----------+-------+------------+---------+---------+-------------+
7 rows in set (0.03 sec)
```

Find details of all students absent for any exam.

```
MySQL> SELECT *
       FROM    Student
       WHERE   Marks = NULL;
Empty set (0.00 sec)
```

Above query will not retrieve any rows since NULL means missing value cannot be equal to anything.

The condition for selecting NULL values can be as follows;

Find details of all students absent for any exam.

Scanned by CamScanner

```
MySQL> SELECT *
       FROM    Student
       WHERE   Marks IS NULL;

+-----+---------+-------+-----------+---------+---------+-------------+
| ID  | Name    | Marks | Department | Subject | Grants | email       |
+-----+---------+-------+-----------+---------+---------+-------------+
|  7  | Kasturi | NULL  | MECH      | CPP     | NULL    | a_bs@myc.edu |
+-----+---------+-------+-----------+---------+---------+-------------+
1 row in set (0.03 sec)
```

Find details of all students present for exam.

```
MySQL> SELECT *
       FROM    Student
       WHERE   Marks IS NOT NULL;

+-----+---------+-------+-----------+---------+---------+-------------+
| ID  | Name    | Marks | Department | Subject | Grants  | email       |
+-----+---------+-------+-----------+---------+---------+-------------+
|  1  | Mahesh  |  90   | IT        | DB      | 5000.00 | a_bh@myc.edu |
|  2  | Suhas   |  60   | CS        | DB      | NULL    | uh@myc.edu  |
|  3  | Jayendra |  89  | IT        | DB      | 3000.00 | a_by@myc.edu |
|  4  | Sachin  |  99   | CS        | DB      | 7000.00 | a_bc@myc.edu |
|  5  | Vishal  |  78   | IT        | DB      | NULL    | is@myc.edu  |
|  6  | Payal   |  90   | MECH      | CPP     | 3000.00 | a_by@myc.edu |
+-----+---------+-------+-----------+---------+---------+-------------+
6 rows in set (0.00 sec)
```

4) NULL value functions

These functions are specially designed for operating NULL value.

(a) NVL()

- This function is provided by oracle and it accepts two parameters.
- If the first parameter of expression is equal to NULL, then, the value of the second parameter is returned else same parameter (parameter 1) is returned.

```
NVL (parameter 1, parameter 2);
```


Example

```
Oracle> SELECT ID, NAME, NVL(Marks, 'Absent')
        FROM   Student;

+-----+---------+--------+
| ID  | Name    | Marks  |
+-----+---------+--------+
|  1  | Mahesh  |   90   |
|  2  | Suhas   |   60   |
|  3  | Jayendra |  89   |
|  4  | Sachin  |   99   |
|  5  | Vishal  |   78   |
|  6  | Payal   |   90   |
|  7  | Kasturi | Absent |
+-----+---------+--------+
7 rows in set (0.00 sec)
```

(b) ISNULL()

- This function accepts two parameters and supported by MySQL.
- If parameter of expression is equal to NULL, then, 0 will be returned or else 1 will be the output.

```
ISNULL (parameter 1);
```

Example

```
MySQL> SELECT ID, NAME, ISNULL(Marks)
       FROM   Student;

+-----+---------+--------+
| ID  | Name    | Marks  |
+-----+---------+--------+
|  1  | Mahesh  |   0    |
|  2  | Suhas   |   0    |
|  3  | Jayendra |  0    |
|  4  | Sachin  |   0    |
|  5  | Vishal  |   0    |
|  6  | Payal   |   0    |
|  7  | Kasturi |   1    |
+-----+---------+--------+
7 rows in set (0.00 sec)
```

Scanned by CamScanner

6.12 Solved Designing Problem

Example 6.12.1

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)

Solution :

Write a Query to select all courses available in institute.

```
MySQL> SELECT *
       FROM   COURSE
```

Example 6.12.2

For the given database, write SQL queries :

employee (eid, employee_name, street, city)

Works (eid, cid, salary)

Company (cid, company_name, city)

Manager (eid, manager_name)

Solution :

(i) Find the names of all employees having "i" as the second letter in their names.

```
MySQL> SELECT Employee_name, street, city
       FROM   Employee
       WHERE  Employee_name like '_i%';
```

(ii) Display the annual salary of all employees.

```
MySQL> SELECT Eid, (salary)*12 [annualsalary]
       FROM   Works;
```

Example 6.12.3 May 2014, 10 Marks

Consider the following relations for a book club :

Members (Member-Id, Name, Designation, Age)

Books (Book-Id, Booktitle, BookAuthor, Bookpublisher, Bookprice)

Reserves (Member-Id, Book-Id, Date)

Solution :

Find the names of members who are professor older than 50 years.

```
MySQL> SELECT Name
       FROM   Members
       WHERE  Designation LIKE 'Professor'
       AND    Age>50;
```

Review Questions

Q. 1 Write a note on various aggregate function.

Q. 2 Explain the concept of NULL values.

Q. 3 What is NULL value ? How it is treated in query processing.

Q. 4 Write short note on : Aggregate Functions in SQL.

Q. 5 Write short notes on : Group by and order by clause in SQL.

Q. 6 Compare the WHERE clause and HAVING clause.

Q. 7 Explain the use of ORDER BY clause.

6.13 University Questions and Answers

May 2014

Q. 1 Consider the following relations for a book club :

Members (Member-Id, Name, Designation, Age)

Books (Book-Id, Booktitle, BookAuthor, Bookpublisher, Bookprice)

Reserves (Member-Id, Book-Id, Date). *(Ans.: Refer Example.6.12.3)* **(10 Marks)**

May 2016

Q. 1(d) Explain Aggregate function with example. *(Ans. : Refer section 6.5)* **(5 Marks)**

Q. 3(a)(ii) Explain the term : Group by clause *(Ans.: Refer section 6.6)* **(2 Marks)**

Dec. 2016

Q. 1(d) Explain Aggregate function with example. *(Ans. : Refer section 6.5)* **(5 Marks)**

□□□

Scanned by CamScanner

6.13 University Questions and Answers 6-39

Chapter 7 : Complex Queries 7-1 to 7-40

✓ **Syllabus Topic :** Recursive Queries 7-1
7.1 SQL Joins Concept 7-1
7.2 Cartesian product / Cross join 7-3
✓ **Syllabus Topic :** Inner join 7-5
7.3 Inner join 7-5
✓ **Syllabus Topic :** Outer join (Left Outer, Right Outer, Full Outer) 7-12
7.4 Outer join 7-12
✓ **Syllabus Topic :** Nested Sub Queries 7-16
7.5 Nested Sub Queries 7-16
7.6 Independent Subquery 7-17
✓ **Syllabus Topic :** Subqueries with IN, EXISTS and ANY / ALL Clause 7-19
7.7 Multiple Row Subquery 7-19
7.8 Correlated Sub Queries 7-27
7.9 Solved Examples 7-30
7.10 University Questions and Answers 7-38

Scanned by CamScanner

Chapter 8 : SQL Security **8-1 to 8-21**

✓ **Syllabus Topic :** Views in SQL 8-1

8.1 Introduction of Views 8-1

8.2 Creating a Views 8-3

✓ **Syllabus Topic :** Dropping Views 8-5

8.3 Dropping Views 8-5

8.4 Modifying a Views 8-6

8.4.1 Advantages of Views 8-6

8.4.2 Disadvantages of Views 8-7

✓ **Syllabus Topic :** Event Condition Action (ECA) model (Triggers) in SQL 8-8

8.5 Trigger 8-8

✓ **Syllabus Topic :** Security 8-12

8.6 Security 8-12

✓ **Syllabus Topic :** Authorization in SQL 8-14

8.7 Authorization 8-14

✓ **Syllabus Topic :** Database Programming with JDBC 8-15

8.8 JDBC (Java Database Connectivity) 8-15

8.9 University Questions And Answers 8-21

Chapter 9 : PLSQL **9-1 to 9-30**

✓ **Syllabus Topic :** Stored Procsdures 9-1

9.1 Stored Procedure 9-1

9.2 Creating Stored Procedure 9-2

9.3 Executing Stored Procedure 9-3

9.4 Parameter Types in Stored Procedure 9-3

9.5 Parameter Modes in Stored Procedure 9-4

9.6 Altering Stored Procedure 9-6

9.7 Dropping Stored Procedure 9-7

9.8 Stored Function 9-7

Scanned by CamScanner

9.9 Compare stored procedure and Function 9-8
9.10 Concept of Cursor 9-8
9.10.1 Implicit Cursor 9-8
9.10.2 Explicit Cursor 9-10
9.10.3 Cursor Basic Loops 9-14
9.10.4 Cursor While Loops 9-18
9.11 Cursor FOR Loops 9-19
9.12 Parameterized Cursor 9-22
9.13 Cursor Variables 9-24
9.14 Sample Programs 9-26

Scanned by CamScanner

Module IV

CHAPTER 7

Complex Queries

Course Outcome

Retrieve any type of information from a data base using complex queries in SQL

Syllabus Covered

- Recursive Queries (SQL JOINS)
- Nested Queries (Nested Sub Queries)

Programming Tool : Oracle MySQL

Syllabus Topic : Recursive Queries

7.1 SQL Joins Concept

MU - May 2016

Q. What is JOIN ? Explain different types of JOIN along with example. **(May 2016, 2 Marks)**

- The relational operations can merge columns from two different tables to form new join table.

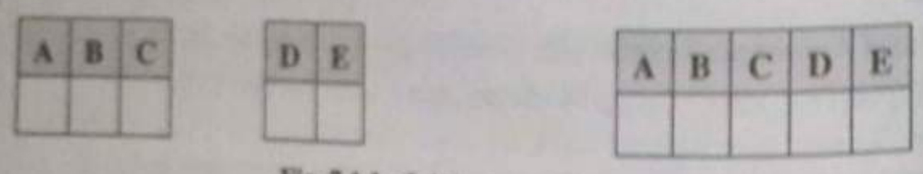


Fig. 7.1.1 : Joining operation

- We can join multiple tables with help of some join condition (Equality or Inequality) or without any join condition (cross join).
- The join concepts will be demonstrated using following HR schema which contains Employee table and Department table. These table also having a common column between them.


```
MySQL > SELECT *
    FROM   Employees;
+------+---------+---------+------------+----------+-----+-------+
| ID   | Name    | Address | Department | Salary   | Age | value |
+------+---------+---------+------------+----------+-----+-------+
| 1001 | Ganesh  | Mumbai  |         10 | 32000.00 |  26 | M     |
| 1002 | Shilpa  | Pune    |         11 | 22000.00 |  25 | F     |
| 1003 | Tanushri | Goa    |         12 | 35000.00 |  32 | F     |
| 1004 | Kasturi | Pune    |         11 | 12000.00 |  30 | F     |
| 1005 | Mayuri  | Mumbai  |         12 | 26000.00 |  31 | F     |
| 1006 | Harshad | Pune    |         10 | 38000.00 |  34 | M     |
| 1007 | Ganesh  | Pune    |       NULL | 18000.00 |  27 | M     |
| 1008 | Mahesh  | Mumbai  |         19 | 20000.00 |  31 | M     |
+------+---------+---------+------------+----------+-----+-------+
8 rows in set (0.00 sec) 8 rows in set (0.08 sec)
-----------------------------------------------------------------
MySQL> SELECT *
    FROM   Departments;
+------+------------+--------+
| ID   | Name       | value  |
+------+------------+--------+
|   10 | Technology | Mumbai |
|   11 | HR         | Mumbai |
|   12 | Admin      | Pune   |
+------+------------+--------+
3 rows in set (0.02 sec)
-----------------------------------------------------------------
MySQL> SELECT *
    FROM   Jobs;
+------+----------+----------+----------+
| ID   | Job_Title | minSalary | value    |
+------+----------+----------+----------+
| 101  | Admin    | 10000.00 | 20000.00 |
| 102  | Officer  | 20000.00 | 30000.00 |
| 101  | Executive | 30000.00 | 50000.00 |
+------+----------+----------+----------+
3 rows in set (0.00 sec)
```

Scanned by CamScanner

7.2 Cartesian product / Cross join

University Question **(May 2016, 5 Marks)**

Q. Explain different types of JOIN along with example.

- A cross join performs the relational product or Cartesian product of two tables in such product every tuple of the first table is joined with each tuple in the second table without any join condition.
- In DBMS, CROSS join occur due to missing WHERE condition in query or some invalid operations in where clause.
- 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.
- In such cross product operation all tuples are merged, although there is a common column or not and even tuple have matching or non-matching values in order to avoid non-matching tuples, always include a valid join condition in a WHERE clause.
- In CROSS joins, every tuple in table 1 will be joined with each tuple in table 2.

Employee Table

Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	100

Department Table

Did	Dname
100	HR
200	TIS

Syntax

```
SELECT   Column_List
FROM     Table1
CROSS JOIN Table2
```

Example

```
MySQL> SELECT *
   FROM    Dept;
+-----+-----------+--------+
| DID | Name      | value  |
+-----+-----------+--------+
|  10 | Technology | Mumbai |
|  11 | HR         | Mumbai |
|  12 | Admin      | Pune   |
+-----+-----------+--------+
```



```
3 rows in set (0.00 sec)

MySQL> SELECT *
   FROM    Emp;
+-----+----------+---------+-----+----------+-----+-------+
| ID  | Name     | Address | DID | Salary   | Age | value |
+-----+----------+---------+-----+----------+-----+-------+
| 1001 | Ganesh  | Mumbai  |  10 | 32000.00 |  26 | M     |
| 1002 | Shilpa  | Pune    |  11 | 22000.00 |  25 | F     |
| 1003 | Tanushri | Goa    |  12 | 35000.00 |  32 | F     |
| 1004 | Kasturi | Pune    |  11 | 12000.00 |  30 | F     |
| 1005 | Mayuri  | Mumbai  |  12 | 26000.00 |  31 | F     |
| 1006 | Harshad | Pune    |  10 | 38000.00 |  34 | M     |
| 1007 | Ganesh  | Pune    | NULL | 18000.00 |  27 | M     |
| 1008 | Mahesh  | Mumbai  |  19 | 20000.00 |  31 | M     |
+-----+----------+---------+-----+----------+-----+-------+
8 rows in set (0.02 sec)
```

Find all employees with their possible department. OR

Find all combinations of employees and departments.

```
Mysql> SELECT   e.ID, e.name, e.did, d.did, d.name
   -> FROM   Emp e CROSS JOIN Dept d;
+-----+----------+-----+-----+-----------+
| ID  | Name     | DID | DID | Name      |
+-----+----------+-----+-----+-----------+
| 1001 | Ganesh  |  10 |  10 | Technology |
| 1001 | Ganesh  |  10 |  11 | HR         |
| 1001 | Ganesh  |  10 |  12 | Admin      |
| 1002 | Shilpa  |  11 |  10 | Technology |
| 1002 | Shilpa  |  11 |  11 | HR         |
| 1002 | Shilpa  |  11 |  12 | Admin      |
   ..................................
| 1008 | Mahesh  |  19 |  10 | Technology |
| 1008 | Mahesh  |  19 |  11 | HR         |
| 1008 | Mahesh  |  19 |  12 | Admin      |
+-----+----------+-----+-----+-----------+
24 rows in set (0.00 sec)
```

Scanned by CamScanner

- In above result set every row of Employee table is merged with each tuple of department table to for a row in result set after cross join.
- The 24 rows would be returned which is multiplication of rows in employee table and rows in departments table (8*3=24). Hence, this join is also called as Cross Product Join.

Syllabus Topic : Inner join

7.3 Inner join

MU - May 2016

University Question

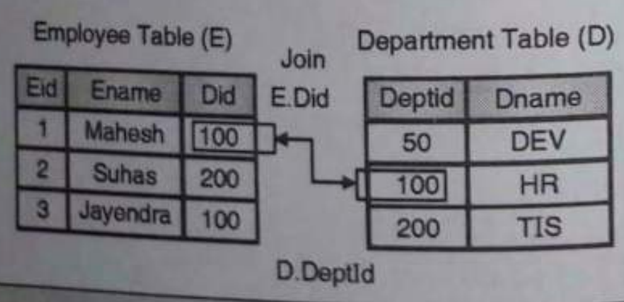
Q. Explain different types of JOIN along with example. **(May 2016, 5 Marks)**

- An INNER JOIN performs the relational product between two tables first and than use join condition to select only tuples satisfying join condition.
- Inner join will ignore all tuples that does not satisfy join condition or having missing values in it.
- Inner join will only merge tuples if merged tuples satisfy join condition. Hence it is called as Inner Join.

Syntax

```
SELECT  Column_List
FROM    Table1
 INNER JOIN Table2
 ON (JOIN_Condition);
OR
SELECT  Column_List
FROM    Table1
 INNER JOIN Table2
 ON (JOIN_Condition);
```

Example




```
MySQL> SELECT *
   FROM    Dept;
+------+----------+--------+
| DID  | Name     | value  |
```



```
+------+----------+--------+
| 10 | Technology | Mumbai |
| 11 | HR         | Mumbai |
| 12 | Admin      | Pune   |
+------+----------+--------+
3 rows in set (0.00 sec)
```

```
MySQL> SELECT *
   FROM    Emp;
+------+----------+---------+------+----------+------+-------+
| ID   | Name     | Address | DID  | Salary   | Age  | value |
+------+----------+---------+------+----------+------+-------+
| 1001 | Ganesh   | Mumbai  |  10  | 32000.00 |  26  | M     |
| 1002 | Shilpa   | Pune    |  11  | 22000.00 |  25  | F     |
| 1003 | Tanushri | Goa     |  12  | 35000.00 |  32  | F     |
| 1004 | Kasturi  | Pune    |  11  | 12000.00 |  30  | F     |
| 1005 | Mayuri   | Mumbai  |  12  | 26000.00 |  31  | F     |
| 1006 | Harshad  | Pune    |  10  | 38000.00 |  34  | M     |
| 1007 | Ganesh   | Pune    | NULL | 18000.00 |  27  | M     |
| 1008 | Mahesh   | Mumbai  |  19  | 20000.00 |  31  | M     |
+------+----------+---------+------+----------+------+-------+
8 rows in set (0.02 sec)
```

INNER Join

```
MySQL> SELECT e.ID, e.name, e.did, d.did, d.name
   FROM    Emp e
    INNER JOIN Dept d
    ON e.did=d.did;
+------+----------+------+------+------------+
| ID   | Name     | DID  | DID  | Name       |
+------+----------+------+------+------------+
| 1001 | Ganesh   |  10  |  10  | Technology |
| 1002 | Shilpa   |  11  |  11  | HR         |
| 1003 | Tanushri |  12  |  12  | Admin      |
| 1004 | Kasturi  |  11  |  11  | HR         |
| 1005 | Mayuri   |  12  |  12  | Admin      |
| 1006 | Harshad  |  10  |  10  | Technology |
+------+----------+------+------+------------+
6 rows in set (0.02 sec)
```

Scanned by CamScanner

- In above result set every row of Employee table is merged with each tuple of department table to for a row in result set after cross join.
- The 24 rows would be returned which is multiplication of rows in employee table and rows in departments table (8*3=24). Hence, this join is also called as Cross Product Join.

Syllabus Topic : Inner join

7.3 Inner join

MU - May 2016

University Question

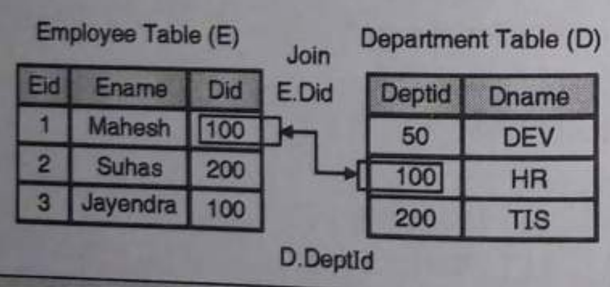
Q. Explain different types of JOIN along with example. **(May 2016, 5 Marks)**

- An INNER JOIN performs the relational product between two tables first and than use join condition to select only tuples satisfying join condition.
- Inner join will ignore all tuples that does not satisfy join condition or having missing values in it.
- Inner join will only merge tuples if merged tuples satisfy join condition. Hence it is called as Inner Join.

Syntax

```
SELECT   Column_List
FROM     Table1
 INNER JOIN Table2
 ON (JOIN_Condition);
OR
SELECT   Column_List
FROM     Table1
 INNER JOIN Table2
 ON (JOIN_Condition);
```

Example




```
MySQL> SELECT *
   FROM    Dept;
+-----+----------+-------+
| DID | Name     | value |
```



```
+-----+----------+-------+
| 10 | Technology | Mumbai |
| 11 | HR         | Mumbai |
| 12 | Admin      | Pune   |
+-----+----------+-------+
3 rows in set (0.00 sec)
```

```
MySQL> SELECT *
   FROM    Emp;
+-----+----------+---------+-----+----------+-----+-------+
| ID  | Name     | Address | DID | Salary   | Age | value |
+-----+----------+---------+-----+----------+-----+-------+
| 1001 | Ganesh   | Mumbai  | 10 | 32000.00 | 26 | M |
| 1002 | Shilpa   | Pune    | 11 | 22000.00 | 25 | F |
| 1003 | Tanushri | Goa     | 12 | 35000.00 | 32 | F |
| 1004 | Kasturi  | Pune    | 11 | 12000.00 | 30 | F |
| 1005 | Mayuri   | Mumbai  | 12 | 26000.00 | 31 | F |
| 1006 | Harshad  | Pune    | 10 | 38000.00 | 34 | M |
| 1007 | Ganesh   | Pune    | NULL | 18000.00 | 27 | M |
| 1008 | Mahesh   | Mumbai  | 19 | 20000.00 | 31 | M |
+-----+----------+---------+-----+----------+-----+-------+
8 rows in set (0.02 sec)
```

INNER Join

```
MySQL> SELECT e.ID, e.name, e.did, d.did, d.name
   FROM    Emp e
    INNER JOIN Dept d
    ON e.did=d.did;
+-----+----------+-----+-----+----------+
| ID  | Name     | DID | DID | Name     |
+-----+----------+-----+-----+----------+
| 1001 | Ganesh   | 10 | 10 | Technology |
| 1002 | Shilpa   | 11 | 11 | HR         |
| 1003 | Tanushri | 12 | 12 | Admin      |
| 1004 | Kasturi  | 11 | 11 | HR         |
| 1005 | Mayuri   | 12 | 12 | Admin      |
| 1006 | Harshad  | 10 | 10 | Technology |
+-----+----------+-----+-----+----------+
6 rows in set (0.02 sec)
```

Scanned by CamScanner

- In MySQL the INNER JOIN selects all tuples from both participating tables if and only if both tables meet the conditions specified in the ON clause.

MU - May 2016

Case 1 : Natural join

University Question

Q. Explain different types of JOIN along with example. **(May 2016, 5 Marks)**

- A natural join will selects all tuples from both participating tables, if and only if both tables has one common column with same attribute value in both tables.
- In this type of join, joining tables must have at least one common column between them which has same column name and data type.
- **Working**
  - Join processor looks for a columns present in both tables.
  - Perform Cross Join between participating tables (unconditional step).
  - Retain row in result if it has matching attribute values in common column.

Syntax

```
SELECT   Column_List
FROM     Table1
    NATURAL JOIN Table2;
```

Example

```
MySQL> SELECT *
    FROM     Dept;
+-----+-----------+-------+
| DID | Name      | value |
+-----+-----------+-------+
|  10 | Technology | Mumbai |
|  11 | HR         | Mumbai |
|  12 | Admin      | Pune   |
+-----+-----------+-------+
3 rows in set (0.00 sec)
-----------------------------------------------------------
MySQL> SELECT *
    FROM     Emp;
+-----+---------+---------+-----+---------+-----+-------+
| ID  | Name    | Address | DID | Salary  | Age | value |
+-----+---------+---------+-----+---------+-----+-------+
```



```
| 1001 | Ganesh   | Mumbai |  10 | 32000.00 |  26 | M |
| 1002 | Shilpa   | Pune   |  11 | 22000.00 |  25 | F |
| 1003 | Tanushri | Goa    |  12 | 35000.00 |  32 | F |
| 1004 | Kasturi  | Pune   |  11 | 12000.00 |  30 | F |
| 1005 | Mayuri   | Mumbai |  12 | 26000.00 |  31 | F |
| 1006 | Harshad  | Pune   |  10 | 38000.00 |  34 | M |
| 1007 | Ganesh   | Pune   | NULL | 18000.00 |  27 | M |
| 1008 | Mahesh   | Mumbai |  19 | 20000.00 |  31 | M |
+-----+---------+---------+-----+---------+-----+-------+
8 rows in set (0.02 sec)
-----------------------------------------------------------
    Natural Join
-----------------------------------------------------------
MySQL > SELECT *
    FROM Emps NATURAL JOIN Depts;
+-----+-----+---------+---------+---------+-----+----------+
| DID | ID  | Name    | Address | Salary  | Age | Dname    |
+-----+-----+---------+---------+---------+-----+----------+
|  10 | 1001 | Ganesh  | Mumbai  | 32000.00 |  26 | Technology |
|  11 | 1002 | Shilpa  | Pune    | 22000.00 |  25 | HR         |
|  12 | 1003 | Tanushri | Goa    | 35000.00 |  32 | Admin      |
|  11 | 1004 | Kasturi | Pune    | 12000.00 |  30 | HR         |
|  12 | 1005 | Mayuri  | Mumbai  | 26000.00 |  31 | Admin      |
|  10 | 1006 | Harshad | Pune    | 38000.00 |  34 | Technology |
+-----+-----+---------+---------+---------+-----+----------+
6 rows in set (0.00 sec)
```

- In MySQL the INNER JOIN selects all tuples from both participating tables if and only if both tables meet the conditions specified in the ON clause.

MU - May 2016

Case 2 : Self join

University Question

Q. Explain different types of JOIN along with example. **(May 2016, 5 Marks)**

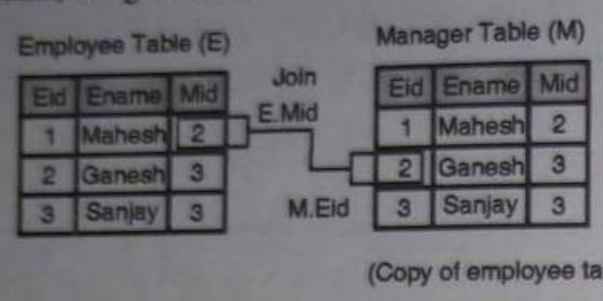
- A Self-join will work exactly same as Inner join but there is only one participating table and both columns involved in joining are in the same table.
- The tables being joined in a query do not need to be distinct; you can join any table to itself.

Scanned by CamScanner

- Self-joins are useful for discovering relationships between different columns of 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 ID (Employee ID) and Mid (Manager Id) column, using the table aliases E and M to distinguish the table references.




```
MySQL> SELECT *
   FROM    Emps;
+------+---------+---------+-----+----------+-----+------+
| ID   | Name    | Address | DID | Salary   | Age | MID  |
+------+---------+---------+-----+----------+-----+------+
| 1001 | Ganesh  | Mumbai  |  10 | 32000.00 |  26 | 1004 |
| 1002 | Shilpa  | Pune    |  11 | 22000.00 |  25 | 1004 |
| 1003 | Tanushri | Goa    |  12 | 35000.00 |  32 | 1004 |
| 1004 | Kasturi | Pune    |  11 | 12000.00 |  30 | 1006 |
| 1005 | Mayuri  | Mumbai  |  12 | 26000.00 |  31 | 1006 |
| 1006 | Harshad | Pune    |  10 | 38000.00 |  34 | 1006 |
| 1007 | Ganesh  | Pune    | NULL | 18000.00 |  27 | 1006 |
| 1008 | Mahesh  | Mumbai  |  19 | 20000.00 |  31 | 1006 |
+------+---------+---------+-----+----------+-----+------+
8 rows in set (0.00 sec)
```

Self Join

```
MySQL> SELECT E.ID, E.Name, E.MID, M.ID, M.Name
    FROM   Emps E
    INNER JOIN Emps M
        ON E.MID = M.ID;
```



```
+------+---------+------+------+---------+
| ID   | Name    | MID  | ID   | Name    |
+------+---------+------+------+---------+
| 1001 | Ganesh  | 1004 | 1004 | Kasturi |
| 1002 | Shilpa  | 1004 | 1004 | Kasturi |
| 1003 | Tanushri | 1004 | 1004 | Kasturi |
| 1004 | Kasturi | 1006 | 1006 | Harshad |
| 1005 | Mayuri  | 1006 | 1006 | Harshad |
| 1006 | Harshad | 1006 | 1006 | Harshad |
| 1007 | Ganesh  | 1006 | 1006 | Harshad |
| 1008 | Mahesh  | 1006 | 1006 | Harshad |
+------+---------+------+------+---------+
8 rows in set (0.00 sec)
```

Case 3 : Non Equi-JOIN

Q. Explain non-equijoin with example.

- A Non equi-join will work exactly same as Inner join but there is equality join condition is involved in joining participating table.
- We can make use of operators like range operator (BETWEEN) or limit operator (IN).
- The join query in which equality condition is not used then such joins are called as **Non Equi-join.**

Example :

To join department and Employee table using joining condition that 'Did' column of employee table should match with 'Did' column of department table.

```
MySQL > SELECT *
   FROM    Empdt;
+------+---------+--------+
| ID   | NAME    | JOB_ID |
+------+---------+--------+
| 1001 | Ganesh  |  101   |
| 1002 | Shilpa  |  102   |
| 1003 | Tanushri |  101  |
| 1004 | Kasturi |  103   |
| 1005 | Mayuri  |  102   |
| 1006 | Harshad |  101   |
| 1007 | Ganesh  |  103   |
```

Scanned by CamScanner

```
| 1008 | Mahesh  |   102 |
+------+---------+-------+
8 rows in set (0.00 sec)
```

```
MySQL>  SELECT  *
    FROM     Jobs;
+------+-----------+----------+----------+
| ID   | Job_Title | minSalary | maxSalary |
+------+-----------+----------+----------+
| 101 | Admin     | 10000.00 | 20000.00 |
| 102 | Officer   | 20000.00 | 30000.00 |
| 101 | Executive | 30000.00 | 50000.00 |
+------+-----------+----------+----------+
3 rows in set (0.00 sec)
```

Non Equi JOIN

```
MySQL> SELECT    E.ID, E.Name, E.Job_Id, J.Job_Title,
      E.Salary, J.minsalary, J.maxsalary
    FROM      Empdt E
      INNER JOIN Jobdt J
      ON E.Salary BETWEEN J.minSalary AND J.maxSalary;
+------+----------+--------+-----------+----------+----------+----------+
| ID   | Name     | Job_Id | Job_Title | Salary   | minsalary | maxsalary |
+------+----------+--------+-----------+----------+----------+----------+
| 1001 | Ganesh   |   101 | Executive | 32000.00 | 30000.00 | 50000.00 |
| 1002 | Shilpa   |   102 | Officer   | 22000.00 | 20000.00 | 30000.00 |
| 1003 | Tanushri |   101 | Executive | 35000.00 | 30000.00 | 50000.00 |
| 1004 | Kasturi  |   103 | Admin     | 12000.00 | 10000.00 | 20000.00 |
| 1005 | Mayuri   |   102 | Officer   | 26000.00 | 20000.00 | 30000.00 |
| 1006 | Harshad  |   101 | Executive | 38000.00 | 30000.00 | 50000.00 |
| 1007 | Ganesh   |   103 | Admin     | 18000.00 | 10000.00 | 20000.00 |
| 1008 | Mahesh   |   102 | Admin     | 20000.00 | 10000.00 | 20000.00 |
| 1008 | Mahesh   |   102 | Officer   | 20000.00 | 20000.00 | 30000.00 |
+------+----------+--------+-----------+----------+----------+----------+
9 rows in set (0.00 sec)
```


Syllabus Topic : Outer join (Left Outer, Right Outer, Full Outer)

7.4 Outer join

University Question MU - May 2016

Q. Explain different types of JOIN along with example. **(May 2016, 5 Marks)**

1. Introduction

- In an inner 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, will merge two table although there is no match between attribute values of common column.
- 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 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.**

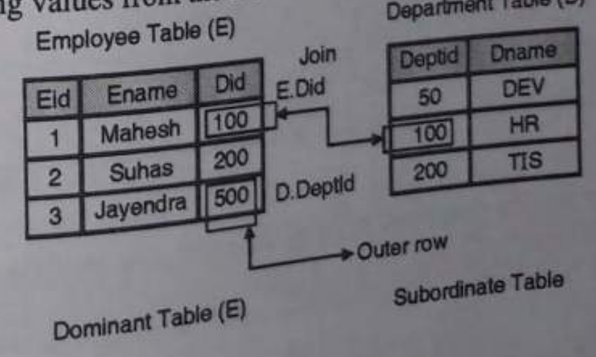
2. Syntax

```
SELECT    Column_List
FROM      Table1
      [LEFT/RIGHT/FULL] OUTER JOIN Table2
  ON Join_Condition;
```

3. Left outer join

- Left outer join has the dominant table of the outer join on left side 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

Scanned by CamScanner

```
MySQL> SELECT    E.ID, E.Name, E.DID, D.DID, D.Name
    FROM    Emps E
    LEFT OUTER JOIN Dept D
    ON E.DID=D.DID;
+------+----------+------+------+------------+
| ID   | Name     | DID  | DID  | Name       |
+------+----------+------+------+------------+
| 1001 | Ganesh   |  10  |  10  | Technology |
| 1006 | Harshad  |  10  |  10  | Technology |
| 1002 | Shilpa   |  11  |  11  | HR         |
| 1004 | Kasturi  |  11  |  11  | HR         |
| 1003 | Tanushri |  12  |  12  | Admin      |
| 1005 | Mayuri   |  12  |  12  | Admin      |
| 1007 | Ganesh   | NULL | NULL | NULL       |
| 1008 | Mahesh   |  19  | NULL | NULL       |
+------+----------+------+------+------------+
8 rows in set (0.00 sec)
```

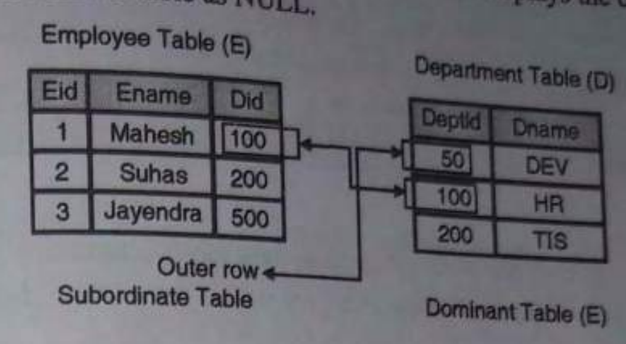
- The above table returns rows in employees table those having or not having any department. The query below will returns only rows in which employees do not have any department.
- In this query, the database server applies the filter in the WHERE clause after it performs the outer join on Did column of the Employee and Department tables.

```
MySQL> SELECT    E.ID, E.Name, E.DID, D.DID, D.Name
    FROM    Emps E
    LEFT OUTER JOIN Dept D
    ON E.DID=D.DID
    WHERE   D.DID IS NULL;
+------+----------+------+------+------+
| ID   | Name     | DID  | DID  | Name |
+------+----------+------+------+------+
| 1007 | Ganesh   | NULL | NULL | NULL |
| 1008 | Mahesh   |  19  | NULL | NULL |
+------+----------+------+------+------+
2 rows in set (0.00 sec)
```

4. Right outer join

- Right outer join has the dominant table of the outer join on right side 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.




```
MySQL> SELECT    E.ID, E.Name, E.DID, D.DID, D.Name
    FROM    Emps E
    RIGHT OUTER JOIN Dept D
    ON E.DID=D.DID;
+------+----------+------+------+------------+
| ID   | Name     | DID  | DID  | Name       |
+------+----------+------+------+------------+
| 1001 | Ganesh   |  10  |  10  | Technology |
| 1002 | Shilpa   |  11  |  11  | HR         |
| 1003 | Tanushri |  12  |  12  | Admin      |
| 1004 | Kasturi  |  11  |  11  | HR         |
| 1005 | Mayuri   |  12  |  12  | Admin      |
| 1006 | Harshad  |  10  |  10  | Technology |
| NULL | NULL     | NULL |  21  | Extra      |
+------+----------+------+------+------+
7 rows in set (0.00 sec)
```

- The above table returns rows in employees table those having or not having any department.
- The query below will returns only rows in department table without any Employees.
- In this query, the database server applies the filter in the WHERE clause after it performs the outer join on Eid column of the Employee and Department tables.

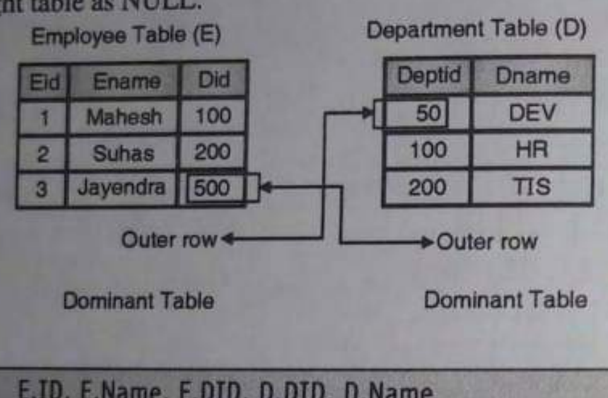
```
MySQL> SELECT    E.ID, E.Name, E.DID, D.DID, D.Name
    FROM    Emps E
    LEFT OUTER JOIN Dept D
    ON E.DID=D.DID
    WHERE   E.DID IS NULL;
```

Scanned by CamScanner

```
+------+----------+------+------+------------+
| ID   | Name     | DID  | DID  | Name       |
+------+----------+------+------+------------+
| NULL | NULL     | NULL |   21 | Extra      |
+------+----------+------+------+------------+
1 rows in set (0.00 sec)
```

5. 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.




```
MySQL> SELECT    E.ID, E.Name, E.DID, D.DID, D.Name
    FROM    Emps E
    FULL OUTER JOIN Dept D
    ON E.DID=D.DID;
+------+----------+------+------+------------+
| ID   | Name     | DID  | DID  | Name       |
+------+----------+------+------+------------+
| 1001 | Ganesh   |   10 |   10 | Technology |
| 1002 | Shilpa   |   11 |   11 | HR         |
| 1003 | Tanushri |   12 |   12 | Admin      |
| 1004 | Kasturi  |   11 |   11 | HR         |
| 1005 | Mayuri   |   12 |   12 | Admin      |
| 1006 | Harshad  |   10 |   10 | Technology |
| 1007 | Ganesh   | NULL | NULL | NULL       |
| 1008 | Mahesh   |   19 | NULL | NULL       |
```



```
| NULL | NULL     | NULL |   21 | Extra      |
+------+----------+------+------+------------+
9 rows in set (0.00 sec)
```

- The above table returns rows in employees table those having or not having any department and vice versa.

```
MySQL> SELECT    E.ID, E.Name, E.DID, D.DID, D.Name
    FROM    Emps E
    LEFT OUTER JOIN Dept D
    ON E.DID=D.DID
    WHERE   E.DID IS NULL OR D.DID IS NULL;
+------+----------+------+------+------------+
| ID   | Name     | DID  | DID  | Name       |
+------+----------+------+------+------------+
| 1007 | Ganesh   | NULL | NULL | NULL       |
| 1008 | Mahesh   |   19 | NULL | NULL       |
| NULL | NULL     | NULL |   21 | Extra      |
+------+----------+------+------+------------+
3 rows in set (0.00 sec)
```

Syllabus Topic : Nested Sub Queries

7.5 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 or Nested sub query is a query within another SQL query.
- Subquery is query appear within **WHERE** or **HAVING** clause of other query.

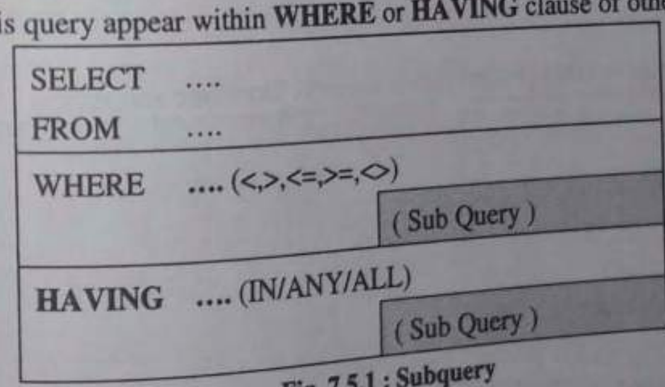


Fig. 7.5.1 : Subquery

- Outer query is called as **main query** and inner query which is written in (where or having clause) main query is called **subquery**.

Scanned by CamScanner

- **Subquery in WHERE clause :** The result of the subquery (inner query) is used to select some rows from main query.
- **Subquery in 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);
OR
SELECT    Select_list
    FROM    Table
GROUP BY Column
    HAVING  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)

7.6 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

Consider an Employee table (Empdt) has a columns like name and salary.

```
mysql>  SELECT  *
   FROM     Empdt;
+-----+----------+----------+
| ID  | NAME     | SALARY   |
+-----+----------+----------+
| 1001 | Ganesh   | 32000.00 |
| 1002 | Shilpa   | 22000.00 |
| 1003 | Tanushri | 35000.00 |
| 1004 | Kasturi  | 12000.00 |
```



```
| 1005 | Mayuri   | 26000.00 |
| 1006 | Harshad  | 38000.00 |
| 1007 | Ganesh   | 18000.00 |
| 1008 | Mahesh   | 20000.00 |
+-----+----------+----------+
8 rows in set (0.00 sec)
```

Find all employees whose salary less than Shilpa's salary.

```
MySQL>  SELECT  ID, NAME, Salary
   FROMEmpdt
   WHERE    Salary < (   SELECT    Salary
            FROM     Empdt
            WHERE  NAME LIKE 'Shilpa');
+-----+----------+----------+
| ID  | NAME     | Salary   |
+-----+----------+----------+
| 1004 | Kasturi | 12000.00 |
| 1007 | Ganesh  | 18000.00 |
| 1008 | Mahesh  | 20000.00 |
+-----+----------+----------+
3 rows in set (0.03 sec)
```

Find all departments has employees more than employees in Goa.

```
MySQL>  SELECT  ID, NAME, Salary
   FROMEmployee
   GROUP BY   Department
   HAVING   Count(ID) > ( SELECT  count(*)
            FROM     Employee
            WHERE  Address = 'GOA');
+-----+----------+----------+
| ID  | NAME     | Salary   |
+-----+----------+----------+
| 1001 | Ganesh   | 32000.00 |
| 1002 | Shilpa   | 22000.00 |
| 1003 | Tanushri | 35000.00 |
+-----+----------+----------+
3 rows in set (0.05 sec)
```

- As only one row (Value) is returned by inner query (Subquery) so these types of query is called as Single row subquery.

Scanned by CamScanner

Syllabus Topic : Subqueries with IN, EXISTS and ANY / ALL Clause

7.7 Multiple Row Subquery

- Multiple row subquery returns one or more than rows to the outer SQL statement.
- We can use the IN, ANY, or ALL operator in outer query to handle a subquery returns multiple rows.
- If subquery returns more than one row then that type of query known as multiple row sub query.
- 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.

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 Tests
3.	ALL	Compare value to every value returned by the subquery	
4.	EXISTS	Checks weather subquery returns a value or not.	Existence Tests

Consider employee details table (Empdtls) shows details of employee with department details table (Deptdtls) represents department employees working for and Job details table (Jobdtls) shows salary details of employee.

```
MySQL> SELECT *
   FROM    Empdtls;
+------+---------+---------+------------+---------+----------+
| ID   | Name    | Address | Department | Jod_id  | salary   |
+------+---------+---------+------------+---------+----------+
| 1001 | Ganesh  | Mumbai  |         10 | Admin   | 32000.00 |
| 1002 | Shilpa  | Pune    |         11 | Officer | 22000.00 |
| 1003 | Tanushri | Goa    |         12 | Admin   | 35000.00 |
| 1004 | Kasturi | Pune    |         11 | Executive | 12000.00 |
| 1005 | Mayuri  | Mumbai  |         12 | Officer | 26000.00 |
| 1006 | Harshad | Pune    |         10 | Admin   | 38000.00 |
```



```
| 1007 | Ganesh  | Pune    |      NULL | Executive | 18000.00 |
| 1008 | Mahesh  | Mumbai  |        19 | Officer   | 20000.00 |
+------+---------+---------+-----------+-----------+----------+
8 rows in set (0.00 sec)
-----------------------------------------------------------------
MySQL> SELECT *
   FROM    Deptdtls;
+------+---------+---------+
| ID   | DName   | Address |
+------+---------+---------+
| 1001 | Ganesh  | Mumbai  |
| 1002 | Shilpa  | Pune    |
| 1003 | Tanushri | Goa    |
| 1004 | Kasturi | Pune    |
| 1005 | Mayuri  | Mumbai  |
| 1006 | Harshad | Pune    |
| 1007 | Ganesh  | Pune    |
| 1008 | Mahesh  | Mumbai  |
+------+---------+---------+
8 rows in set (0.03 sec)
-----------------------------------------------------------------
MySQL> SELECT *
   FROM    Deptdtls;
+------+---------+---------+
| ID   | DName   | Address |
+------+---------+---------+
| 1001 | Ganesh  | Mumbai  |
| 1002 | Shilpa  | Pune    |
| 1003 | Tanushri | Goa    |
| 1004 | Kasturi | Pune    |
| 1005 | Mayuri  | Mumbai  |
| 1006 | Harshad | Pune    |
| 1007 | Ganesh  | Pune    |
| 1008 | Mahesh  | Mumbai  |
+------+---------+---------+
8 rows in set (0.03 sec)
-----------------------------------------------------------------
MySQL> SELECT *
```

Scanned by CamScanner

```
    FROM     Jobdtls;
+-----+----------+----------+----------+
| ID  | Job_Title | minSalary | maxSalary |
+-----+----------+----------+----------+
| 101 | Admin     | 10000.00 | 20000.00 |
| 102 | Officer   | 20000.00 | 30000.00 |
| 101 | Executive | 30000.00 | 50000.00 |
+-----+----------+----------+----------+
3 rows in set (0.04 sec)
```

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.
- IN operator used to check equlity of data.
- Inversely we can use NOT IN condition to check test value is not a member of result set of inner subquery.

Example

Find out all employees working for HR and Admin department.

```
MySQL> SELECT *
       FROM     Empdtls
     WHERE Department IN (SELECT   ID
                          FROM Deptdtls
                          WHERE Dname IN ('HR','Admin'));
+-----+----------+---------+----------+----------+----------+
| ID  | Name     | Address | Department | Jod_id   | salary   |
+-----+----------+---------+----------+----------+----------+
| 1002 | Shilpa  | Pune    |       11 | Officer  | 22000.00 |
| 1003 | Tanushri | Goa    |       12 | Admin    | 35000.00 |
| 1004 | Kasturi | Pune    |       11 | Executive | 12000.00 |
| 1005 | Mayuri  | Mumbai  |       12 | Officer  | 26000.00 |
+-----+----------+---------+----------+----------+----------+
4 rows in set (0.00 sec)
```

Find out all employees not working for departments in Mumbai and Pune.

```
MySQL> SELECT *
       FROM     Empdtls
```



```
     WHERE Department NOT IN (
                    SELECT ID
                    FROM Deptdtls
                    WHERE Address IN ('Mumbai','Pune'));
+-----+----------+---------+----------+----------+----------+
| ID  | Name     | Address | Department | Jod_id   | salary   |
+-----+----------+---------+----------+----------+----------+
| 1008 | Mahesh  | Mumbai  |       19 | Officer  | 20000.00 |
+-----+----------+---------+----------+----------+----------+
1 row in set (0.00 sec)
```

Find out all employees with salary lower than 20000 and working for departments in Mumbai.

```
MySQL> SELECT *
       FROM     Empdtls
     WHERE Salary < 40000
       AND Department NOT IN (
                    SELECT ID
                    FROM Deptdtls
                    WHERE Address = 'Mumbai');
+-----+----------+---------+----------+----------+----------+
| ID  | Name     | Address | Department | Jod_id   | salary   |
+-----+----------+---------+----------+----------+----------+
| 1003 | Tanushri | Goa    |       12 | Admin    | 35000.00 |
| 1005 | Mayuri  | Mumbai  |       12 | Officer  | 26000.00 |
| 1008 | Mahesh  | Mumbai  |       19 | Officer  | 20000.00 |
+-----+----------+---------+----------+----------+----------+
3 rows in set (0.00 sec)
```

2. ANY

- This multiple row operator used to check that a given tuple in main query satisfies given condition for at least one tuple value (out of set of values) returned by a subquery.
- Checking to see whether the comparison holds true for some values of subquery.
- It is used along with Less than (<) or greater than (>) operators for testing set comparison.

Example

Find out all employees having salary less than any of the Officer.

Scanned by CamScanner

```
MySQL> SELECT *
       FROM     Empdtls
    WHERE Salary > (   SELECT   Salary
                       FROM   Empdtls
                       WHERE Job_id = 'Officer');
ERROR 1242 (21000): Subquery returns more than 1 row
-----------------------------------------------------------
MySQL> SELECT *
       FROM     Empdtls
    WHERE Salary > ANY (SELECT Salary
                       FROM   Empdtls
                       WHERE Job_id = 'Officer');
+------+---------+--------+------------+---------+---------+
| ID   | Name    | Address | Department | salary  | Job_Id  |
+------+---------+--------+------------+---------+---------+
| 1001 | Ganesh  | Mumbai |        10 | 32000.00 | Admin   |
| 1002 | Shilpa  | Pune   |        11 | 22000.00 | Officer |
| 1003 | Tanushri | Goa   |        12 | 35000.00 | Admin   |
| 1005 | Mayuri  | Mumbai |        12 | 26000.00 | Officer |
| 1006 | Harshad | Pune   |        10 | 38000.00 | Admin   |
+------+---------+--------+------------+---------+---------+
5 rows in set (0.00 sec)
```

Find out all employees who are not officer but have salary more than any of the Officer.

```
MySQL> SELECT *
       FROM     Empdtls
    WHERE Job_Id <> 'Officer'
       AND Salary > ANY (   SELECT Salary
                            FROM   Empdtls
                            WHERE  Job_id = 'Officer');
+------+---------+--------+------------+---------+---------+
| ID   | Name    | Address | Department | salary  | Job_Id  |
+------+---------+--------+------------+---------+---------+
| 1004 | Kasturi | Pune   |        11 | 12000.00 | Executive |
| 1007 | Ganesh  | Pune   |      NULL | 18000.00 | Executive |
+------+---------+--------+------------+---------+---------+
2 rows in set (0.00 sec)
```

Find out all Executives having salary less than any one officer in employees.


```
MySQL> SELECT *
       FROM     Empdtls
    WHERE Job_Id = 'Executive'
       AND Salary < ANY (   SELECT Salary
                            FROM   Empdtls
                            WHERE  Job_id = 'Officer');
+------+---------+--------+------------+---------+---------+
| ID   | Name    | Address | Department | salary  | Job_Id  |
+------+---------+--------+------------+---------+---------+
| 1004 | Kasturi | Pune   |        11 | 12000.00 | Executive |
| 1007 | Ganesh  | Pune   |      NULL | 18000.00 | Executive |
+------+---------+--------+------------+---------+---------+
2 rows in set (0.00 sec)
```

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 the Officers.

```
MySQL> SELECT *
       FROM     Empdtls
    WHERE Salary > (   SELECT   Salary
                       FROM   Empdtls
                       WHERE Job_id = 'Officer');
ERROR 1242 (21000): Subquery returns more than 1 row
-----------------------------------------------------------
MySQL> SELECT *
       FROM     Empdtls
    WHERE Salary > ALL (SELECT Salary
                       FROM   Empdtls
                       WHERE Job_id = 'Officer');
+------+---------+--------+------------+---------+---------+
| ID   | Name    | Address | Department | salary  | Job_Id  |
+------+---------+--------+------------+---------+---------+
| 1001 | Ganesh  | Mumbai |        10 | 32000.00 | Admin   |
```

Scanned by CamScanner

```
| 1003 | Tanushri | Goa    |      12 | 35000.00 | Admin |
| 1006 | Harshad  | Pune   |      10 | 38000.00 | Admin |
+------+----------+--------+---------+----------+-------+
3 rows in set (0.00 sec)
```

Find out all employees having salary more than all employees in HR department.

```
MySQL> SELECT *
       FROM    Empdtls
       WHERE Salary > ALL
             (SELECT Salary
              FROM  Empdtls
              WHERE  Department IN (SELECT ID
                                    FROM Deptdtls
                                    WHERE Dname= 'HR'));
+------+----------+--------+----------+----------+--------+
| ID   | Name     | Address | Department | salary   | Job_Id |
+------+----------+--------+----------+----------+--------+
| 1001 | Ganesh   | Mumbai |      10 | 32000.00 | Admin  |
| 1003 | Tanushri | Goa    |      12 | 35000.00 | Admin  |
| 1005 | Mayuri   | Mumbai |      12 | 26000.00 | Officer |
| 1006 | Harshad  | Pune   |      10 | 38000.00 | Admin  |
+------+----------+--------+----------+----------+--------+
4 rows in set (0.00 sec)
```

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.
- NOT EXISTS is complementary test to above existence test checks whether a subquery does not return any values which satisfy search criteria.
- The NOT EXISTS clause results in True if the subquery is empty.

Example

Print all Employees only if there are few employees have salary more than 35000.

```
MySQL> SELECT *
       FROM    Empdtls
       WHERE EXISTS (  SELECT ID
                       FROM  Empdtls
```



```
                       WHERE Salary > 35000);
+------+----------+--------+----------+----------+--------+
| ID   | Name     | Address | Department | salary   | Job_Id |
+------+----------+--------+----------+----------+--------+
| 1001 | Ganesh   | Mumbai |      10 | 32000.00 | Admin  |
| 1002 | Shilpa   | Pune   |      11 | 22000.00 | Officer |
| 1003 | Tanushri | Goa    |      12 | 35000.00 | Admin  |
| 1004 | Kasturi  | Pune   |      11 | 12000.00 | Executive |
| 1005 | Mayuri   | Mumbai |      12 | 26000.00 | Officer |
| 1006 | Harshad  | Pune   |      10 | 38000.00 | Admin  |
| 1007 | Ganesh   | Pune   |    NULL | 18000.00 | Executive |
| 1008 | Mahesh   | Mumbai |      19 | 20000.00 | Officer |
+------+----------+--------+----------+----------+--------+
8 rows in set (0.00 sec)
```

Find out all employees having salary more than all employees in HR department.

```
MySQL> SELECT *
       FROM    Empdtls
       WHERE Salary > ALL
             (SELECT ID
              FROM  Empdtls
              WHERE  Department IN (SELECT ID
                                    FROM Deptdtls
                                    WHERE Dname= 'HR'));
+------+----------+--------+----------+----------+--------+
| ID   | Name     | Address | Department | salary   | Job_Id |
+------+----------+--------+----------+----------+--------+
| 1001 | Ganesh   | Mumbai |      10 | 32000.00 | Admin  |
| 1003 | Tanushri | Goa    |      12 | 35000.00 | Admin  |
| 1005 | Mayuri   | Mumbai |      12 | 26000.00 | Officer |
| 1006 | Harshad  | Pune   |      10 | 38000.00 | Admin  |
+------+----------+--------+----------+----------+--------+
4 rows in set (0.00 sec)
```

Print all Department if there are few employees with no department.

```
MySQL> SELECT *
       FROM    Deptdtls
       WHERE EXISTS (  SELECT ID
                       FROM  Empdtls
                       WHERE Department IS NULL);
+------+----------+--------+
| ID   | DName    | Address |
```

Scanned by CamScanner

```
+------+-------------+---------+
| 10 | Technology | Mumbai |
| 11 | HR         | Mumbai |
| 12 | Admin      | Pune   |
| 21 | Extra      | Pune   |
+------+-------------+---------+
4 rows in set (0.00 sec)
```

Print all officer employees if there are no employee work as manager.

```
MySQL> SELECT *
       FROM      Empdtls
     WHERE Job_Id = 'Officer'
        AND NOT EXISTS (   SELECT ID
                     FROM  Empdtls
                     WHERE Job_Id = 'Manager');
+------+--------+---------+-------------+---------+---------+
| ID   | Name   | Address | Department | salary  | Job_Id |
+------+--------+---------+-------------+---------+---------+
| 1002 | Shilpa | Pune    |        11 | 22000.00 | Officer |
| 1005 | Mayuri | Mumbai  |        12 | 26000.00 | Officer |
| 1008 | Mahesh | Mumbai  |        19 | 20000.00 | Officer |
+------+--------+---------+-------------+---------+---------+
3 rows in set (0.00 sec)
```

Print all officer employees if there are no employee work as Executives.

```
MySQL> SELECT *
       FROM      Empdtls
     WHERE Job_Id = 'Officer'
        AND NOT EXISTS (   SELECT ID
                     FROM  Empdtls
                     WHERE Job_Id = 'Executive');
Empty set (0.00 sec)
```

7.8 Correlated Sub Queries

(1) Introduction

- In a case of nested subquery, the first subquery will be solved then using this result, outer query will be evaluated.
- A simple subquery is evaluated only once for each query.
- A correlated subquery is evaluated once for each result row of main query.
- In a case of correlated subquery, at first main query evaluated to get first result row and than solve entire subquery for that first row of main query.

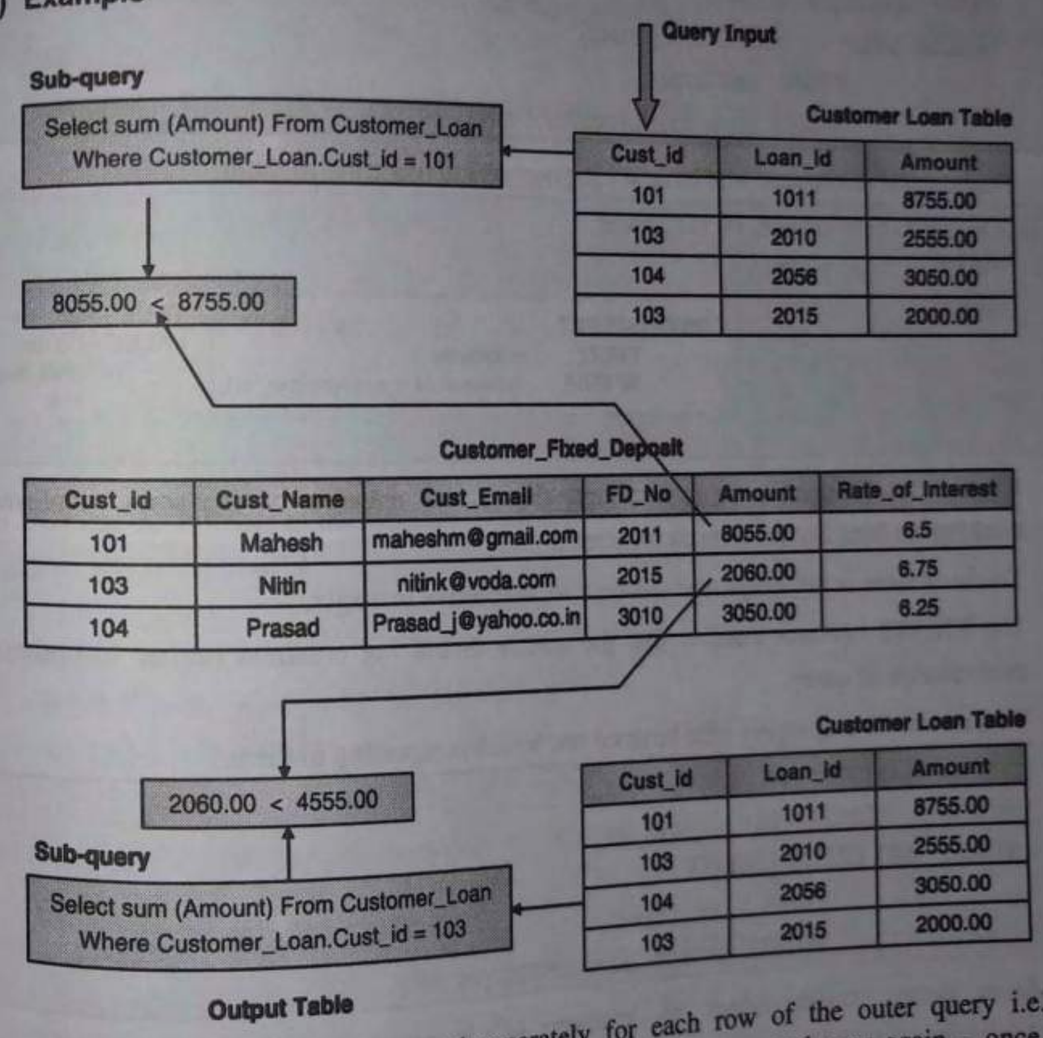
- Correlated sub queries processes data in row-by-row pattern. Each subquery is executed once for every row of the outer query.
- We can use a correlated subquery in the HAVING clause also.

(2) Syntax

```
SELECT *
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



- 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.

Fig.7.8.1

Scanned by CamScanner

Example

Find customers with fixed deposit less than the sum of all their loans.

```
MySQL> SELECT Cust_id, Cust_Name
    FROM    Customer_Fixed_Deposit CFD
    WHERE   Amount < ( SELECT SUM(Amount)
                 FROM Customer_Loan
                WHERE Cust_id = CFD.Cust_id);
```

Find all employees who earn less than the average salary in own department.

```
MySQL> 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.

```
MySQL> SELECT  employee_id, last_name
    FROM       employees e
    WHERE
          EXISTS (SELECT   'X'                          TRUE - Print row
                  FROM     employee                     FALSE - Do not
                  WHERE    manager_id = e.employee_id);          print that
                                                                 row
```

- If result set of inner query is nonempty depicts that manager having some employees working for him. So, print result on screen.
- To check there is any employee working for respective manager.
- 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.

```
MySQL> 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.

7.9 Solved Examples

Example 7.9.1

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)

Solution :

(i) Finding the total number of people who owned cars that were involved in an accident in 2007.

```
MySQL> SELECT COUNT(*)
    FROM Person
    WHERE Driver_id IN
    (Select Driverid FROM Participated
     WHERE Car IN
     (Select ReportCno FROM Accident
      WHERE Year(Date) = 2007);
```

(ii) Finding the number of accidents in which the cars belonging to "Ajay" were involved.

```
MySQL> 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.

```
MySQL> SELECT COUNT(*)
    FROM accident
    WHERE reportCno IN
    (Select report_number FROM Participated
     WHERE Car IN
     (Select ReportCno FROM Accident
```

Scanned by CamScanner

```
    WHERE location = 'Mumbai'
        AND Year(Date) = 2004);
```

Example 7.9.2

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)

Solution :

Finding all employees in the database who live in the same cities as the company for which they work.

```
MySQL> SELECT *
   FROM      Employee
   WHERE  City IN (SELECT City  FROM  Company
                   WHERE  Eid = Employee·Eid)
```

Example 7.9.3

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)

Solution :

(1) Finding all student details registered for course id 10.

```
MySQL> SELECT *
   FROM STUDENT S
   WHERE Ssn = (SELECT Ssn FROM ENROLL E
            WHERE E·Course_id = 10);
```

(2) Finding various book titles and authors for semester higher than 3.

```
MySQL> SELECT *
   FROM BOOK_ISSUED B
      NATURAL JOIN TEXT T
   WHERE  B·Semester > 3
```


(3) Finding all students belongs to IT Department (without join)

```
MySQL> SELECT Ssn, CName
   FROM     STUDENT
   WHERE    Ssn IN (SELECT Ssn FROM ENROLL
            WHERE Course_id IN
             (SELECT Course_id FROM Course
              WHERE    Dept = 'IT'))
```

(4) Finding total number of student s enrolled in IT Department.

```
MySQL> SELECT COUNT (E·Ssn) Total_Students
   FROMSTUDENT
    NATURAL JOIN ENROLL
    NATURAL JOIN Course;
```

Example 7.9.4

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".

Solution :

(i) Getting the employee name, dept-name and division name for all employees whose salary is above 20,000/-

```
MySQL> SELECT *
   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) List the dept-name and employee names in that dept, for all department whose location is "Mumbai".

```
MySQL> SELECT *
   FROM     EMPLOYEE e
        INNER JOIN Department d ON e·dept = d.dept
   WHERE    d·location = 'Mumbai';
```

Scanned by CamScanner

Example 7.9.5 May 2014, 10 Marks

Consider the following relations for a book club :

Members (Member-Id, Name, Designation, Age)

Books (Book-Id, Booktitle, BookAuthor, Bookpublisher, Bookprice)

Reserves (Member-Id, Book-Id, Date)

Solution :

(i) List the titles of books reserved by professors.

```
MySQL> SELECT Booktitle
    FROM Books
    WHERE Book-id IN
     (SELECT Book-id FROM Reserves
      WHERE Member-id IN
          (SELECT Member-id FROM Members
           WHERE Designation='Professor'));
```

(ii) Find Ids of members who have reserved books that cost more than 500.

```
MySQL> SELECT Member-id FROM Members
    WHERE Member-id IN
     (SELECT Member-id FROM Reserves
      WHERE Book-id IN
          (SELECT Book-id FROM Books
           WHERE Bookprice > 500));
```

(iii) Find the authors and titles of books reserved on 20-09-2012.

```
MySQL> SELECT Booktitle, BookAuthor
    FROM Books
    WHERE Book-id IN (SELECT Book-id
                      FROM Reserves
                      WHERE Date = '2012-09-02');
```

Example 7.9.6 Dec. 2013, 10 Marks

Consider the following employee database.

Employee (empname, street, city, date_of_joining)

Works (empname, company _name, salary)

Company (company_name, city)

Manages (empname, manager_name).

Solution :

(i) List all employees who live in the same cities as their managers.

```
MySQL> SELECT *
    FROM Employee E, Manages M
    WHERE E.empname = M.empname
     AND E.city = M.city);
```

(ii) Find all employees who earn more than average salary of all employees of their company.

```
MySQL> SELECT *
    FROM Employee E, Works W
    WHERE E.empname= W.empname
     AND W.salary > (SELECT AVG(salary)
                     FROM Employee
                     WHERE company_name = W.company_name);
```

Example 7.9.7 MU - Dec. 2015, 10 Marks

Refer education atabase mentioned in Q. 2 (a) Write SQL queries for the following :

i) List all the teacher who conduct the course titled Database Systems.

ii) List all the course offered in Thane on 15/8/15.

iii) Find the course enrolled by Monali.

iv) List all the employees who work as Teachers.

Solution

i) List all the teacher who conduct the course titled "Database Systems".

```
SELECT    *
FROM      Teacher
WHERE     course-no IN ( SELECT course-no
                         FROM   Course
                         WHERE  title = 'Database Systems')
```

ii) List all the course offered in 'Thane' on 15/8/15.

```
SELECT    *
FROM      Teacher
WHERE     location LIKE 'Thane'
      AND off-date = '15-08-2015'
```

iii) Find the course enrolled by "Monali".

```
SELECT    *
FROM      Course
WHERE     course-no IN
```

Scanned by CamScanner

(SELECT course-no FROM Enrolment WHERE stud-no IN
(SELECT stud-no FROM Student WHERE stud-name = 'Monali'))

iv) List all the employees who work as Teachers.

```
SELECT   *
FROM     Employee
WHERE    emp-no IN ( SELECT emp-no FROM Teacher )
```

Example 7.9.8 **MU - May 2015, 10 Marks**

Consider the following employee database

Employee (empname, street, city, date of join)

Works (empname, company-name, salary)

Company (company-name, city)

Manages (empname, manager-name)

Write SQL queries for the following statements :

(i) Find all employees who joined in the month of October.

(ii) Modify the" database so that 'Peter' now lives in 'Newton'.

(iii) List all employees who live in the same cities as their managers.

(iv) Find all employees who earn more than average salary of all employees of their company.

(v) Give all employees of XYZ corporation a 15 percent raise.

Solution :

(i)

```
SELECT *
FROM Employee
WHERE MONTH (date of join) = 10;
```

(ii)

```
UPDATE Employee
SET city= 'Newton'
WHERE empname = 'Peter';
```

(iii)

```
SELECT *
FROM Employee e
WHERE e. city = (SELECT emp.city
                 FROM   Employee emp, Manages  mgr
                 WHERE emp.empname=mgr.empname
                 AND      emp.empname=e.empname    )
```

(iv)

```
SELECT    e.empname
FROM      Employee e NATURAL JOIN Works w
```



```
WHERE     empname >
(SELECT avg (salary) FROM   Works w WHERE w.empname=e.empname)
```

(v) Give all employees of XYZ Corporation a 15 percent raise.

```
UPDATE    Employee
SET       salary = 1.15*salary
WHERE     empname > (SELECT empname
                FROM   Works w
                WHERE company-name= 'XYZ corporation')
```

Example 7.9.9 **MU - May 2016, 10 Marks**

Consider the following employee database

Employee (emp_name,street,city,date_of_joining)

Works (emp name, company_name, salary)

Company (company_name, city)

Manages (emp _ name,manager_name)

Write SQL queries for following :

(i) Modify the database so that 'Deepa' lives in 'Pune.

Solution :

```
Update Employee
Set city ='Pune'
Where emp_name= 'Deepa';
```

(ii) Give all employees of 'XYZ corporation' a 10% rise in salary.

```
MySQL>  SELECT *
   FROM Employee E, Manages M
   WHERE E.empname = M.empname
    AND E.city = M.city);
```

(iii) List all employees who lives in the same city as their companycity.

```
MySQL>  SELECT *
   FROM Employee E, Manages M
   WHERE E.empname = M.empname
    AND E.city = M.city);
```

(iv) Display all employees who joined in the month of 'March'.

```
SELECT *
FROM Employee
WHERE month(date_of_joining) = 3;
```

Scanned by CamScanner

(v) Find all employees who earn more than average salary of all employees of their company. *(Ref. Dec. 2013 Q. 6 Chapt 7)*

```
MySQL> SELECT *
  FROM Employee E, Works W
  WHERE E.empname= W.empname
   AND W.salary > (SELECT AVG(salary)
                   FROM Employee
                   WHERE company_name = W.company_name);
```

Review Questions

Q. 1 What is outer join ?

Q. 2 Explain external join. How is it different from natural join ?

Q. 3 Explain the concepts of Outer joins.

Q. 4 Explain self join with suitable Example.

Q. 5 What are JOINS ? What are different types of JOINS give example of each.

Q. 6 What is outer join and why it is used ? Give explain its types with example.

Q. 7 Give difference between.

(a) Inner Join and outer Join

(b) Left Outer Join and Right outer Join

(c) Equi Join and Non Equi Join

(d) Equi Join and Self Join

Q. 8 What is a view ? How is it created and stored ?

Q. 9 Specify the following queries in SQL based on given schema :

Suppliers (sid, sname, city)

Part (pid, pname, color)

Sp (sid, pid, quantity)

(i) Get the names of suppliers whose name begins with R.

(ii) Get pairs of supplier numbers that both operate from same city.

(iii) Get the names of suppliers who supply part P2.

(iv) Get the supplier id and names of suppliers in descending order of city.

(v) Get the part numbers and total quantity supplied.

Q. 10 For the following relational schema :

Employee (EmpId, EmpName, Street, City)

Works (EmpId, CompId, Salary)

Company (CompId, CompName, City)

Give an expression in SQL for each of the following queries :

(i) Find the names of employees whose salary is more than the average salary of all employees of their company.

(ii) Find the names of companies that has the largest number of employees.

(iii) Find names of employees whose salary is more than that of every employee of company named 'ABCD'.

(iv) Find names of companies located in the city in which company 'ABCD' is located.

(v) Give all employees a 20 percent raise in salary whose city is different from the city they work in.

Q. 11 Consider the following schema :

EMPLOYEE (EID, EmployeeName, Street, City, Deptt, CompanyName)

COMPANY (CompanyName, City)

WORKS (EmployeeName, CompanyName, Salary)

MANAGES (EmployeeName, ManagerName)

Write SQL queries for the following :

(i) Find out the names of all employees that have 'A' anywhere in their name and are in department 'IT'.

(ii) List the names of departments in ascending order and their employees in descending order.

(iii) Find the names, city, deptt of all employee who work for 'TCS'

(iv) Find the name of employee who earns salary more than 30000.

(v) List all manager names.

Q. 12 Explain concept of sub query.

Q. 13 Describe various operators for multiple sub query.

Q. 14 What is nested sub query ? Explain ANY ALL operators with example.

Q. 15 What is correlated sub query ? Explain with example.

7.10 University Questions and Answers

Dec. 2013

Q. 1 Consider the following employee database.

Employee (empname, street, city, date_of_joining)

Works (empname, company _name, salary)

Company (company_name, city)

Manages (empname, manager_name). **(10 Marks)**

(Ans. : Refer Example 7.9.6)

May 2014

Q. 2 Consider the following relations for a book club :

Members (Member-Id, Name, Designation, Age)

Books (Book-Id, Booktitle, BookAuthor, Bookpublisher, Bookprice) **(10 Marks)**

Reserves (Member-Id, Book-Id, Date) *(Ans.: Refer Example 7.9.5)*

Scanned by CamScanner

2015

Consider the following employee database
Employee (empname, street, city, date of join)
Works (empname, company-name, salary)
Company (company-name, city)
Manages (empname, manager-name)
Write SQL queries for the following statements :

(i) Find all employees who joined in the month of October.
(ii) Modify the" database so that 'Peter' now lives in 'Newton'.
(iii) List all employees who live in the same cities as their managers.
(iv) Find all employees who earn more than average salary of all employees of their company.
(v) Give all employees of XYZ corporation a 15 percent raise. **(10 Marks)**

(Ans.: Refer Example 7.9.8)

2015

Refer education database mentioned in Q. 2 (a). Write SQL queries for the following :

(i) List all the teacher who conduct the course titled Database Systems.
(ii) List all the course offered in Thane on 15/8/15.
(iii) Find the course enrolled by Monali.
(iv) List all the employees who work as Teachers. **(10 Marks)**

(Ans.: Refer Example 7.9.7)

2016

a) What is JOIN ? Explain different types of JOIN along with example. **(10 Marks)**
(Ans. : Refer Sections 7.1,7.2,7.3 and 7.4)

b) Consider the following employee database
Employee (emp_name,street,city,date_of_joining)
Works (emp name, company_name, salary)
Company (company_name, city)
Manages (emp _ name,manager_name)
Write SQL queries for following :

(i) Modify the database so that 'Deepa' lives in 'Pune.
(ii) Give all employees of 'XYZ corporation' a 10% rise in salary.
(iii) List all employees who lives in the same city as their companycity.
(iv) Display all employees who joined in the month of 'March'.
(v) Find all employees who earn more than average salary of allemployees of their company.

(Ans. : Refer Example 7.9.9) **(10 Marks)**

□□□

CHAPTER 8

Module IV

SQL Security

Course Outcome

Create and populate a RDBMS for a real life application, Retrieve any type of information from a data base using complex queries in SQL.

Syllabus Covered

- Views in SQL
- Event Condition Action (ECA) model (Triggers) in SQL
- Security and Authorization in SQL
- Database Programming with JDBC

Programming Tool : Oracle MySQL

Syllabus Topic : Views in SQL

8.1 Introduction of Views

MU - May 2012, May 2015, May 2016, May 2017

University Questions

Q. What is a view ? How is it created and stored ? **(May 2012, May 2016, 4 Marks)**
Q. What is the view in SQL, how it is defined ? Discuss the problem that may arise when we attempt to update a view. How views are implemented ? **(May 2015, 10 Marks)**
Q. Describe view. **May 2017, 3 Marks)**

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.

Scanned by CamScanner

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)

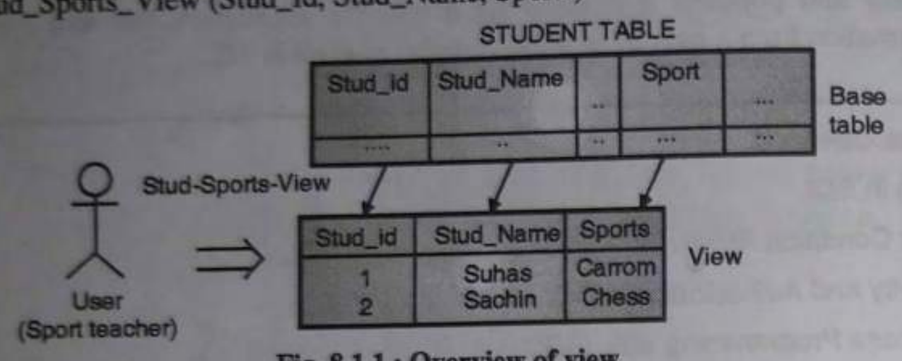


Fig. 8.1.1 : Overview of view

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.

8.2 Creating a Views

Q. Explain syntax for creating views.

1. Introduction

- To create a view a subquery must be embedded within the CREATE VIEW statement.
- A simple query is designed and its output can be recorded as a view.
- 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 must have privileges to access all of the base tables on which view is defined.
- Also, the user must have create view permissions from DBA to create a view in the database.
- The create view can change the name of the 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.

```
SQL>CREATE VIEW IT_Faculty
  AS
```

Scanned by CamScanner

```
SELECT   *
FROM     FACULTY
WHERE    Faculty_Dept= 'IT';

/* Selecting Data */

SQL>SELECT  *
FROM     IT_Faculty;
```

- Consider a default HR schema we will create a view of employees having salary below 3000 and retrieve data from view.

```
SQL> CREATE VIEW EmpBelow3K
 2 AS
 3 SELECT *
 4 FROM Emp
 5 WHERE Sal < 3000;

View created.

SQL> SELECT * FROM EmpBelow3K;

 EMPNO ENAME     JOB       HIREDATE      SAL     COMM    DEPTNO
------- ---------- --------- --------- ---------- ---------- ----------
  7698 BLAKE     MANAGER   01-MAY-81     2850     30
  7782 CLARK     MANAGER   09-JUN-81     2450     10
  7566 JONES     MANAGER   02-APR-81     2975     20
  7369 SMITH     CLERK     17-DEC-80     800      20
  7499 ALLEN     SALESMAN  20-FEB-81     1600     300      30
  7521 WARD      SALESMAN  22-FEB-81     1250     500      30
  7654 MARTIN    SALESMAN  28-SEP-81     1250     1400     30
  7844 TURNER    SALESMAN  08-SEP-81     1500     0        30
  7876 ADAMS     CLERK     12-JAN-83     1100              20
  7900 JAMES     CLERK     03-DEC-81     950               30
  7934 MILLER    CLERK     23-JAN-82     1300              10

11 rows selected.
```

Create a view of employee's jobs having salary below 3000 and retrieve data from view.

```
SQL> CREATE VIEW EmpBelow3K
 2 AS
```



```
3 SELECT DISTINCT(Job)
4 FROM Emp
5 WHERE Sal < 3000;

View created.

SQL> SELECT * FROM JobBelow3K;
JOB
---------
CLERK
SALESMAN
MANAGER

3 rows selected.
```

Syllabus Topic : Dropping Views

8.3 Dropping Views

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 created in above step.

```
SQL>DROP VIEW JobBelow3K;

View Dropped;
```

Scanned by CamScanner

8.4 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]
[WITH READ ONLY];
```

Example

Consider a view defined above 'IT_Faculty' and change it to select all IT faculties having salary above Rs. 25000.

```
SQL>CREATE OR REPLACE VIEW IT_Faculty
  AS
  SELECT   *
  FROM     FACULTY
  WHERE    Faculty_Dept= 'IT'
  AND Salary > 25000;
```

8.4.1 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) Does not allows direct access to the tables of 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.

8.4.2 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.

Scanned by CamScanner

Syllabus Topic : Event Condition Action (ECA) model (Triggers) in SQL

MU - Dec. 2014, May 2017

8.5 Trigger

University Questions

Q. What are triggers ? Explain with example. **(Dec. 2014, 10 Marks)**

Q. Describe Trigger. **(May 2017, 3 Marks)**

1. Introduction

- 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 objects.
- 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.

2. Components of Trigger (E-C-A model)

- **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.

3. 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;
```

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

Fig. 8.5.1 : Trigger parameters

4. 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 a triggering statement do not affects 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 execute 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

5. Trigger example

- Creating a trigger on employee table whenever new employee added a comment is written to EmpLog Table.
- Let us writ a trigger and study its effect.

Example

```
SQL> CREATE OR REPLACE TRIGGER AutoRecruit
  2  AFTER INSERT ON EMP
```

Scanned by CamScanner

```
3   FOR EACH ROW
4   BEGIN
5       Insert into EmpLog values ('Employee Inserted');
6   END;
7   /

Trigger created.

SQL> INSERT INTO EMP
 2  VALUES (1,'Mahesh','Manager','1-JAN-1986',3000,null,10);

1 row created.

SQL> SELECT * FROM EmpLog;

STATUS
------------------------------
Employee Inserted
```

- Consider other example, whenever there comes a new student add him to CS (Computer Science).

Example

```
SQL>CREATE TRIGGER CSAutoRecruit
    AFTER INSERT ON Student
    FOR EACH ROW
    BEGIN
    INSERT INTO Take VALUES (111, 'CS');
    END;
```

6. Trigger operations

(a) Data dictionary for triggers

Once triggers are created their definitions can be viewed by selecting it from system tables as shown below :

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};
```

7. Trigger advantages

- 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.
- After finding that the new information is inconsistent with the database, raise an error that will cause the entire transaction to roll back.

8. Trigger disadvantages

A trigger **hampers the performance of system** as database operations will go on slower due to triggering action.

- **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.
- **Mutating table errors :**
  - This happens when the trigger is querying or modifying table whose modification activated 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.

9. Solution for mutating table error

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

Scanned by CamScanner

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.

Syllabus Topic : Security

8.6 Security

MU - Dec. 2012

University Question

Q. Explain security in SQL. **(Dec. 2012, 5 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.
- **Example :** 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.
- **Example :** Only account department can change financial details of company.

(c) Availability

- Authorized users can be able to accesses data any time he wants.
- **Example :** 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 Mechanisms

(a) Access control

- 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.
- **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 can'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) Inference 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.

Scanned by CamScanner

Syllabus Topic : Authorization in SQL

8.7 Authorization

MU - May 2012, Dec. 2012

University Questions

Q. Explain Authorization in SQL. **(Dec. 2012, 5 Marks)**

Q. Write short notes on : Authorization in SQL. **(May 2012, 5 Marks)**

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. **Example :** 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. 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

4. ROLE Based Access Control (RBAC)

- 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 analysing 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.

5. Discretionary Access Control

- To maintain access rights of each database user discretionary access control is used. It allows multiple users to access some object (i.e. grants the privileges) in a specified mode, like read, write or combination of these.
- Privilege is set of actions that a user can perform on a database object are called the privileges. It 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.
- This access control is done by two commands GRANT and REVOKE.
- GRANT command helps us to give user privileges to base table and views.

Account creation

- Allows to create a new user account and password for same.
- Allows to create a new user group (Role).

Grant Privilege

DBA can grant some privilege to certain user or role.

Revoke Privilege

DBA can revoke (cancel) all privileges from user or role which are given by him.

Syllabus Topic : Database Programming with JDBC

8.8 JDBC (Java Database Connectivity)

Q. Explain the detailed concept of Database connectivity using JDBC.

1. Introduction

(a) **Java Database Connectivity (JDBC)** is a java API enables the java programs to execute SQL statements.

(b) JDBC provides some methods for querying and updating the data in Database Management system such as SQL, Oracle etc.

(c) JDBC connections which supports creating and executing statements such as SQL INSERT, UPDATE and DELETE.

(d) Driver Manager is the backbone of the JDBC architecture.

Scanned by CamScanner

2. Types of JDBC Drivers

A) JDBC Type 1 : JDBC-ODBC Bridge Driver

Q. Write a note on : JDBC-ODBC Bridge driver.

1. Architecture

- Lower level JDBC Driver which makes use of he ODBC driver to connect to the database.
- The driver converts JDBC method calls into ODBC calls and sends them to the ODBC driver.
- This implies that the ODBC driver, as well as the client database code, installed on the client machine.

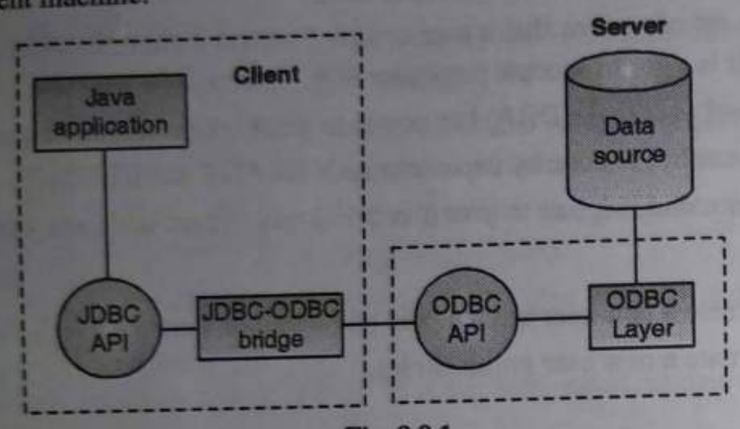


Fig. 8.8.1

Advantages

- Many data base contains ODBC drivers so can be implemented generally.
- Easy Installation
- Very useful for which only a small number of clients access to the database.

Disadvantages

- Partially written in Java which makes it not portable.
- Not suitable for Web Applications
- Performance overhead as JDBC calls is converted into ODBC calls and then passed on to the ODBC driver.
- ODBC driver required on the client machine

B) JDBC Type 2 : Native-API/Partly Java Driver

Q. Write a note on : Partly Java driver.

1. Architecture

- Lower-level JDBC Driver API that makes use of the client-side libraries of the database to connect.

- The driver converts JDBC calls into database-specific calls.
- The type 2 driver is not written entirely in Java as it interfaces with non-Java code that makes the final database calls.

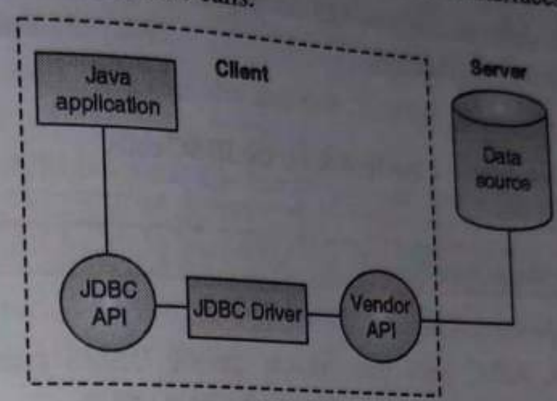


Fig. 8.8.2

2. Advantages

No ODBC drivers are required.

3. Disadvantages

- Database vendor specific client API libraries must be installed on the client machine.
- Not all the database vendors provide client API libraries.
- Not suitable for Web Applications

C) JDBC Type 3 : Network Protocol Driver

Q. Write a note on : Network Protocol driver.

- Known as the Pure Java Driver for database Middleware.
- Three-tiered approach where the client code sends the JDBC calls through the network to a middle-tier server.
- The middle-tier converts the JDBC calls directly or indirectly into database specific protocol, which will forward the JDBC calls to the database.

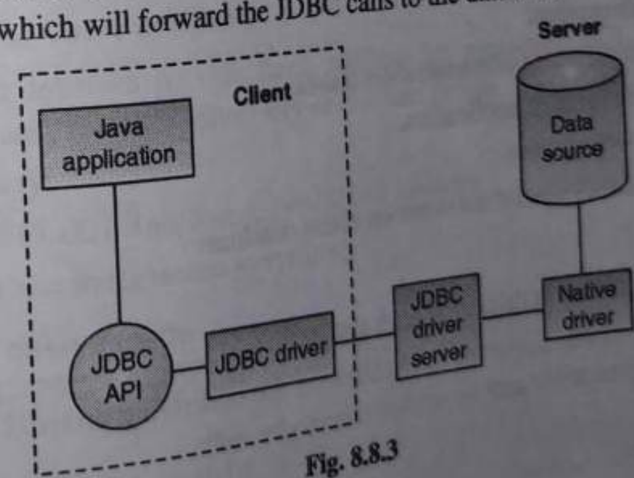


Fig. 8.8.3

Scanned by CamScanner

1. **Advantages**
   - No database library is required on the client side.
   - It can provide additional services like caching, auditing etc.
   - Well suited for web applications
2. **Disadvantages**

The additional layer might be a bottleneck for the JDBC calls.

D) Type 4 : Pure Java Driver

Q. Write a note on : Pure Java driver

- Completely implemented in java
- Driver converts JDBC calls into vendor specific DBMS protocol so that client applications can directly communicate with database server.
- It is also called as thin client driver as all processing at server.
- At client side no processing related to database occurs.

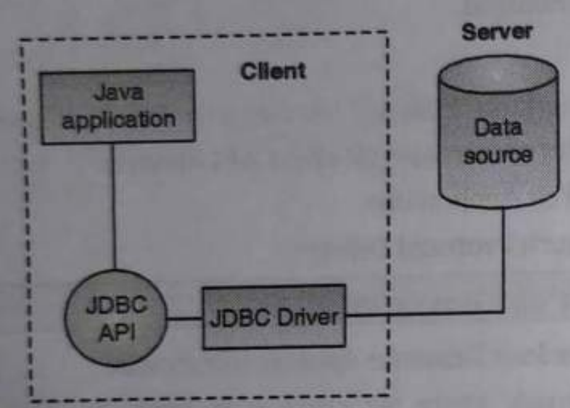


Fig. 8.8.4

1. **Advantages**
   - Platform independent
   - Eliminate deployment administration issues.
   - Most suitable for web application.
   - Very good performance
   - No need to install special software on client machine.
2. **Disadvantages**
   - User needs a different driver for each database.
   - May not be suitable for some applications if the underlying protocol does not handle issues such as security and network connectivity well.

3. Java Database Connectivity Steps

Q. Explain steps of creating JDBC connection.

1. **Define the Connection URL**
   - Connection URL specifies the database location as well as driver configuration information.
   - String oracleURL = jdbc:oracle:thin:@ hostname :port : dbName;

```
try
{
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver ");
} catch(ClassNotFoundException e)
{
  System.out.println("Error loading driver: " + e);
}
```

2. **Establish (open) database the Connection**
   - Most database applications operate in a client-server environment.
   - JDBC applications act as clients and to work with the server they must establish a physical connection to it.
   - String username = "User_id";
   - String password = "pwd";
   - Connection conn =DriverManager.getConnection(oracleURL, username, password);
3. **Create a Statement Object**
   - A Statement object is used to send queries and commands to the database.
   - To instantiate a Statement object we call the Connection.createStatement() method.

   Statement stmt = conn.createStatement();
4. **Execute (SQL) Query or Update**
   - Statement interface provide three method for executing SQL statements. These methods are execute,executeQuery or executeUpdate.
   - String query;

   query = "SELECT Rno, Name, Fee FROM students";

   ResultSet rs = stmt.executeQuery(query);
5. **Display the query results**

```
while(rs.next())
{
    //Retrieveing data by using column name
    int rno= rs.getInt("rno");
```

Scanned by CamScanner

```
        String name = rs.getString("name");

        //Retrieving data by using column index
        double fee = rs.getDouble(3);
        //Display values
        System.out.print("RNO: " + rno);
        System.out.print(" Name: " + name);
        System.out.print(" Fee: " + fee);
    }
```

6. Close the Connection

- Closing database resources is simple: Just call the close() method of each JDBC object.

```
        //Clean-up environment
        rs.close();
        statement.close();
        connection .close();
```

- We have close all ResultSet, Statement, and Connection objects.
- In general, closing the Connection object also closes the ResultSet and Statement object.

Review Questions

Q. 1 What are Triggers ? Give an example.

Q. 2 What is a view ? How is it created and stored ?

Q. 3 What is view in SQL ? Discuss the problem that may arise when we attempt to update a view. How views are implemented ?

Q. 4 Write short notes on Views in SQL.

Q. 5 Explain syntax for creating views.

Q. 6 Explain various security mechanisms in databases.

Q. 7 What is view in SQL, how it is defined? Discuss the problem that may arise when we attempt to update a view. How views are implemented ?

Q. 8 What do mean by authorization, authentication and access control in databases explain difference between them ?

Q. 9 Discuss the different security and authorization mechanisms in DBMS.

Q. 10 What do you mean by authorization in DBMS ? What is granting of privileges ?

Q. 11 Explain security and authorization in SQL.

Q. 12 Explain working of triggers with help of example. In which conditions Triggers cannot be used in database.

Q. 13 Explain various security mechanisms in databases.

Q. 14 What are triggers ? Give an example. Illustrate the cases when triggers must not be used.

Q. 15 What is mean by authorization in data base system ? What is granting of privileges ?

8.9 University Questions And Answers

May 2012

Q. 1 What is a view ? How is it created and stored ? *(Ans. : Refer section 8.1)* **(4 Marks)**

Q. 2 Write short notes on : Authorization in SQL *(Ans. : Refer section 8.7)* **(5 Marks)**

Dec. 2012

Q. 3 Explain security in SQL. *(Ans. : Refer section 8.6)* **(5 Marks)**

Q. 4 Explain Authorization in SQL. *(Ans. : Refer section 8.7)* **(5 Marks)**

Dec. 2014

Q. 5 What are triggers ? Explain with example. *(Ans. : Refer section 8.5)* **(10 Marks)**

May 2015

Q. 6 What is the view in SQL, how it is defined ? Discuss the problem that may arise when we attempt to update a view. How views are implemented ? *(Ans. : Refer section 8.1)* **(10 Marks)**

Dec. 2015

Q. 7 Discuss what is meant by each of the following terms : Trigger *(Ans.: Refer section 8.5)*

May 2016

Q. 3(b) What is a view ? How is it created and stored ? *(Ans. : Refer section 8.1)* **(4 Marks)**

May 2017

Q. 1(b) Describe view and Trigger. *(Ans. : Refer sections 8.1 and 8.5)* **(5 Marks)**

□□□

Scanned by CamScanner

Module IV

CHAPTER 9

PLSQL

Course Outcome

Create and populate a RDBMS for a real life application

Retrieve all type of information from a data base

Syllabus Covered

- Procedures in SQL
- Functions in SQL
- Cursors

Programming Tool : Oracle 9i Onwards

Pre-Requisites : Appendix C - PLSQL Programming

Syllabus Topic : Stored Procsdures

9.1 Stored Procedure

- A stored procedure is a named PL/SQL block that can take some parameters (arguments) and produce some output once it is called using stored procedure execution.
- Stored procedures are just text objects, and don't store any data.
- Procedures provides access to data and returns datasets, just like a view.
- Generally speaking you use a procedure to perform a specific action.
- The stored procedure can call data from Table or view.
- **Benefits of stored procedures**
  - Stored procedure offers modularity of code.
  - Procedures promote better reusability and maintainability for code.

  - Once validated, they can be used in 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.

9.2 Creating Stored Procedure

1. A stored procedure has a header, a declaration section, an executable section, and an optional exception-handling section.
2. 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.
3. Stored procedure blocks can start with either BEGIN or the declaration of local variables and end with END statment.
4. We cannot reference any host or bind variables inside a stored procedure.
5. Using REPLACE option if that procedure exists, it will be dropped and replaced with the new procedure created by the statement.

Syntax

```
CREATE [OR REPLACE] PROCEDURE procedure_name
[(parameter1 [mode1] datatype1, parameter2 [mode2] datatype2)]
AS
BEGIN
  PLSQL CODE;
END;
```

Example

Create a procedure to increment salary of all employees by 10%.

```
SQL> CREATE PROCEDURE spIncrSalary
 AS
 Begin
    UPDATE  Emp
    SET     Sal = 1.1*Sal
    WHERE   Job = 'MANAGER';
 End;
```

6. 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).

Scanned by CamScanner

b. From SQL*Plus, run the script file to compile the source code.
c. Use SHOW ERRORS command to see any compilation errors.

7. How to create a stored procedure using oracle SQL command line
   a. Connect to oracle server by providing valid credantials.
   b. Write a valid code for stored procedure using above syntax.
   c. Use SHOW ERRORS command to see any compilation errors.

9.3 Executing Stored Procedure

1. We can execute the stored procedure with help of EXEC command or with help of any application program.
2. Executing stored procedure with Oracle

```
SQL> EXEC spIncrSalary;
```

3. Invoking a procedure from PL/SQL Block or from other procedure.

```
DECLARE
....
BEGIN
  spIncrSalary;          -- Invoke procedure from procedure

END;
```

4. Invoking a procedure from another procedure

```
SQL> CREATE OR REPLACE PROCEDURE access_stud
  IS
  BEGIN
        spIncrSalary;  -- Invoke procedure

  END;
```

9.4 Parameter Types in Stored Procedure

1. Formal parameters

Variables declared in the parameter list of a stored procedure are called as Formal parameters.

Example

```
SQL> CREATE PROCEDURE raise_Fees (p_id NUMBER, p_fees NUMBER)
     Begin
     ...
  END;
```


2. Actual parameters

Variables or expressions referenced in the parameter list of a stored procedure call are nothing but actual parameters.

Example

```
SQL> EXEC raise_fees ('1001', 2000);
```

9.5 Parameter Modes in Stored Procedure

1. IN

- This is default mode for any variable declared in PLSQL to pass value 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.

```
SQL> CREATE PROCEDURE Raise_Fees
 (v_id IN stud.studno%TYPE)
 IS
 BEGIN
     UPDATE  stud
     SET   fees = fees * 1.10
     WHERE   studno = v_id;
 END;
SQL> EXEC Raise_Fees (69);
```

2. OUT

- The OUT mode of variable is declared to returns value to calling environment
- These are not initialized and cannot have a default value assigned to it.

Write a procedure to access student details having given id number.

```
SQL> CREATE 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
```

Scanned by CamScanner

```
    SELECT name, fees, course
  INTO p_name, p_fees, p_course
    FROM stud
  WHERE studno = p_id;
END;
```

 # Executing above stored Procedure

1. Run the above QUERY_EMP procedure.
2. Declare host variables
3. Execute the QUERY_STUD procedure and print global 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.

Example

Write a procedure to increase 10 % fees of student having given id number.

```
SQL >   CREATE 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;
```


Executing above stored procedure,

1. Create a host variable using the VARIABLE command.
2. Then populate this host variable with a value, using a 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.

```
SQL >   VARIABLE g_ mob VARCHAR2(15)
BEGIN
  :g_ mob := '9821666231';
END;
/
SQL >   PRINT g_ mob
SQL >   EXECUTE Format_Mob_Num (:g_ mob)
SQL >   PRINT g_mob
```

9.6 Altering Stored Procedure

1. Introduction

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.

2. Syntax

```
CREATE OR REPLACE PROCEDURE procedure_name
(parameter1 [mode1] datatype1,  parameter2 [mode2] datatype2)]
AS
BEGIN
  PL/SQL Statements;
END;
```

Scanned by CamScanner

Example

Write a procedure to increase 20% fees of student having given id number.

```
SQL> CREATE OR REPLACE PROCEDURE Raise_Fees
  (v_id IN stud.studno%TYPE)
  IS
  BEGIN
        UPDATE  stud
        SET   fees = fees * 1.20
        WHERE   studno = v_id;
  END;
```

9.7 Dropping Stored Procedure

1. Introduction

a. DROP command is used to remove the procedure from database memory.

b. It will remove procedure details from data dictionary also.

2. Syntax

```
DROP PROCEDURE procedure_name
```

3. Example

```
SQL> DROP PROCEDURE Raise_Fees;
```

9.8 Stored Function

1. Introduction

- 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.myFunction
(
  /*
  @parameter1 datatype = default value,
  @parameter2 datatype
  */
```



```
)
RETURNS /* Datatype */
AS
BEGIN
  /* SQL statement ... */
RETURN
  /* value */
END;
```

9.9 Compare stored procedure and 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.

9.10 Concept of Cursor

- 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

9.10.1 Implicit Cursor

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.

Scanned by CamScanner

- 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 then one row or a SELECT INTO statement returned more then 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 then one row or a SELECT INTO statement returned more then 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

Write a PLSQL block to increment salary of all employees above age 25 in company by 5000 and print number of rows updated by above DML operation.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

-- Optional declarative section
SQL> DECLARE
 V_number NUMBER(6);
-- Executable section
BEGIN
  UPDATE Emp
  SET Sal = Sal + 5000
  WHERE age > 25;
  V_number:= SQL%ROWCOUNT;
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;

SQL> /
Salary of 20 Employees updated.
PL/SQL procedure successfully completed.
```

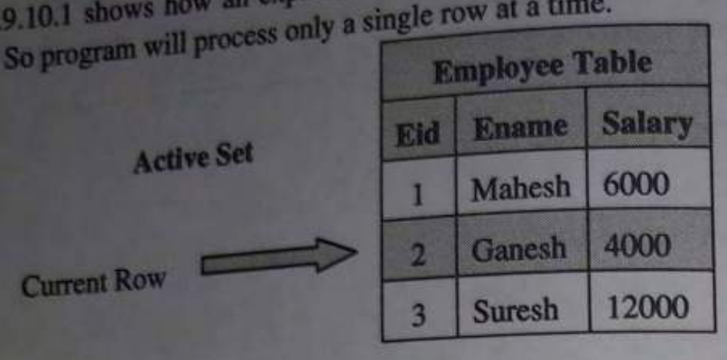
9.10.2 Explicit Cursor

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.

Scanned by CamScanner

- Explicit cursor is declared in the DECLARE section of PL/SQL block.
- The Fig.9.10.1 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.



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.

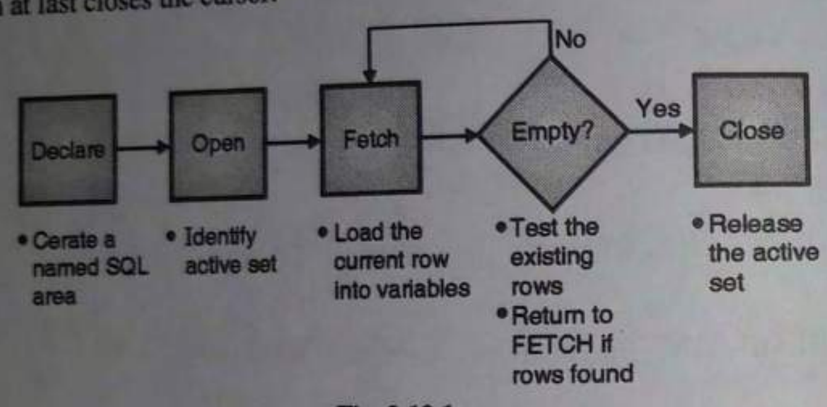


Fig. 9.10.1

a) Declare the cursor

In declare section we defines 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;
```

Scanned by CamScanner

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 then 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 then 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

Write a PLSQL block to print employee name and salary of given employee.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
  CURSOR Emp_Cur
```



```
  IS SELECT Ename, Sal
    FROM  EMP
    WHERE  Empno = &Eid;
  v_ename   emp.ename%TYPE;
  v_salary  emp.sal%TYPE;

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;

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.
```

9.10.3 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.

Scanned by CamScanner

- **The EXIT statement**
  - You can terminate a basic loop using the EXIT statement by checking weather it is last row in table.
  - Flow of control is 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)*

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
     FROM   EMP
     WHERE  age > &Age;
  v_ename   emp.ename%TYPE;
  v_salary  emp.sal%TYPE;

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;

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)

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
DECLARE
```

Scanned by CamScanner

```
 CURSOR Emp_Cur
 IS SELECT Ename,Sal
   FROM   EMP
   WHERE  age > &Age;
 Emp_Rec   Emp_Cur %ROWTYPE;

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;
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.
```


9.10.4 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
```

Where,

Condition - This condition is evaluated at the starting of each iteration; this loop will be repeated while a condition is TRUE.

3) Example

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
    FROM   EMP
    WHERE  age > &Age;
  Emp_Rec   Emp_Cur%ROWTYPE;

BEGIN
  IF NOT Emp_Cur%ISOPEN THEN
    OPEN Emp_Cur;
```

Scanned by CamScanner

```
  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;
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.
```

9.11 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

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
  CURSOR Emp_Cur
  IS SELECT Ename,Sal
     FROM   EMP
     WHERE  age > &Age;
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('-------------------------');
```

Scanned by CamScanner

```
DBMS_OUTPUT.PUT_LINE(Emp_Cur%ROWCOUNT||' rows Selected.');
DBMS_OUTPUT.PUT_LINE('-------------------------');
END;

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

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> 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.
```

Scanned by CamScanner

9.12 Parameterized Cursor

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 PLSQL 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

Write a PLSQL block to print employee name and salary of all employees above given age.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  CURSOR Emp_Cur (p_age NUMBER)
  IS SELECT Ename,Sal
     FROM  EMP
     WHERE  age > p_age;
  Emp_Rec Emp_Cur%ROWTYPE;

 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;

 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.
```

9.13 Cursor Variables

1) Introduction

- Cursor variables are like C pointers, which hold only memory location (address) instead of the item itself. So, declaring a cursor variable creates only a pointer and not an item.
- In PL/SQL, a pointer has datatype as,

  REF X

Scanned by CamScanner

where REF = REFERENCE

X = class of objects.

- Hence, a cursor variable has datatype called as a REF CURSOR.
- We use an explicit cursor which names the work area or we can also use a cursor variable which points to the work area.
- A cursor always refers to the same work area. But a cursor variable can refer to different work areas. So both are operationally different
- We use cursor variables to pass query result sets between various PL/SQL stored subprograms and clients. So they only shears a pointer to the query work area in which the result set is stored.
- A query work area accessible till the time cursor variable points to it. Therefore, we can pass the value of a cursor variable from one scope to another easily.
- Cursor variables are available for use to every client in database environment.
- We can pass cursor variables from application software to the database server using remote procedure calls.

2) Syntax

Define a REF CURSOR type

TYPE **ref_type_name** IS REF CURSOR

[RETURN return_type];

Declare a cursor variable of that type

ref_cv ref_type_name

3) Example

Write a PLSQL block to print employee name and salary of all employees.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> DECLARE
 TYPE Emp_Ref_Cur IS REF CURSOR;
 Emp_Cur Emp_Ref_Cur;
 TYPE rec_emp is record
 (
  name  varchar2(20),
  sal   number(6)
 );
 Emp_Rec rec_emp;
BEGIN
```



```
 OPEN Emp_Cur for select ename,sal from emp;
 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.name,10,' ')||'| '||Emp_Rec.sal);
 END LOOP;
 DBMS_OUTPUT.PUT_LINE('--------------------------');
 CLOSE EMP_CUR;
END;

SQL> /
--------------------------
Name      | salary
--------------------------
SMITH     | 10800
ALLEN     | 11600
WARD      | 11250
JONES     | 12975
MARTIN    | 11250
BLAKE     | 12850
CLARK     | 12450
--------------------------
7 rows Selected.
--------------------------
PL/SQL procedure successfully completed.
```

9.14 Sample Programs

Write a PLSQL block to print employee name and salary and show salary increment of 10% of all employees crossing age of 60.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
```

Scanned by CamScanner

```
SQL> 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;

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
```


Write a PLSQL block to print all employees name with last digit of age is 6.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> 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;
```

Write a PLSQL BLOCK to accept radius of circle from user and insert radius and area in area table.

```
SQL> Create table area (radius Number (7, 2),Area  Number(7,2));
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;

SQL> 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;
```

Scanned by CamScanner

Write implicit cursor to count number of rows updated by update statement.

```
-- The following parameter is required for printing DBMS_OUTPUT.
SQL> SET SERVEROUTPUT ON;

SQL> 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('NO ROWS AFFECTED');
 END IF;
END;
```

Review Questions

Q. 1 Write note on User Define Function.

Q. 2 Write short note on stored procedure

Q. 3 Write short note on implementation of stored procedure

Q. 4 Explain how to pass parameters of stored procedures

Q. 5 Explain how we can change definition of stored procedures

Q. 6 How to drop stored procedures

Q. 7 Explain how we can change definition of stored procedures.

Q. 8 What is PLSQL Cursors ?

Q. 9 Explain various types of cursors with example.

Q. 10 Explain Implicit cursor attributes in details with example.

Q. 11 Explain cursor attributes in details with example.

Q. 12 Explain cursor for loops in details with example.

Q. 13 Write short notes on :

(a) Implicit Cursor

(b) Cursor For Loops

(c) Cursor Loops

Q. 14 What are the parameterized cursors in PLSQL ?

Q. 15 Explain cursor variables in details.

Q. 16 Explain REF Cursors in details.

Q. 17 Crete a functions which returns the total number of records in Employee table.

□□□

Scanned by CamScanner

Module V

Chapter 10 : Relational Database Design 10-1 to 10-34

✓ **Syllabus Topic :** Design guidelines for relational schema 10-1

10.1 Design Guidelines for Relational Schema 10-1

✓ **Syllabus Topic :** Functional Dependencies 10-5

10.2 Functional Dependencies 10-5

10.3 Solved Examples on Functional Dependencies 10-6

10.4 Types of Functional Dependencies 10-9

10.5 FD Properties (Armstrong's Axioms / Closures of FD) 10-12

10.6 Decomposition 10-14

10.7 Keys and Attributes in keys 10-15

10.8 Normalization Process 10-18

✓ **Syllabus Topic :** First Normal Form (1NF) 10-19

10.9 First Normal Form (1NF) 10-19

✓ **Syllabus Topic :** Second Normal Form (2NF) 10-20

10.10 Second Normal Form (2NF) 10-20

✓ **Syllabus Topic :** Third Normal Form (3NF) 10-22

10.11 Third Normal Form (3NF) 10-22

Scanned by CamScanner

✓ **Syllabus Topic : BCNF** 10-25

10.12 Boyce-Codd Normal Form (BCNF) 10-25

✓ **Syllabus Topic : Converting Relational Schema to higher normal forms** 10-28

10.13 Converting Relational Schema to higher normal forms 10-28

10.14 Solved Examples on Normalization 10-30

10.15 University Questions and Answers 10-32

Module VI

Chapter 11 : Storage and Indexing **11-1 to 11-25**

✓ **Syllabus Topic :** Operations on Files 11-1

11.1 Operations on Files 11-1

11.2 Types of Records in Files 11-2

11.3 File Organizations 11-4

11.4 Data Dictionary 11-5

✓ **Syllabus Topic :** Hashing Techniques 11-6

11.5 Hashing Techniques 11-6

11.5.1 Static Hashing 11-7

11.5.2 Dynamic Hashing 11-9

11.5.3 Extendable Hashing 11-9

✓ **Syllabus Topic :** Types of Indexes 11-10

11.6 Index - Concept of Indexing 11-10

✓ **Syllabus Topic :** Single-Level Ordered Indexes 11-12

11.7 Ordered Indices 11-12

11.7.1 Single Level ordered Index 11-12

✓ **Syllabus Topic :** Multilevel Indexes 11-12

11.8 Multilevel Index 11-16

✓ **Syllabus Topic :** Overview of B-Trees and B+-Trees 11-16

11.9 Overview of B Tree and B + Tree 11-17

✓ **Syllabus Topic :** Indexes on Multiple Keys 11-17

- **Appendix A : PLSQL Basics** ..... A-1 to A-20

❑❑❑

Scanned by CamScanner

Module V

CHAPTER 10

Relational Database Design

Course Outcome

After learning this topic students will be able to,

1. Analyze the existing design of a database
2. Apply concepts of normalization to design an optimal database

Syllabus Covered

- Design guidelines for relational schema
- Functional Dependencies
- Definition of Normal Forms - 1NF, 2NF, 3NF, BCNF
- Converting Relational Schema to higher normal forms

Syllabus Topic : Design guidelines for relational schema

10.1 Design Guidelines for Relational Schema

Q. Describe various design guidelines for relational schema.

(1) Introduction

In previous chapters, we have already studied various aspects of the relational model and the languages associated with it. Each relation schema consists of a number of attributes, the relational database schema consists of a number of relation schemas.

Relational Schema (Table Structure)	Relational Database Schema (Database Structure)
$R_1 (A_1, A_2,.., A_n)$ $R_2 (B_1, B_2,.., B_n)$ ... $R_m (C_1, C_2,.., C_n)$	$R_1, R_2, \ldots, R_n$

Where,

$R_1, R_2, \ldots R_n$ = Relation or Tables

A, B, C = Attributes or Columns

- The attributes are grouped to form a relation schema by mapping a conceptual data model i.e. ER or EER data model to relational schema.
- The relational mapping will identify entity types and relationship types and their respective attributes.

(2) Goodness of Relational Design

- In process of database design, we need to develop some measure of goodness for the quality of the design. In this module, we will discuss some measure of goodness to evaluate design quality of relational schemas.
- The above measure will help the developer to analyze why one grouping of attributes is better than another grouping of attributes.
- There are two levels for measuring goodness of relation schemas.

- **Logical (or Conceptual) Level**
  - The goodness of relational schema depends on how users understand the meaning of various attributes of relation.
  - This will helps users to understand the meaning of the data stored in the relations, and hence it will make easy to formulate relational queries correctly.
- **Implementation (or Physical Storage) Level**
  - It helps user to understand how the tuples in a base relation are stored and updated.
  - This level is applied to base relational schema stored as files.

(3) Database Design Approach

The database design can be done using the bottom-up approach to design relational schema using individual attributes of relation or top-down approach to identify individual attributes from relational schema.

- **Bottom-up design methodology / Design by Synthesis**
  - The basic relationships among individual attributes are identified then it will be combined to construct relation schemas.
  - This is not very popular approach as it needs to study many binary relationships among attributes at the beginning which is very difficult.
- **Top-down design methodology / Design by Analysis**
  - The starts with relational schema and it will be further decomposed to a group of attributes to achieve desirable properties.
  - This is very practical approach and used in real world database projects.

Scanned by CamScanner

(4) Guidelines for Relational Schema

To determine the quality of relation schema design some informal guidelines can be used.

- **Guideline 1 : Clear semantics of the attributes in relational schema**
  - Semantics of attribute should be very clear in relational schema so that relational schema will have some real-world meaning associated with it.
  - The relational schema has a clear meaning associated with it.
  - Example,

    Employee (Emp_Id, Ename, Address, Salary)

    The Employee table contains information about all employees in company with their address and Salary.
- **Guideline 2 : Reducing the Redundant Data in Tuples**
  - A relational schema may have some redundancy in database design, if it stores data redundantly.
  - If same data is stored at more then one locations will leads to redundancy and wastage of memory space.
  - **Data Anamolies**: An inconsistent data may cause some problems while adding, updating or deleting data in table which is called as data anomalies.
  - Redundant data is more vulnerable to various data anomalies as if data is updated at only one location and not at other locations, then that data becomes inconsistent, and this problem referred to as an update anomaly.
  - A normalized database stores non-primary key data in only one location.
  - A relational database table should avoid all data anomalies.

Example,

Employee (Emp_Id, Ename, Address)

Emp_Salary (Emp_Id, Ename, payScale, grossSalary, netSalary)

Emp_Designation (Emp_Id, Ename, Desg, fromDate, toDate)

(a) Update Anomaly

- The relational schema may have same data stored in multiple relations, if we update such data from only one relation may result in logical inconsistencies.
- In above example,
- All 3 tables contain the Ename attribute, thus any change in name of one employee will lead to updating his name in all 3 tables. Otherwise, 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 in database.

- An Insert Anomaly arises when certain attributes cannot be added into the database without the presence of other attributes.
- In above example,
- It is not possible to add a row in Emp_Salary table or Emp_Designation table for an employee who is not exists in employee table.

(c) Deletion Anomaly

- If data deleted from one table all relevant data in another related tables must also be deleted otherwise it will create data inconsistency problem.
- Deletion of some data from one relation necessitates the deletion of some other data in other table.
- In above example,
- It is not possible to delete a row in Employee table if Emp_Salary table or Emp_Designation table contains data for respective employee.

- **Guideline 3 : Reducing Null values in Tuples**
  - A value of NULL is different from an empty, White spaces (blank spaces) or zero value.
  - Null values in tuple will cause wastage of memory space and it will also create problem of understanding.
  - Relations should be designed in such a way that their tuples should not contain any NULL values.
  - We can at least try make number of NULL values as low as possible.
  - Attributes with NULL values can be placed in separate relations with the primary key.
  - There are certain resons for Null values,
  - Not applicable data
  - Invalid data
  - Unknown data or data not available
- **Guideline 4 : Dissallowing Spurious Tuples**
  - The bad designs of a relational database may result in erroneous results for some JOIN operation. As it is not possible to get original relation data from new relation after JOIN operation.
  - The relational schema must be designed to satisfy the property of lossless join.
  - If original relation contains fewer number of tuple then tuples generated by doing a natural-join of original relations.

Scanned by CamScanner

Syllabus Topic : Functional Dependencies

10.2 Functional Dependencies

Q. Explain concept of functional dependency.

(1) Introduction

- The concept of functional dependency is given by E. F. Codd.
  - Functional Dependency (FD) provides a constraint between various attributes of a relation.
  - Functional dependencies are restrictions imposed between two set of attributes in relation from a database.
  - In a Relation R with attributes X and Y represented as R(X, Y), where Y is functionally dependent on other column X or we can say X functionally determines Y.
  - This dependency can be denoted with help of arrow (→)

$$X \rightarrow Y$$

  - The data value in column Y must change when data value in another column X is modified.
  - All the attributes before arrow is called as **determinant** and attributes after arrow is called as determine.

(2) Examle

- Consider an employee table with columns as shown in Table 10.2.1

Table 10.2.1 : Employee Table

Employee_Id	Ename	Salary	Project_Id	Hours	Allowance
10	Mahesh	50000	E001	44	40000
12	Suresh	25000	B056	31	30000
15	Ganesh	26000	C671	23	20000
18	Mahesh	50000	E002	12	15000

- **Case 1 : (X→Y)**
  - Consider an Employee table for specific employee_Id there is one and only one Ename whereas for another employee_Id there can be other Ename.

Employee_Id → Ename

  - As per above constraint, it is possible to have multiple employees with same Ename and different Employee_Id. But it is not allowed to have two employees with same Employee_Id and different Ename.
- **Case 2 : (X→YZ)**
  - In above Employee table using below given functional dependency, for specific employee_Id there is one and only one set of Ename and Salary whereas for another employee_Id there can be other values of Ename and salary.

Employee_Id → Ename, Salary

  - As per above constraint, it is possible to have multiple employees with same Ename and Salary.
- **Case 3 : (XY→ZW)**
  - In above Employee table using below given functional dependency, for one employee_Id and Project_Id pair there is only one amount of time spent (Hours) and allowance given by company whereas for another pair there can be other values of Hours and Allowance.

Employee_Id, Project_Id → Hours, Allowance

  - As per above constraint, it is possible to have multiple employee_Id and Project_Id pairs with same values of Hours and Allowance.

10.3 Solved Examples on Functional Dependencies

Example 10.3.1 MU - Dec. 2013, May 2015, 5 Marks

List all functional dependencies satisfied by the relation.

a	b	c
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Solution :

To find out all functional dependencies satisfied by the relation, we must remember one determinant (attributes before arrow) will give one and only one value of determine (attribute after arrow).

A	B	C	Tuple
a_1	b_1	c_1	Tuple 1
a_1	b_1	c_2	Tuple 2
a_2	b_1	c_1	Tuple 3
a_2	b_1	c_3	Tuple 4

Scanned by CamScanner

Step 1 : Test for type 1 FD (X→Y)

Functional Dependency	Violated First by Tuple	Result
A → B	--	Valid FD
A → C	Tuple 2 $a_1 \rightarrow c_1$ $a_1 \rightarrow c_2$	Not Valid FD
B → A	Tuple 3	Not Valid FD
B → C	Tuple 2	Not Valid FD
C → A	Tuple 3	Not Valid FD
C → B	--	Valid FD

Step 2 : Test for type 1 FD (X→YZ)

Functional Dependency	Violated First by Tuple	Result
A → BC	Tuple 2 $a_1 \rightarrow b_1c_1$ $a_1 \rightarrow b_1c_2$	Not Valid FD
B → AC	Tuple 2	Not Valid FD
C → AB	Tuple 3	Not Valid FD

Step 3 : Test for type 1 FD (XY→Y)

Functional Dependency	Violated First by Tuple	Result
AB → C	Tuple 2 $a_1b_1 \rightarrow c_1$ $a_1b_1 \rightarrow c_2$	Not Valid FD
AC → B	--	Valid FD
BC → A	Tuple 3	Not Valid FD

Therfore, all FDs satisfied by relation are,

A → B; C → B; AC → B

Example.10.3.2 MU - May 2014, 10 Marks

Consider the following relation.

A	B	C	Tuple #
10	b_1	c_1	#1
10	b_2	c_2	#2

A	B	C	Tuple #
11	b_4	c_1	#3
12	b_3	c_4	#4
13	b_1	c_1	#5
14	b_3	c_4	#6

Given the previous state which of the following dependencies may hold in the above relation ? If the dependency cannot hold explain why by specifying the tuples that cause the violation :

(i) A →B (ii) B →C (iii) C → B (iv) B → A (v) C → A

Solution :

To find out all functional dependencies satisfied by the relation, we must remember one determinant (attributes before arrow) will give one and only one value of determine (attribute after arrow).

(I) A →B

As in above relation to test A → B, {10} → {b1} but in Tuple # 2 {10} → {b2}

This violates dependency.

Therefore, Dependency cannot hold in above table values due to Tuple #2.

(ii) B → C

As in above relation to test B → C, {b1}→ {c1}, {b2}→{c2}, {b3}→{c4} for all tuples.

Therefore, Dependency holds for all tuples in above table.

(iii) C → B

As in above relation to test A → B, {c1} → {b1} but in Tuple #3 {c1} → {b4}

This violates dependency.

Therefore, Dependency cannot hold in above table values due to Tuple #3.

(iv) B → A

As in above relation to test A → B, {b1} → {10} but in Tuple #5 {b1} → {13}

Also {b3} → {13} but in Tuple #6 {b3} → {14}

This violates dependency.

Therefore, Dependency cannot hold in above table values due to Tuple #5 and Tuple #6.

(v) C→ A

As in above relation to test C→ A, In Tuple #1 {C1} → {10,11,13}

But in Tuple #3 {c1} → {11} and in Tuple #5 {c1} → {13}

This violates dependency.

Therefore, Dependency cannot hold in above table values due to Tuple #3 and Tuple #5.

Scanned by CamScanner

10.4 Types of Functional Dependencies

Q. What are types of functional dependencies ?

(1) Full functional dependency

- A functional dependency is a full functional dependency if removal of any attributes from determinant will invalidate the dependency.
- 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

- Consider an employee table with columns as shown in Table 10.4.1.

Table 10.4.1 : Employee Table

Employee_Id	Ename	Salary	Project_Id	Hours	Allowance
10	Mahesh	50000	E001	44	40000
12	Suresh	25000	B056	31	30000
15	Ganesh	26000	C671	23	20000
18	Mahesh	50000	E002	12	15000

- In the above example, Hours and allowance are fully functionally dependent on both Employee_Id and Project_Id.

Employee_Id, Project_Id → Hours, Allowance

- 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.

(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.
- 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.
- If determine (attributes after arrow) attributes depends on **part of** (partial) determinant attributes. Such dependency is called as partial functional dependency.

Example

Consider an employee table with columns as shown in Table 10.4.2.

Table 10.4.2 : Employee Table

Employee_Id	Ename	Salary	Project_Id	Hours	Allowance
10	Mahesh	50000	E001	44	40000
12	Suresh	25000	B056	31	30000
15	Ganesh	26000	C671	23	20000
18	Mahesh	50000	E002	12	15000

- In the above example, attribute salary is considered to be functionally dependent on both Employee_Id and Project_Id.

Employee_Id, Project_Id → Salary

- But, Attribute salary functionally dependent on Employee_Id is also holds true.

Employee_Id → Salary

- So, Salary is partial functionally dependent on attribute pair Employee_Id and Project_Id.

(3) Transitive dependency

- If one attribute of relation is functionally dependent on other dependent attribute then such a dependency is called as **transitive dependency**.
- 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

- Consider an employee table schema,
- Employee_Id → Department_Id
- Department_Id → Dname

Table 10.4.3 : Employee Table

Employee_Id	Ename	Salary	Department_Id	Dname
10	Mahesh	50000	C1	IT
12	Suresh	25000	E2	HR
15	Ganesh	26000	C1	IT
18	Mahesh	50000	E2	HR

- Dependency of Department_Id on key attribute Dname is transitive functional dependency as Dname is depends on Department_Id which is depend on Employee_Id. So, Dname is transitive functionally dependent on Employee_Id.

Scanned by CamScanner

(3) Trivial functional dependency

- Functional dependency (FD) X → Y and Y is a subset of X, then it is called as a trivial FD.
- Functional dependency X → Y and Y is not a subset of X, then it is called as a non-trivial functional dependency.

Example

- For employee table the below given FD is trivial as Ename is a subset of {Employee_Id, Ename}

Employee_Id, Ename → Ename

- And below given FD is non-trivial as Hours is not a subset of {Employee_Id, Project_Id}

Employee_Id, Project_Id → Hours

(4) Multivalued dependency

- 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.
- **Multivalued dependency** in turn, is defined as relationship which accepts the cross product pattern.

Example

- For employees car table as given below,

Table 10.4.4 : Employee_Car Table

Employee_Id	Ename	Car
10	Mahesh	Ertiga
12	Suresh	Zen
15	Ganesh	Sentro
10	Mahesh	Wagon R

- The FDs are given as below,

Employee_Id →→ Car

- α →→ β says relationship between α and β independent of relationship between α and R-β. That means **Employee_Id →→ Car** relationship is independent of Employee_Id and EName relation. i.e. **Employee_Id →→ Car** is independent of Employee_Id →→ Ename

10.5 FD Properties (Armstrong's Axioms / Closures of FD)

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 Relation R(X, Y, Z, W); represents a table R with set of indivisible attributes X, Y, Z and W.
- It is possible to derive many properties of functional dependencies.
- Axioms are nothing but rules of inference which provides a simple technique for reasoning about functional dependencies.

(1) Primary Properties

a. Subset property (Axiom of Reflexivity)

For given relation R(X, Y, Z, W),

If Y is a subset of X as shown in diagram,

Then **X → Y**

(Which can be referred as X is functionally dependent on Y)

b. Append Property (Axiom of Augmentation)

For given relation R(X, Y, Z, W),

If X → Y

Then XZ → YZ

It is possible to append attribute Z to both sides of FD provided that it is part of sa table.

c. Transitivity (axiom of transitivity)

For given relation R(X, Y, Z, W),

If X → Y and Y → Z

Then X → Z

It is possible to use transitivity if attribute X, Y and Z are part of the same table.

(2) Secondary Properties

a. Union

For given relation R(X, Y, Z, W),

If X → Y and X → Z

Then **X → YZ**

b. Decomposition

For given relation R(X, Y, Z, W),

If X → YZ

Scanned by CamScanner

Then $X \rightarrow Y$ and $X \rightarrow Z$

c. Pseudo Transitivity

For given relation R(X, Y, Z, W),

If $X \rightarrow Y$ and $YZ \rightarrow W$

Then $XZ \rightarrow W$

Example 10.5.1

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) $AC \rightarrow DF$

Solution :

(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 $BC \rightarrow D$ and $BC \rightarrow F$

So using union rule,

$\therefore$ **$BC \rightarrow DF$**

(iii) $AC \rightarrow D$

As $A \rightarrow B$ and $BC \rightarrow D$

So using pseudo transitivity,

$\therefore$ **$AC \rightarrow D$**

(iv) $AC \rightarrow DF$

As $AC \rightarrow D$ and $AC \rightarrow F$

So using union rule,

$\therefore$ **$AC \rightarrow DF$**

Example 10.5.2

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

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

$C \rightarrow D$ $B \rightarrow E$

$AC \rightarrow F$

Calculate some closures as $\{A\}^+$, $\{B\}^+$, $\{AC\}^+$ and also find key of above relation.

Solution :

(i) $A \rightarrow B, A \rightarrow C$ $\{A\}^+ = \{A, B, C\}$

So using union rule,

$A \rightarrow BC$

(ii) $B \rightarrow E$ $\{B\}^+ = \{B, E\}$

(iii) $\{A\}^+ = \{A, B, C\}$ and $\{AC\}^+ = \{A, B, C, D, E, F\}$

$C \rightarrow D, B \rightarrow E$ and $AC \rightarrow F$

(iv) $\{AC\}^+ = \{A, B, C, D, E, F\}$ $\{AC\}$ can determine all attributes in relation R

So, $\{AC\}$ is a key of Relation R

10.6 Decomposition

Q. What is decomposition ? Explain various rules of normalization.

1. Introduction

(a) 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.

(b) This is a process of dividing one table into multiple tables can be done using projection operator.

(b) Decomposed table can be reconstructed using join operation.

2. Desirable Properties or Goals

Q. What are goals of decomposition ?

Q. What are goals of normalization ?

(a) Lossless join decomposition

- The original relation and relation reconstructed from joining decomposed relations must contain same number of tuples if number is increased or decreased then it is Losssy Join decomposition.
- 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.

Example,

- Employee (Employee_Id, Ename, Salary, Department_Id, Dname)
- Can be decomposed using lossless decomposition as,
- Employee_desc (Employee_Id, Ename, Salary, Department_Id)
- Department_desc (Department_Id, Dname)
- Alternatively the lossy decomposition would be as joining these tables is not possible so not possible to get back original data.
- Employee_desc (Employee_Id, Ename, Salary)
- Department_desc (Department_Id, Dname)
- Rules for lossless decompositions are,

(i) The relations to be decomposed must have at least one common attribute in pair of relations.

i.e. $R_1 \cap R_2 \neq \Phi$

Scanned by CamScanner

(ii) The attributes in common must be a key for one of the relation for decomposition to be lossless.

(b) Dependency preservation

- All functional dependencies result in just one relation.
- 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.

Example,

- Consider relation R(X, Y, Z, W) that has the following dependencies F,

$$X \rightarrow Y$$
$$Y \rightarrow ZW$$

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

(c) No repetition of information

- Decomposition that we have done should not suffer from any repetition of information problem.
- It is desirable not to have any redundancy in database.

10.7 Keys and Attributes in keys

Q. Write a note on : Candidate Key, Secondary Key and Super Key.

(1) Introduction

- Normalization process will makes use of some types of keys to remove the redundancies present in data of relational tables.
- 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**.

- The non-normalized relations may causes redundancy problems which leads to data anomalies.

(2) Super Key

Q. Write a note on : Super Key.

- A **superkey** of a relation is a set of attributes $S \subseteq R$ with the property that no two tuples in relation will have same combination of attribute values in S.
- In other words, a relation will have unique set of attributes S.
- If we add additional attributes to above set of attributes S, the resulting combination would still uniquely identifies a single record in a table. Such augmented keys are also called as **superkey**.

Example,

Consider a Relation R (A, B, C, D, E, F) with Functional dependencies,

$A \rightarrow BC$

$AC \rightarrow DE$

$E \rightarrow F$

To find out key of relation R,

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

$\{AC\}^+ = \{A, B, C, D, E, F\}$

$\{ACD\}^+ = \{A, B, C, D, E, F\}$

So {A}, {AC}, {ACD} are all superkey.

Example,

- STUDENT (<u>Id, Name</u>, Class, Branch, Age, Address, Mobile)
- The ID is a key attribute of STUDENT table, so ID and Name can be a superkey.

(3) Candidates Key

Q. Write a note on : Candidate 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 candidates 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.

Scanned by CamScanner

Example,

Consider a Relation R (A, B, C, D, E, F) with Functional dependencies,

$A \rightarrow BC$

$AC \rightarrow DE$

$E \rightarrow F$

To find out key of relation R,

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

$\{AC\}^+ = \{A, B, C, D, E, F\}$

$\{ACD\}^+ = \{A, B, C, D, E, F\}$

So {A}, {AC}, {ACD} are all superkey and {A} is a candidates key.

Example,

STUDENT (Id, Name, Class, Branch, Age, Address, Mobile)

The ID is a candidate key attribute of STUDENT table.

(4) Secondary Key

Q. Write a note on : Secondary Key.

- A candidate key selected to uniquely identify all other attribute values in any given row.
- If a number of candidate keys in a relation schema then one is arbitrarily selected as **primary key** and other keys are called as **secondary keys**.
- Secondary key of a table is a column or combination of some columns used for data retrieval process.

Example,

Consider a Relation R (A, B, C, D, E, F) with Functional dependencies,

$AB \rightarrow C, BC \rightarrow DE, E \rightarrow AF$

To find out key of relation R,

$\{AB\}^+ = \{A, B, C, D, E, F\}$

$\{BC\}^+ = \{A, B, C, D, E, F\}$

So {AB} and {BC} are all candidates key.

If {AB} is selected as primary key then {BC} is called as secondary key.

(5) Prime Attributes

Q. Write a note on : Prime Attributes.

- An attribute in a relation schema R is called as prime attribute, if it is a member of any of the candidate key present in a relation.

- If an attribute is not a member of any candidate key then it is called as nonprime attribute.

Example

- Consider an Employee table with following FDs,
- Employee_Id → Ename, Salary
- Employee_Id, Project_Id → Hours, Allowance

Employee_Id	Ename	Salary	Project_Id	Hours	Allowance
10	Mahesh	50000	E001	44	40000
12	Suresh	25000	B056	31	30000
15	Ganesh	26000	C671	23	20000
18	Mahesh	50000	E002	12	15000

In above table candidate key is {Employee_Id, Project_Id}

- **Prime Attributes :** Employee_Id, Project_Id
- **Non-Prime Attributes :** Ename, Salary, Hours, Allowance

10.8 Normalization Process

MU - May 2016

University Question

Q. What is Normalization ? **(May 2016, 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.
- Normalization is a step by step decomposition and database designers may not normalize relation to the highest possible normal form.
- The relations may be left in a lower normal form like 2NF, which may cause some penalties like data anomalies.

(2) Definition

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

(3) Goals of Database Normalization

1. Ensures data integrity.

Scanned by CamScanner

2. Prevents redundancy in data.
3. To avoid data anomaly
   (a) Update anomaly
   (b) An insertion anomaly
   (c) Deletion anomaly

Syllabus Topic : First Normal Form (1NF)

10.9 First Normal Form (1NF)

MU - May 2016

University Question

Q. Explain 1NF. **(May 2016, 2 Marks)**

(1) Introduction

- This is 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 all attributes in relation must have atomic (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,
- A columns in a table should contain only indivisible data.

(3) Example

- Consider an employee table with columns as shown in diagram,
  - The relational schema not in 1 NF is represented as,

Employee Table

Employee_Id	Ename	Salary	Ecity

- The state of Employee relational schema is as given below and it contains the Ecity which is non atomic (divisible) domain.

Table 10.9.1 (Non-Normalised Employee Table

Employee_Id	Ename	Salary	Ecity
10	Mahesh	50000	Mumbai, Pune
12	Suresh	25000	Mumbai
15	Ganesh	26000	Pune
18	Kasturi	50000	Mumbai, Delhi

- To convert relational schema in 1NF, the Ecity attribute is divided in atomic domains it may introduce some data redundancy.

Fig. 10.9.2 1NF Employee Table

Employee_Id	Ename	Salary	Ecity
10	Mahesh	50000	Mumbai
10	Mahesh	50000	Pune
12	Suresh	25000	Mumbai
15	Ganesh	26000	Pune
18	Kasturi	50000	Mumbai
18	Kasturi	50000	Delhi

(4) Minimizing Domain Redundancy

- The first normal for will solve the group redundancy occurs in domain value as it allows only a single value from the domain of that attribute.
- So, 1NF will solve all problems related to domain redundancy.
- Nested relations must be removed to convert relation in 1 NF.

Syllabus Topic : Second Normal Form (2NF)

10.10 Second Normal Form (2NF)

MU - May 2016

University Question

Q. Explain 2NF. **(May 2016, 2 Marks)**

(1) Introduction

- This normal form makes use of full functional dependency and tries to remove problem of redundant data introduced by 1NF decomposition.
- This normal form is given by E.F. Codd in 1971.

(2) Definition

- A relation is in 2NF, if it is in 1NF and all non-key attributes in relation are 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 complete primary key of relation (and not depends on part of (partial) primary key).
- In short 2NF means,

Scanned by CamScanner

- It should be in 1NF.
- There should not be any partial dependency on primary key attributes.

(3) Example

- Consider an employee table with columns as shown in diagram,
- The relational schema **not in 2 NF** is represented as,

Consider an Employee table with following FDs,

Employee_Id → Ename, Salary

Employee_Id, Project_Id → Hours, Allowance

As

{Employee_Id, Project_Id} → Ename, Salary, Hours, Allowance

Therefore,

Candidate key {Employee_Id, Project_Id} is selected as primary key.

- As attributes Hours, Allowance of employee table are full functionally dependent on primary key whereas attributes Ename and Salary are partially depends on primary key. (As Ename, Salary are depends **on part of primary key**)
- The state of Employee relational schema is,

Table 10.10.1 : Non-2NF Employee Table

Employee_Id	Ename	Salary	Project_Id	Hours	Allowance
10	Mahesh	50000	E001	44	40000
12	Suresh	25000	B056	31	30000
15	Ganesh	26000	C671	23	20000
18	Mahesh	50000	E002	12	15000
15	Ganesh	26000	E001	24	20000
18	Mahesh	50000	B056	11	10000

- To normalize above schema to 2NF we can decompose tables as ,
- Employee (Employee_Id, Ename, Salary)
- Employee_Id → Ename, Salary

Table 10.10.2 : 2NF Employee Table

Employee_Id	Ename	Salary
10	Mahesh	50000
12	Suresh	25000

Employee_Id	Ename	Salary
15	Ganesh	26000
18	Mahesh	50000

- Project (Employee_Id, Project_Id, Hours, Allowance)
- Employee_Id, Project_Id → Hours, Allowance

Table 10.10.3 : 2NF Project Table

Employee_Id	Project_Id	Hours	Allowance
10	E001	44	40000
12	B056	31	30000
15	C671	23	20000
18	E002	12	15000
15	E001	24	20000
18	B056	11	10000

- Consider, Relation R(A, B, C, D, E, F) and the FDs as below,

A → BC , B → DC, D → EF

(i) The candidate Key is {AD} → {A, D, B, C, E, F} selected as primary key.

All attributes are partially dependent on primary key.

Hence, Relation R is **not in 2NF**.

(ii) The 2NF Relation Schema is,

R1 (A, B, C, D) with FDs A → BC, B → DC

R2 (D, E, F) with FDs D → EF

(3) Minimizing Tuple Redundancy

- The second normal form will avoid same tuples to be repeated in a table as it forces all non-key attributes must be full functionally depends on primary key of a relation.
- 2NF will create a new table for each partial key with all its dependent attributes.

Syllabus Topic : Third Normal Form (3NF)

MU - May 2016

10.11 Third Normal Form (3NF)

University Question (May 2016, 2 Marks)

Q. Explain 3NF.

Scanned by CamScanner

(1) Introduction

- This normal form is given by E.F. Codd in 1971.
- This normal form introduced to minimize the transitive redundancy.
- To remove the data anomalies left in relational schema even after applying second normal form like transitive dependencies.

(2) Definition

- A relation is in 3NF, if it is in 2NF and all non-prime attributes of the relation are non-transitively dependent on the every key.
- A relation R is in 3NF if all non prime attributes are,
  1. Full functionally dependent on primary key.
  2. Non-transitive dependent on every key.
- A relational schema R is in 3NF, if non-trivial functional dependency X→A holds true where X is a superkey and A is a prime attribute.

(3) Example

- Consider an employee table with columns as shown in diagram,
- The relational schema **not in 2 NF** is represented as,
- Consider an Employee table with following FDs,
- Employee_Id → Ename, Salary, Department_Id
- Department_Id → Dname
- The state of Employee relational is,

Table 10.11.1 : Employee Table

Employee_Id	Ename	Salary	Department_Id	Dname
10	Mahesh	50000	C1	IT
12	Suresh	25000	E2	HR
15	Ganesh	26000	C1	IT
18	Mahesh	50000	E2	HR

{Employee_Id} → Ename, Salary, Department_Id, Dname

Therefore,

Candidate key {Employee_Id} is selected as primary key.

- As all attributes in employee table are full functionally dependent on primary key. Therefore, **Relation R is in 2NF.**
- Non-prime attributes Ename, Salary, Department_Id are non-transitively dependent on primary key. But, Dname attribute is transitively dependent on key. Therefore, **Relation R is not in 3NF.**

Employee_Id → Department_Id

Department_Id → Dname

- To normalize above schema to 3NF we can decompose tables as ,

Employee (Employee_Id, Ename, Salary, Department_Id)

Employee_Id → Ename, Salary, Department_Id

Table 10.11.2 3NF Employee Table

Employee_Id	Ename	Salary	Department_id
10	Mahesh	50000	C1
12	Suresh	25000	E2
15	Ganesh	26000	C1
18	Mahesh	50000	E2

Department (Department_Id, Dname)

Employee_Id → Dname

Table 10.11.3 : 3NF Department Table

Department_Id	Dname
C1	IT
E2	HR

- Consider, Relation R(A, B, C, D, E, F) and the FDs as below,

  A → BC , B → D, D → EF

(i) The candidate Key is {A} → {A, D, B, C, E, F} selected as primary key.
- All attributes are full functionally dependent on primary key.
- Hence, Relation R is **in 2NF**.
- But, non-prime attributes B, D, E and F are transitively depends on key.
- So, Relation R is **not in 3NF**.

(ii) The 3NF Relation Schema is,
- R1 (A, B, C) with FDs A → BC
- R2 (B, D) with FDs B → D
- R3 (D, E, F) with FDs D → EF

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

(4) Minimizing Group Redundancy

- The third normal form will avoid repeating groups in same table as it forces all non-prime attributes must be non-transitively depends on key of a relation.

Scanned by CamScanner

- 3NF will create a new table for each transitive attribute and its dependent attributes.

Table 10.11.4 : Summary of normal form based on primary keys and corresponding normalization

Normal form	Test	Remedy (normalization)
First (1NF)	Relation should have to multivalued attributes or nested relations.	Form new relations for each multivalued attributed to nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attributed should be functionally.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original promarykey and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should bot have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determinte(s) other nonkey attribute(s).

Syllabus Topic : BCNF

10.12 Boyce-Codd Normal Form (BCNF)

MU - Dec. 2015, Dec. 2016

University Question

Q. Describe BCNF in detail. **(Dec. 2015, Dec. 2016, 5 Marks)**

(1) Introduction

- This normal form is governed by Raymond F. Boyce and E.F. Codd in 1974.
- BCNF is more rigorous form of 3NF and every relation in BCNF is always in 3NF.
- The intention of Boyce-Codd Normal Form (BCNF) is that 3NF does not satisfactorily handle the case of overlapping candidate keys.
- If transitivity is present in prime attributes of relation may not be removed by 3NF.

(2) Definition

- A relation R is said to be in BCNF, if and only if every determinant is a candidate key.
- A relational schema is in BCNF, if a non-trivial functional dependency X→A is true then X is a superkey of relation R.
- In 3NF definition A should be prime attribute, which is not the case in BCNF definition.

(3) Example

- Consider an employee table in which employee can work in more than one department,
- The relational schema **not in 2 NF** is represented as,
- Consider an Employee table with following FDs,
- Employee_Id → Ename, Salary, Department_Id
- Department_Id → Dname
- The state of Employee relational is,

Table 10.12.1 : Employee Table

Employee_Id	Ename	Department_Id	Dname	Dtype
10	Mahesh	C1	IT	Technical
12	Ganesh	E2	HR	Skill
12	Ganesh	C1	IT	Technical
10	Mahesh	E2	HR	Skill
13	Satish	E1	TS	Technical

Employee_Id → Ename

Department_Id → Dname, Dtype

Therefore,

- Candidate key {Employee_Id, Department_Id} is selected as primary key.
- As no attribute in employee table is full functionally dependent on primary key. Therefore, **Relation R is not in 2NF.** All non-prime attribute are non-transitively dependent on primary key. Therefore, **Relation R is in 3NF.**
- To normalize above schema to **BCNF** we can decompose tables as ,
- **Employee** (Employee_Id, Ename)
- Employee_Id → Ename
- The determinant Employee_Id is candidate key.

Scanned by CamScanner

Table 10.12.2 : 3NF Employee Table

Employee_Id	Ename
10	Mahesh
12	Ganesh
13	Satish

- **Department** (Department_Id, Dname, Dtype)
- Department_Id → Dname, Dtype
- The determinant Department_Id is candidate key.

Table 10.12.3 : 3NF Department Table

Department_Id	Dname	Dtype
C1	IT	Technical
E2	HR	Skill
E1	TS	Technical

- **Emp_Dept** (Employee_Id, Department_Id)

Table 10.12.4 : 3NF Emp Dept Table

Employee_Id	Department_Id
10	C1
12	E2
12	C1
10	E2
13	E1

(4) Minimizing Key Transitivity (Key redundancy)

- The BCNF will produces a table for each functional dependency to make all determinants as key of relation.
- BCNF will create a new table for each transitive attribute and its dependent attributes.

Example 10.12.1 MU - Dec. 2014,10 Marks

Consider the following relation,

CAR-SALE (Car#, Date-sold, Salesman#, commission%, Discount-amt)

Assume that {Car#, Salesman#} is the primary key.

Additional dependencies are,

Date-sold → Discount_amt

Salesman# → commission%

Based on the given primary key, is this relation in 1NF, 2NF or 3NF ?

Why or why not ?

How would you successively normalize it completely ?

Solution :

CAR-SALE (Car#, Salesman#, Date-sold, commission%, Discount-amt)

Assuming {Car#, Salesman#} is the primary key

Therefore,

Car#, Salesman# → Date-sold, commission%, Discount-amt

Additinally,

Date-sold → Discount_amt

Salesman# → commission%

- The above relation **is in 1NF as** all attributes of relation are atomic domains.
- The above relation **is in 2NF as** primary key is assumed and all non key attributes are full functionally depends on primary key.
- The above relation **is in 3NF as** all non prime attributes are non-transitively depending on primary key.
- Noramlized Relation

  CAR-SALE (Car#, Salesman#, Date-sold, commission%, Discount-amt)

  Car#, Salesman# → Date-sold, commission%, Discount-amt

  Date-sold → Discount_amt

  Salesman# → commission%

Syllabus Topic : Converting Relational Schema to higher normal forms

10.13 Converting Relational Schema to higher normal forms

Example 10.13.1

Relation R(A, B, C, D, E, F, G, H, I, J). Having following set of FD, show convert table to highest normal form.

AB → C, C → EF, AD → GH, G → I, H → J

Solution :

(a) 1NF

Assuming all attributes are atomic domains so relation R is in 1NF.

(b) 2 NF

$\{A, B, D\}^+ \rightarrow \{\underline{A, B, D}, C, E, F, G, H, I, J\}$

{A, B, D} is candidate key for relation R.

No attribute in relation is full functionally dependent on above key.

Therefore, Relation R is not in 2NF.

Scanned by CamScanner

Decomposition to 2NF

R_1 (A, B, C, E, F); AB → C, C → EF

R_2 (A, D, C, E, F); AD → GH, G → I, H → J

(c) 3NF

In relation R1, non-prime attributes E, F are transitively depends on key. So, relation is not in 3NF.

The relation R1 in 3NF can be written as,

R_{1a} (A, B, C); AB → C

R_{1a} (C, E, F); C → EF

In relation R2, non-prime attributes I, J are transitively depends on key. So, relation is not in 3NF.

The relation R1 in 3NF can be written as,

R_{2a} (A, D, G, H); AD → GH

R_{2b} (G, I); G → I

R_{2c} (H, J); H → J

(c) BCNF

Relation	FDs	Determinant	Key	BCNF?
R_{1a} (A, B, C)	AB → C	AB	AB	Yes
R_{1a} (C, E, F)	C → EF	C	C	Yes
R_{2a} (A, D, G, H)	AD → GH	AD	AD	Yes
R_{2b} (G, I)	G → I	G	G	Yes
R_{2c} (H, J)	H → J	H	H	Yes

So, relational schema in BCNF is as given below,

R_{1a} (A, B, C); AB → C

R_{1a} (C, E, F); C → EF

R_{2a} (A, D, G, H); AD → GH

R_{2b} (G, I); G → I

R_{2c} (H, J); H → J

Example 10.13.2

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

Solution :

1. $\{A\}^+$

(a) Axiom of Pseudo transitivity

$$A \rightarrow BC \text{ and } B \rightarrow E \quad \ldots(1)$$

$$\therefore A \rightarrow EC \quad \ldots(2)$$

(c) Decomposition from Equations (1) and (2)

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

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

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

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

10.14 Solved Examples on Normalization

Example 10.14.1 MU - Dec. 2013, 10 Marks

Consider a dependency diagram of relation R and normalize it up to third normal form.

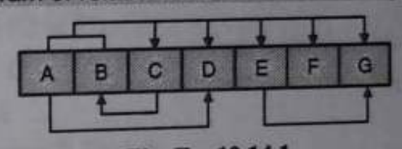


Fig. Ex. 10.14.1

Solution :

Noramlized Relation R (A, B, C, D, E, F, G) with set of FDs

AB → CDEFG

C → B

A → D

E → G

Example 10.14.2 MU - Dec.2013, 10 Marks

Consider a dependency diagram of relation R and normalize it upto third normal form.

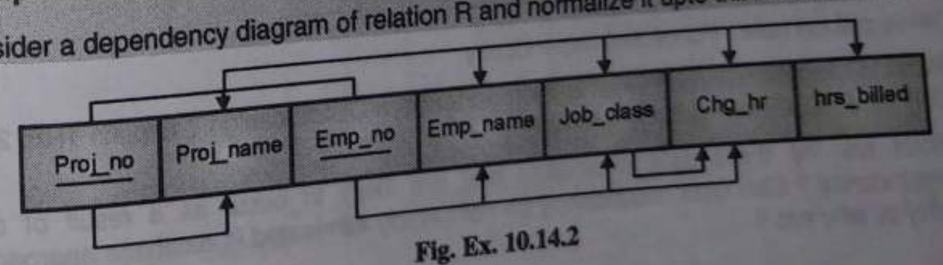


Fig. Ex. 10.14.2

Scanned by CamScanner

Solution :

- Noramlized Relation,
- Employees (Proj_no, Emp_no, Proj_name, Emp_name, Job_Class, Chg_Hr, Hrs_Billed)
- With set of FDs

Proj_no, Emp_no → Proj_name, Emp_name, Job_Class, Chg_Hr, Hrs_Billed

Emp_no → Emp_name, Job_Class, Chg_Hr

Proj_no → Proj_name

Job_Class → Chg_Hr

Example 10.14.3 MU - May 2017, 10 Marks

Consider the following dependency diagram of relation R and normalize till 3NF form.

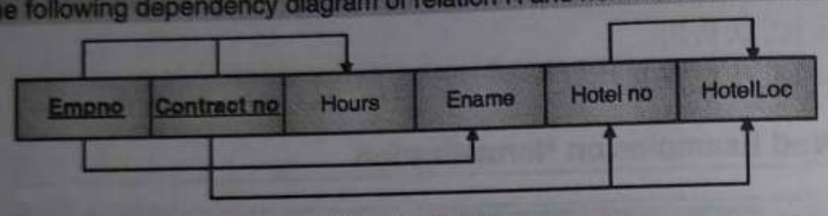


Fig. Ex. 10.14.3

Solution :

Converting dependency diagram to dependencies,

Empno,Contactno -> Hours

Empno ->Ename

Contactno ->Hotelno,HotelLoc

Hotelno->HotelLoc

3NF schema is as given below,

Emp(Empno,Contactno,Hours); Empno,Contactno -> Hours

Emp_Details(Empno,Ename); Empno ->Ename

Hotel_Contact(Contactno,Hotelno,HotelLoc); Contactno ->Hotelno,HotelLoc

Hotel(Hotelno,HotelLoc); Hotelno ->HotelLoc

Review Questions

Q. 1 Write a short note : Functional dependency

Q. 2 What is nformalizations ? What is its importance in DBMS design ? Explain 1NF, 2NF, 3NF and BCNF with suitable example.

Q. 3 What are the three data anomalies that are likely to occur as a result of data redundancy ? Can data redundancy be completely eliminated in database approach ? Why or why not ?

Q. 4 What is closure set of functional dependency ?

Q. 5 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 → EC, D → EF

Q. 6 Describe data redundancy in relational schema.

Q. 7 Distinguish between functional dependency and multivalued dependency.

Q. 8 List the Armstrong's axioms for functional dependencies ?

Q. 9 Why BCNF is more desirable than 3 NF ?

Q. 10 State and compare 3 NF and BCNF.

Q. 11 Suppose we have a relation ABCD with some FD's F as follows :
F = AB → C, C → D, D → A
Compare closure A+, C+, (AB)+, (AC)+ and all FD'S (functional dependencies) that follow from F.

Q. 12 Differentiate between full functional dependency and partial functional dependency.

Q. 13 What are the various anomalies associated with RDBMS ?

Q. 14 Define 3NF. What are the differences between 3NF and BCNF ?

Q. 15 Give two sets F1 and F2 of FDs for a relation (A, B, C, D, E)
F1 : A → B, AB → C, D → AC, D → E
F2 : A → BC, D → AE
Are F1 and F2 equivalent ? Explain.

Q. 16 Consider the relation R = (A, B, C, D, E, F, G, H) with following FDs :
F = {AC → G, D → EG, BC → D, CG → BD, ACD → B, CE → AG}
Find the canonical cover and minimal of F.

10.15 University Questions and Answers

Dec. 2013

Q. 1 List all functional dependencies satisfied by the relation.

a	b	c
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

(10 Marks)

(Ans.: Refer Example 10.3.1)

Scanned by CamScanner

Q. 2 Consider a dependency diagram of relation R and normalize it up to third normal form. **(10 Marks)**
(Ans.: Refer Example 10.14.1)

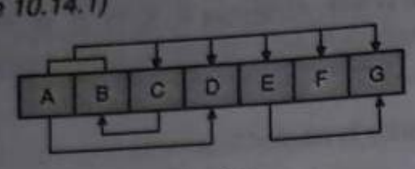


Fig. Q.1

Q. 3 Consider a dependency diagram of relation R and normalize it upto third normal form. **(10 Marks)**
(Ans.: Refer Example 10.14.2)

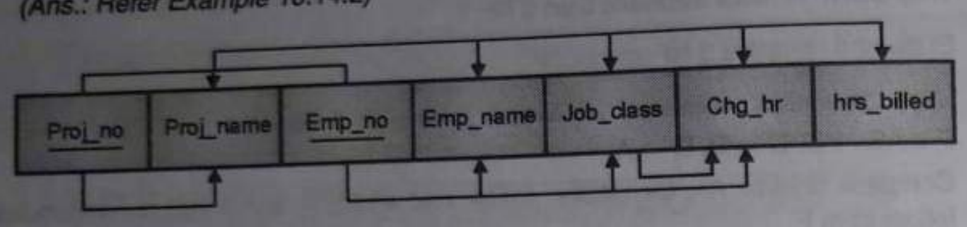


Fig. Q. 2

Dec. 2014

Q. 4 Consider the following relation,
CAR-SALE (Car#, Date-sold, Salesman#, commission%, Discount-amt)
Assume that {Car#, Salesman#} is the primary key.
Additional dependencies are,
Date-sold → Discount_amt
Salesman# → commission%
Based on the given primary key, is this relation in 1NF, 2NF or 3NF ?
Why or why not ?
How would you successively normalize it completely ?
(Ans.: Refer Example 10.12.1) **(10 Marks)**

May 2014

Q. 5 Consider the following relation.

A	B	C	Tuple #
10	b_1	c_1	#1
10	b_2	c_2	#2
11	b_4	c_1	#3
12	b_3	c_4	#4
13	b_1	c_1	#5
14	b_3	c_4	#6

Given the previous state which of the following dependencies may hold in the above relation ? If the dependency cannot hold explain why by specifying the tuples that cause the violation :
(i) A →B (ii) B →C (iii) C → B (iv) B → A (v) C → A
(Ans. : Refer Example.10.3.2) **(10 Marks)**

May 2015

Q. 6 List all functional dependencies satisfied by the relation.
(Ans. : Refer Example 10.3.1) **(5 Marks)**

a	b	c
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Dec. 2015

Q. 7 Describe BCNF in detail. *(Ans. : Refer section 10.12)* **(5 Marks)**

May 2016

Q. 5(a) Define normalization ? Explain 1NF, 2NF and 3NF with example.
(Ans. : Refer Sections 10.8,10.9,10.10 and 10.11) **(10 Marks)**

Dec.2016

Q. 4(a) Describe BCNF in detail. *(Ans. : Refer Section 10.12)* **(5 Marks)**

May 2017

Q.4(a) Consider the following dependency diagram of relation R and normalize till 3NF form.
(Ans. : Refer Example 10.12.3)

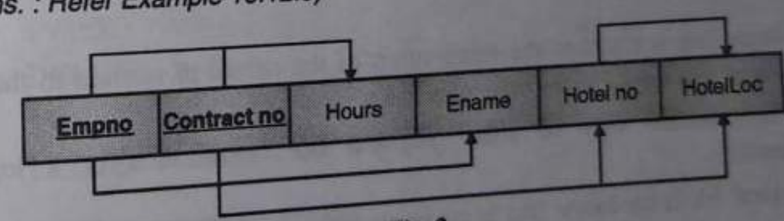


Fig. 3

□□□

Scanned by CamScanner

Module VI

CHAPTER 11

Storage and Indexing

Course Outcome

After learning this topic students will be able to,

1. Build indexing mechanisms for efficient retrieval of information from a database

Syllabus Covered

- Operation on Files
- Hashing Techniques
- Types of Indexes : Single-Level Ordered Indexes; Multilevel Indexes
- Overview of B-Trees and B+-Trees
- Indexes on Multiple Keys

Syllabus Topic : Operations on Files

11.1 Operations on Files

Q. Describe various operations on file.

1. Introduction

- File organization refers to the relationship of the record in relation to the physical location of that record in the computer file.
- File organization may be either physical file organisation or a logical file organisation.
- A physical file is hardware, like as magnetic tape or a disk.

2. Goals

- To give ease of creation and maintenance of database in terms of file organisation.
- To create an efficient way of storing and retrieving information from file system.

3. Overview

- A logical file is a complete set of records for a specific purpose or designated to specific application. A logical file can be treated as a part of physical file.
- In case of file organisation, database is stored in form of a collection of files.
- Each file is organized logically as a sequence of multiple records.
- A record is a sequence of fields (columns) in a relation (table).
- Records are mapped onto disk blocks for its storage.
- Size of such records on file system may vary as per requirements.
- In relational database, tuples of different relations may be of different sizes.
- One approach to mapping database to files is to store records of one length in a given file called as fixed length records.
- An alternative approach is variable-length records. But, Fixed-length approach is easier to implement then variable length.

4. Types of file organisation

- Heap based file organization
- Sequential file organization
- Index file organization
- Hashing file organization
- Clustering file organization

5. File Operations

- Operations are classified into two categories
  - a) **OPEN :** Reads the file for access
  - b) **FIND :** Searches for the first file record that satisfies a given condition and makes that record as current file record
  - c) **READ :** Reads the current file record into a variable
  - d) **INSERT :** Adds a new record into the file
  - e) **DELETE :** Removes the current record from the file
  - f) **MODIFY :** Changes the values of some fields in the current file record
  - g) **CLOSE :** Terminates access to the file

11.2 Types of Records in Files

Q. Describe various types of records in files.

1. Introduction

- Data in file organization is usually stored in the form of records.

Scanned by CamScanner

- Records contain fields which have values of a particular entry in database.
- There are various types of records,
- Fixed Length Records
- Variable length Records

2. Fixed Length Record

- Consider a file of student records of the form :

```
Type t_student = record
    Student_name : char(22);
    Sid : char(10);
    fees : real;
end
```

- If we assume that each character occupies one byte, an integer occupies 4 bytes, and a real 8 bytes, our deposit record is 40 bytes long.
- The simplest approach is to use the first 40 bytes for the first record, the next 40 bytes for the next record, and so on.

Problems

- It is difficult to delete a record from such fix structure.
- Block size should be multiple of 40. It would then require two block accesses to read or write a record which is more then size 40.

3. Variable Length Record

- Variable length records are required in system if Storage of multiple record types in a file, Record types that allow variable lengths for a single or multiple fields.
- It can be implemented as follows,

1. Byte string representation of variable length record.

SEIT	IT-101	45000	IT-201	40900	IT-301	70000	⊥
SECE	IT-102	35000	⊥				
TEIT	IT-103	42000	⊥				
ITCE	IT-104	47000	IT-204	45000	⊥		
BEIT	IT-105	43000	⊥				
BECE	IT-106	41000	⊥				

- In method above attach a special end of record (⊥) symbol to the end of each record.
- In other method we store the record length at the beginning of each record.

11.3 File Organizations

Q. Write a note on : File Organization.

1. Heap file organization

- Any record can be placed anywhere in the file where there is space for that record.
- There is no proper ordering of records in a file. This is file organization can be also called as 'pile file'.
- In this file organisation each new record is inserted at the end of the file.
- To search any record, a linear (sequential) search through the file is necessary.
- This requires reading and searching all file blocks in worst case, and is hence quite expensive.
- In this file organisation insertion of record is quite efficient.
- Sorting the file is required for Reading the records in order of a particular field.

2. Sequential file organization

- In A sequential file records are chained together with help of pointers to permit faster access in order of search key value.
- This file organisation is having efficient processing of records in sorted order based on some search key.
- Pointer points to next consecutive record in order inside file.
- Records are physically located in order of search key value.
- Deletion can be done with help of pointers put new record in an overflow block.

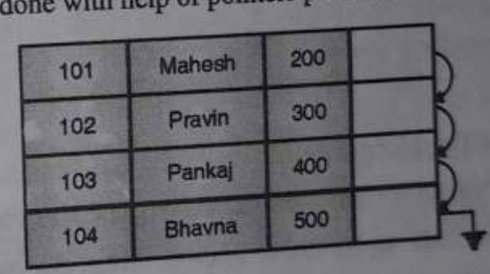


Fig. 11.3.1

3. Hashing file organization

- A hash function is computed with help of some attribute of every record. This hash function is used to identify location of record inside the file.
- Hashing technique done for disk files is also called External Hashing
- Method of hashing
- The file blocks are divided into M equal-sized *buckets* (0 to M-1). Each bucket corresponds to one disk block.

Scanned by CamScanner

- One of the file fields is called to be as hash key of the file.
- The record with hash key value K is stored in bucket number i,

  $i = h(K)$, Where h = *hash function*
- Search is very efficient using the hash key.
- Collisions can happen when a new record hashes to a bucket which is already full. Such records are added to overflow bucket.

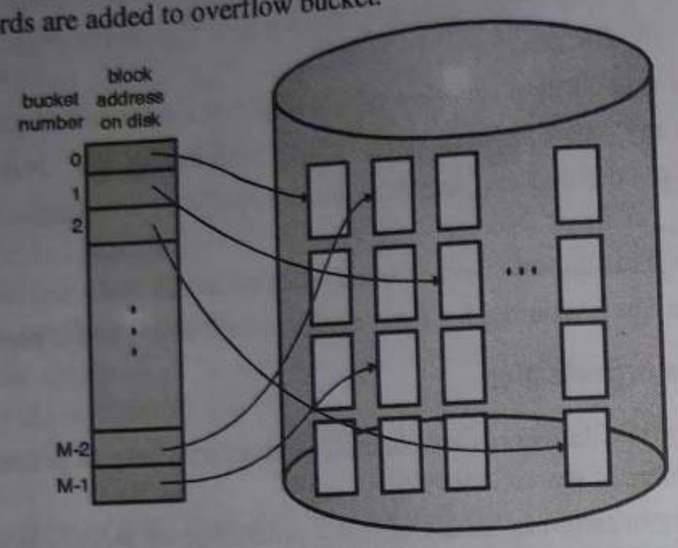


Fig. 11.3.2

4. Clustering file organization

- Records of multiple relations (tables) can be stored in the same file. Related records of the different relations are stored on the same block so that one I/O operation can fetch related records from any relations.
- Relation per file = 1

  Record type = fixed-length record
- Such file system is good for small size databases, which can reduce the code size.
- This Method combines tuples from multiple relations, but allows for efficient processing of the join.
- If the customer can have many accounts which cannot fit in one block, the remaining records can be stored on nearby blocks. Such file structure, called clustering.

11.4 Data Dictionary

Q. Write a note on : Data Dictionary.

Q. What do mean by metadata in dbms.

1. A DBMS needs to maintain a description of all the data items that it contains. Like a RDBMS contains information about all relation, database objects, views and index present in it.
2. The DBMS should also maintains information about views (a definition of the view) to compute the tuples that belong in the view whenever the view is queried or called.
3. The multiple relations which stores information about data in database. Such tabes are created and maintained by the system itself. They are called as **catalog relations.**
4. The catalog relations are also called the **system catalog** or the **data dictionary.**
5. The system catalog is sometimes also referred to as **metadata (data about data)**
6. The information in the system catalog is generally used only for query optimization.

Relation
– Name of given relation
– Name of Attribute and type of attributes of relation
– Index name for all indexes on the relation.
– Integrity constraints imposed on the relation
Index
– Name of index and structure of the index (e.g., hash index)
– Search key attributes
View
– Name of view name
– View definition.

Fig. 11.4.1 : Information stored in System Catalog for each data objects

Syllabus Topic : Hashing Techniques

11.5 Hashing Techniques

Q. Explain concept of hashing and its type.

1. Introduction

- Hashing is a type of primary file organization which provides very fast access to records in file on certain search conditions. This type of file system is also called as a **hash file.**

Scanned by CamScanner

- Hashing is a technique used with file organization for accessing records comparatively faster without using indices.
- Hashing can also be used for creation of index-structure.

2. Terms Related to Hashing

(a) Bucket

A bucket is a unit of storage containing one or more records (a bucket is typically a disk block which stores records).

(b) Hash Key

The search condition in hashing must be an equality condition on a single field, called the hash field or key field of the file, this is also called the hash key.

(c) Hash Function

- Uniform distribution of search values across a range of buckets. The bucket to which a value is assigned is determined by a function, called a hash function.
- Ideal hash function is random, so that each bucket will have the same number of records assigned to it irrespective of the actual distribution of search-key values in the file.

(d) Hash index

- Hash indices are always secondary indices.
- Hash index is one which organizes the search keys, with their associated record pointers, into a hash file structure.

(e) Bucket Overflow

- Bucket overflow can occur because of insufficient buckets.
- Skew in distribution of records. This can occur due to two reasons :
  - Multiple records have same search-key value
  - Chosen hash function produces non-uniform distribution of key values
- Although the probability of bucket overflow can be reduced, it cannot be eliminated; it is handled by using overflow buckets.

11.5.1 Static Hashing

1. Introduction

- Index schemes forces us for traverse an index structure while, hashing technique avoids this.

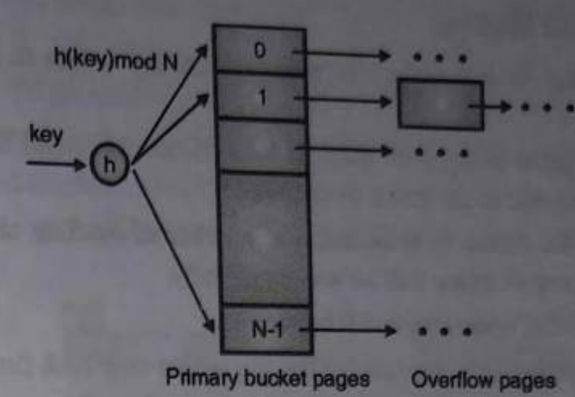


Fig. 11.5.1 Concept of Hashing

- The basic idea behind using hashing technique is to make use a hash function, which is used to map values in a search fields into a range of bucket numbers to find the page number on which a desired data is stored.

2. Method

- To search data in file system, we must apply a hash function h which gives the bucket number it belongs to and then it also search that bucket on file system.
- In a hash file organization we obtain the bucket number of a record directly from its search-key value using a hash function.
- Hash function is used to locate records for retrieval purpose, insertion as well as for deletion.
- Records with different search-key values may be mapped to the same bucket; thus entire bucket has to be searched sequentially to locate that specific record.
- In static hashing, function h maps search-key values to a fixed set of B of bucket addresses.
- The hash function is an important part of the hashing approach. It must distribute values of the search field uniformly over the collection of buckets.

3. Example

$F_1 \rightarrow 2N$ ..... (Hash functions)

$F_2 \rightarrow 2N + 1$ ..... Where N = 0, 1, 2, 3

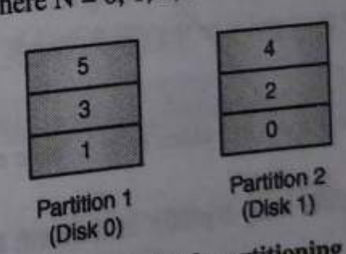


Fig.11.5.2 : Hash partitioning

Scanned by CamScanner

4. Deficiencies of Static Hashing

- In static hashing, hash function (h) maps search key values to a fixed set of bucket addresses (B).
- As Databases grow along with time. If buckets are too small initially, performance will be degrades due to too much overflows.
- If file size in the future is projected and number of buckets allocated accordingly, significant amount of space will be wasted initially.
- If database shrinks, again space will be wasted.
- One option is periodically reorganize the file with a new hash function, but it may be very expensive.
- These problems can be avoided by using techniques that allow the number of buckets to be modified dynamically.

11.5.2 Dynamic Hashing

- Good for database that grows and shrinks in size.
- Allows the hash function to be modified dynamically so problem of static hashing can be overcome.

11.5.3 Extendable Hashing

1. Introduction

- This is type of dynamic hashing.
- An extendible hash index consists of two parts:

(a) Buckets

- The buckets have a physical address on the disc
- Bucket contains a fixed number of records.

(b) Directory

- The directory indexes the buckets using some binary code.
- The directory consists of two parts: A binary code which is calculated from the hash function and A pointer to the bucket containing records matching the binary code.
- Two entries in directory may point to the same record.

2. Method

- We will generate a Hash function values over a large range typically b-bit integers, with b = 32.
- At any point of time we use only a prefix of the hash function to index into a table of bucket addresses.
- Let the length of the prefix be i bits, $0 \leq i \leq 32$.

- Bucket address table size = 2i.
- Initially i = 0
- Value of i will grows and shrinks as the size of the database grows and shrinks.
- Multiple entries in the bucket address table may point to a bucket.
- Thus, actual number of buckets is < 2i.
- The number of buckets also changes dynamically due to coalescing and splitting of buckets.

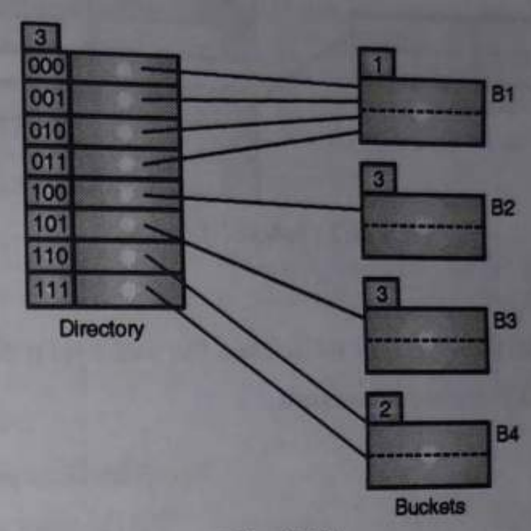


Fig. 11.5.3

3. Advantages

- Performance of method does not degrade as file size increases
- Stores the minimum number of buckets so optimizes storage space.
- Number of buckets grows/shrinks dynamically

4. Disadvantages

- The directory must be searched while looking for data.
- The directory required to be stored.

Syllabus Topic : Types of Indexes

11.6 Index - Concept of Indexing

Q. What do mean by index ? Explain with example.

1. Introduction

- Indexes are used to speed up search process like for example searching a book in library from many available books.
- In many places indexes are used for faster access to user data from set of data.

Scanned by CamScanner

- Goals of indexing are to improve Speed and efficiency of search process.

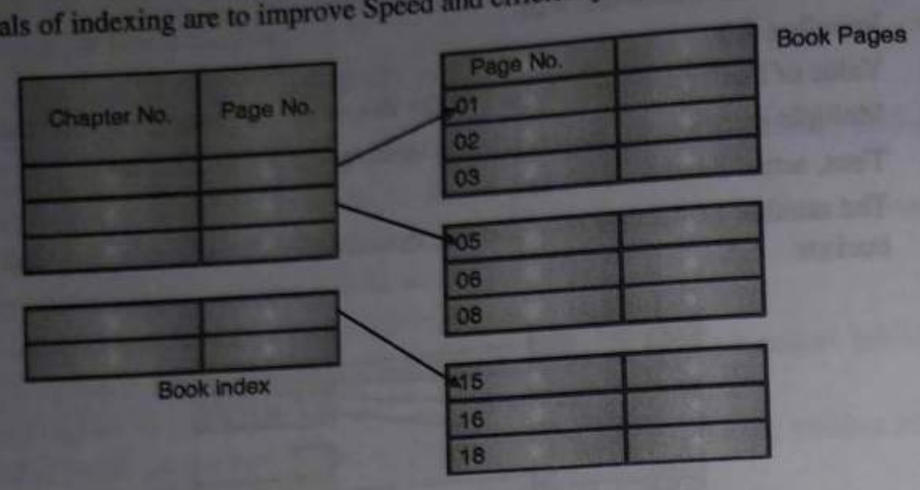


Fig. 11.6.1 : Indexing Concept

Search Key

- A field which is used to create index for database file, such field is called as search key.
- Any file is usually is composed of two parts,
  (a) Data
  (b) Index

Index File

- The index table is generally stored in a separate file which is called as the *index files*.
- Index file serves as the fast data dictionary index into an encoded file.
- For a given data file F, we create an index file called F_{idx}. Many file systems separate data and dictionary data, for efficiency and reliability.
- Data dictionary is considered more important and so it gets cached, stored, and updated separately.
- The index file is separate from the encoded file data for the same reasons and to allow us to manage each part separately and simply.

Efficiency of index based on,

- **Access type** - Type of data access that are supported efficiently
- **Access time** - Time to access a particular data item.
- **Insertion time** - Time taken to add a new data item
- **Deletion time** - Time taken to remove particular data item
- **Space overhead** - Extra space occupied by an index structure.

Syllabus Topic : Single-Level Ordered Indexes

11.7 Ordered Indices

Q. What do mean by index ? Explain concept of Single level ordered index.

1. Introduction

- Ordered access structures are similar to table of contents or indexes in books.
- The indexes or tables of contents list the chapter in sequential order, along with page numbers where the detailed information about it can be found.
- We are using indexing on a key field. If indexes are stored and sorted on basis of such search key field then it is called as ordered index.
- E.g. Author's Catalog in library
- Search key is a field, which uniquely identify the record in a file. Index files are typically much smaller than the original file. Index record holds search key value and pointers to the records with the value.
- Index-Sequential files are files (which hold information for data) ordered sequentially on a search key.

11.7.1 Single Level ordered Index

- The alternative to index of book is to have a linear search of the textbook, looking through each page one at a time.
- For a file consisting of several fields, and index access structure is usually defined on a single field, which is called an **indexing field**.
- An index usually is composed of two parts,
  (a) Index field value
  (b) List of pointers to all disk blocks which contain records with that field value.
- The values in the index are ordered, so we can do a binary search on such index.
- The index file is much smaller than the data file, so the binary search is much faster.
- There are different types of indexes, these include:

1. Primary Indexes
2. Clustering index
3. Secondary index

1. Primary index

- In a sequentially ordered file, the index whose search key specifies the sequential order of the data file is called as primary index.

Scanned by CamScanner

- The search key of a primary index is usually primary key.
- Records of these file are of fixed length with two fields.
  (a) Ordering key field primary key of data file
  (b) Pointer to a disk clock.
- There is one index entry in the index file for each block in the data file.
- Two field values of index entry i as <K(i), P(i)>.
- Following is the example for primary key index.

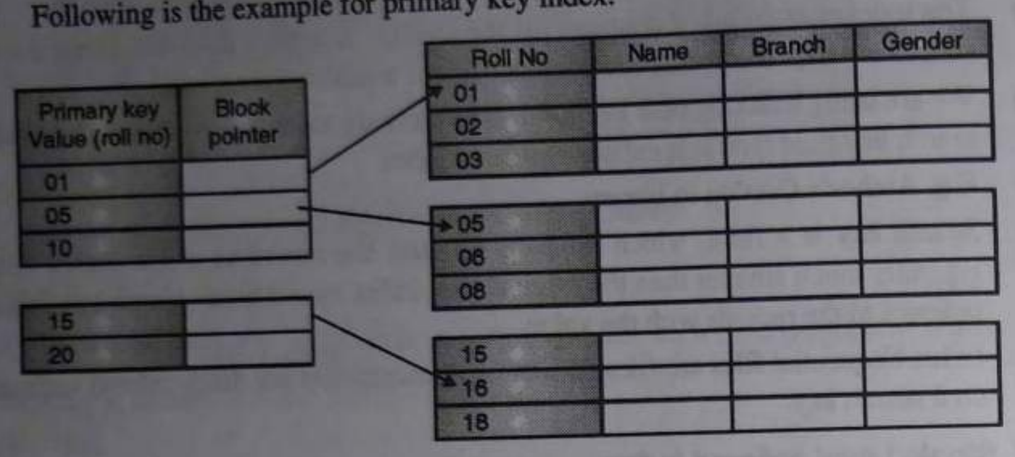


Fig. 11.7.1 : Primary Index

- The first record in each block of the data file is called anchor record or block anchor.
- The indexes can also be categorized into following types.

(a) Dense index

- Index record appears for each and every search-key value in the file is called as dense index.
- In a dense primary index, the index record is

< Search key value, Pointer to first record with search-key>

(b) Sparse Index

- Sparse index contains index records for only some search-key values.
- It is Applicable when records are sequentially ordered on search-key.
- To locate a record with search-key value K we:
- Find index record with largest search-key value < K>
- Search file sequentially starting at the record to which the index record points

(c) Comparison

- Sparse index occupies less space and less maintenance overhead for insertions and deletions.

- Sparse index is generally slower than dense index for locating records.
- Sparse index with an index entry for every block in file, corresponding to least search-key value in the block.
- Primary index is a non dense index.

2. Clustering Indexes

- A clustering index is used to determine how rows in table are physically ordered (clustered) in a table space.
- Clustering indexes provide a good performance in some operations which involve large number of records.
- Examples are grouping operations, ordering operations, and comparisons.
- If records of a file are physically ordered based on any non key fields (a key which may not have a distinct value for each record) such field is called the clustering field. This differs from a primary index (primary key requires that the ordering field of the data file have a distinct value for every record).
- Clustering index is also an ordered file following fields.
  - Data file
  - Block pointer
- There is one clustering index for each distinct value of the clustering field, containing the data value and a pointer to its first block in the data file that has a record with that value for its clustering field.
- Clustering index is a non dense index.
- An index is similar to the directory structure used for extensible hashing. Both are searched to find a pointer to the data block containing the desired record.
- A main difference between index and hashing is that, index search uses the values of the search field itself; whereas it has function directory search uses the hash value that is calculated by applying some hash function to the search field.
- **E.g.** Clustering index can be applied on branch not for Student. Here there can be branch numbers duplicated in student table. Here the clusters are also contains many search key values into one cluster for example in first cluster the records are containing branch no 1 and 2.

Scanned by CamScanner

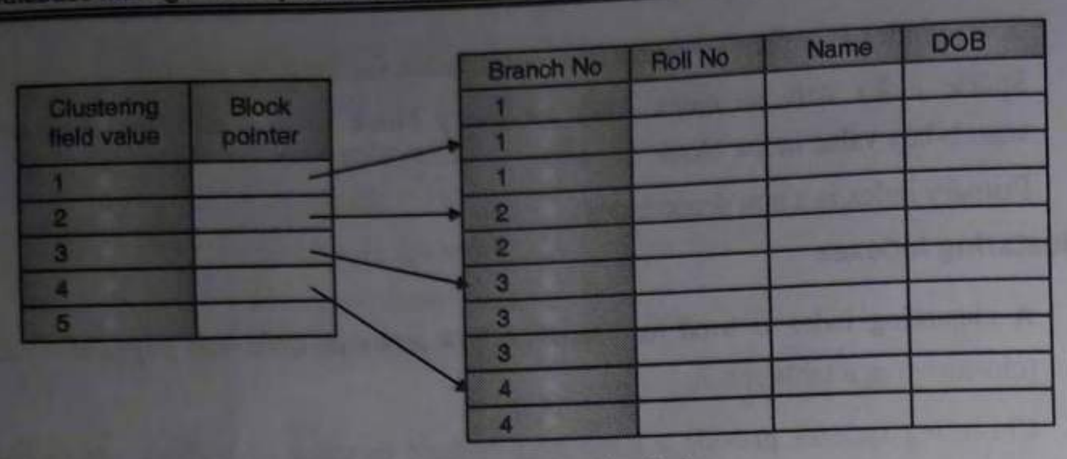


Fig. 11.7.2 : Clustering Index

For example clustering index with a separate block cluster for each group of records that hare the same value for the clustering field.

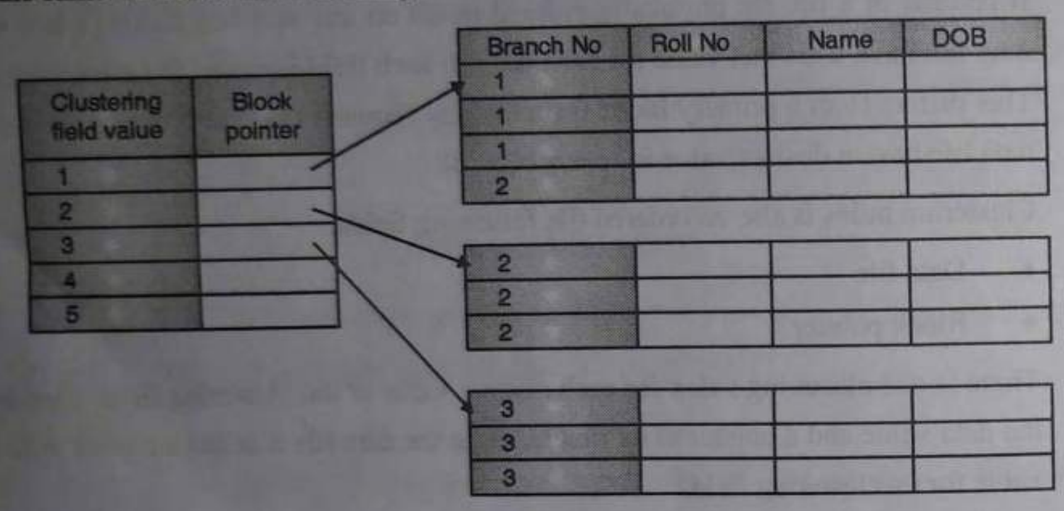


Fig. 11.7.3 Clustering Index

3. Secondary index

- Secondary index is a type of index which is maintained for a data file, but not used to control the current processing order of the file.
- Example, a secondary index could be maintained for customer name, while the primary index is set up for customer_ID.
- Sometimes we may wants to find all the records whose values in a certain field fulfil some condition. This field may not the search-key of the primary index.
- We can use a secondary index with an index record for each search-key value.
- Secondary index is a non-clustering index.
- A secondary index provides a secondary means of accessing a file for which some primary access already exists.

- Secondary indices must be **dense**, with an index entry for every search-key value, and a pointer to every record in the file.
- The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non key with duplicate values.
- Secondary indices improve the performance of queries on non-primary keys.

Syllabus Topic : Multilevel Indexes

11.8 Multilevel Index

- The idea of multilevel indexes is to reduce the part of the index that we continue to search by a larger factor. If primary index does not fit in memory, access becomes expensive.
- The blocking factor of the index, where the bfr_i is greater than 2. The blocking factor of the index, bfr_i is called the fan-out, or f_o of the multilevel index.
- The index file is called the first level of a multilevel index. It is an ordered file with a distinct value for each key value K(i). Therefore we can create a primary index for the first level. This index to the first level is called the second level of the multilevel index.
- The number of third level entries is $r_3 = [(r_2/f_o)]$.
  - Outer index – a sparse index of primary index.
  - Inner index – the primary index file.
- If even outer index is too large to fit in main memory, yet another level of index can be created, and so on Indices at all levels must be updated on insertion or deletion from the file.

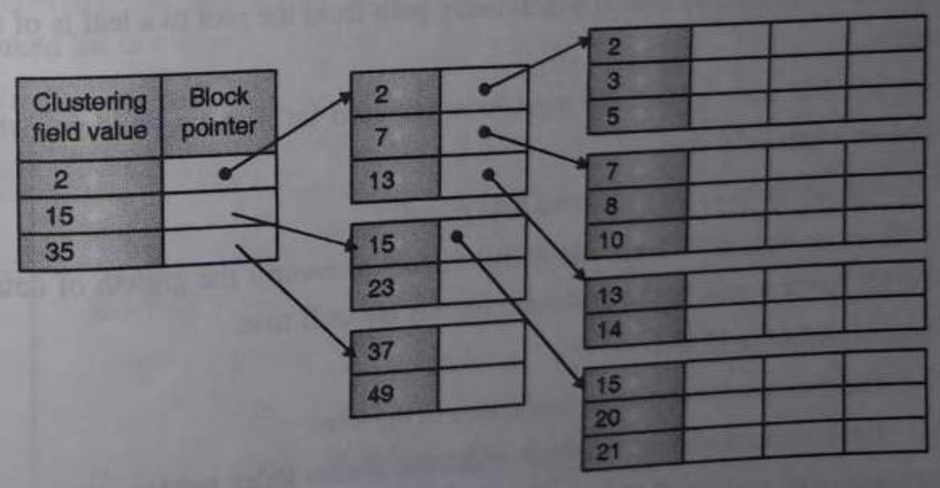


Fig. 11.8.1 : A top level primary index resembling ISAM (Indexed Sequential Access Method) organization

Scanned by CamScanner

- A common file organization used in business data processing is an ordered file with a multilevel primary index on its ordering key field. Such organizations are called an indexed sequential file (Example is ISAM).

Syllabus Topic : Overview of B-Trees and B+-Trees

11.9 Overview of B Tree and B + Tree

Q. Explain concept of B Trees and B+ Trees.

1. Introduction

Primary disadvantage of index-sequential file organization (ISAM) is that performance degrades as the size of data file grows.

2. B Trees

Q. Write a note on on B Trees.

- B Trees are multi-way trees means each node contains a set of keys and pointers.
- A minimum size of B Tree is with 4 keys and five pointers.
- B Trees are dynamic in nature in which the height of the tree grows or shrinks as new records are added and old records are deleted.

3. B + Trees

Q. Explain concept of B+ Trees. Give all b+ tree operations.

- A B+ Tree combines features of ISAM and B Trees.
- A B+ tree is a balanced tree in which every path from the root to a leaf is of the same length.
- Each non-leaf node in the tree must have between [n/2] and n children, where n is fixed for a particular tree.
- A B+ tree contains index pages and data pages.
- B+ Trees as well as B Trees use a "fill factor" to control the growth of data file. A 50% fill factor would be the minimum for any B+ or B tree.
- Features similar to ISAM
  - The data always appear as leaf nodes in the tree.
  - The root node and intermediate nodes are always index pages.
- Contrasting features to ISAM
  - Overflow pages are not used in case of B+ trees.

4. B + Tree Operations

(a) Searching for node in B+ Tree

A B+ Tree Starts search at root node then uses key comparisons to go to leaf through intermediate nodes.

Example

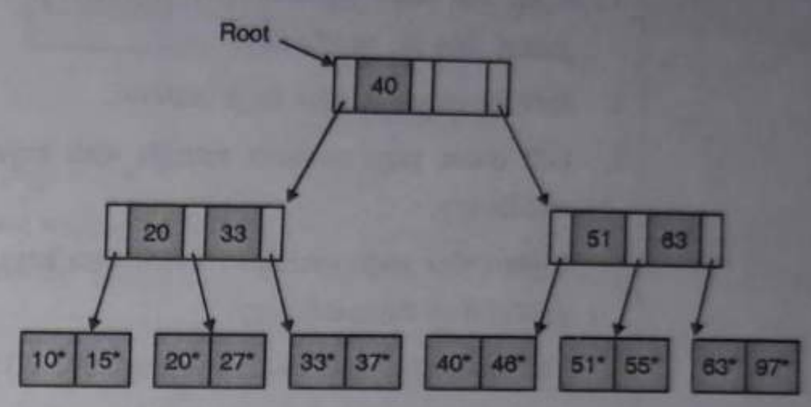


Fig. 11.9.1

- In above tree if we want to search 15 then we will start with root which is 50 then go to next node (20, 30) and finally reaches to node 15.

(b) Insertion of node in B+ Tree

- A key value determines placement of record in a B+ tree.
- Generally, the leaf pages will be maintained in sequential order we use a doubly linked list to connect each leaf page with its sibling page(s).
- We must consider three scenarios while inserting a record to a B+ tree.

Leaf Page	Index Page	Algorithm to follow
Not Full	Not Full	Place a given record in sorted position in the proper leaf page
Full	Not Full	1. Split the required leaf page into two. 2. Select middle key value from above values. 3. Rearrange all key values in the sorted order. 4. Left leaf page contains records with keys below the middle key. 5. Right leaf page contains records with keys equal to or greater than the middle key.

Scanned by CamScanner

Leaf Page	Index Page	Algorithm to follow
Full	Full	1. Split the required leaf page into two. 2. Left leaf page contains records with keys below the middle key. 3. Right leaf page contains records with keys equal to or greater than the middle key. 4. Split the required index page into two. 5. Left index page contains records with keys below the middle key. 6. Right index page contains records with keys equal to or greater than the middle key. 7. The middle key goes to the next (higher level) index. If the next level index page is full, continue splitting the index pages.

Example

- Fig.11.9.2 contains 4 key values. We will use this tree to demonstrate above insertion operations.

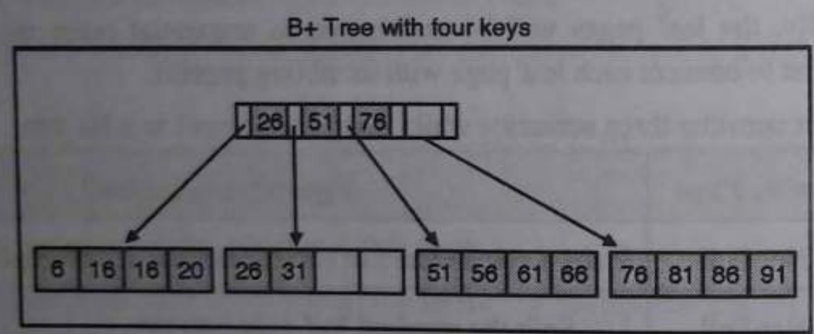


Fig.11.9.2

Case 1 : Leaf and Index Both are Not Full

- Suppose we want to add value 30 to existing B+ tree. In this operation we found appropriate position for 30.We can place 30 in node (27, 32) which having sufficient place in it. So we will update this node as (27, **30**, 32) in which 30 is placed in increasing order.

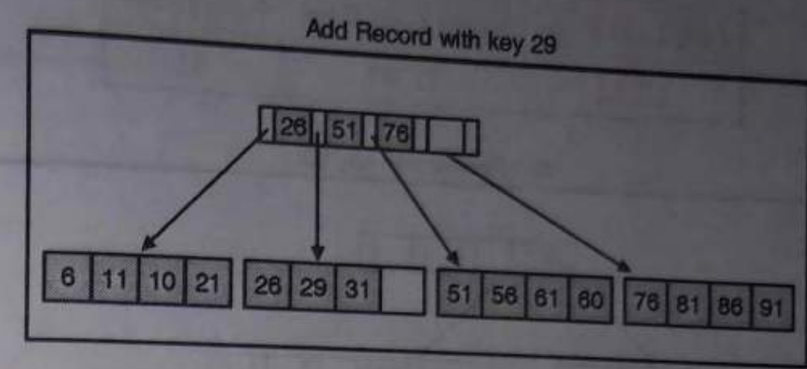


Fig.11.9.3

Case 2 : Leaf page is Full But not Index page

- Suppose we want to add value 72 to existing B+ tree. This record should go in the leaf page containing (52, 57, 62, 67), But this page is full. So, we must split the page as follows:

Left Leaf Page	Right Leaf Page
52 ,57, 62	67, 72

- The middle key of 62 is placed in the index page between 62 and 77. The following table shows the B+ tree after the addition of 72.

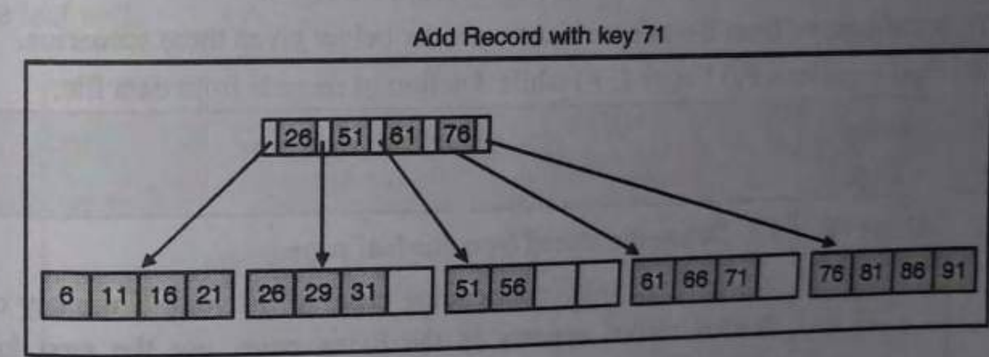


Fig.11.9.4

Case 3 : Leaf and Index Both are Full

- Suppose we want to add value 97 to existing B+ tree. This record belongs in the page containing (77, 82, 87, 92). Since this page is full we need to split it into two pages:

Left Leaf Page	Right Leaf Page
77, 82	87, 92, 97

- The middle key is 87 which rise to the index page. But, the index page is also full, so we split the index page:

Scanned by CamScanner

Left Leaf Page	Right Leaf Page	New Index
27, 52	77, 87	62

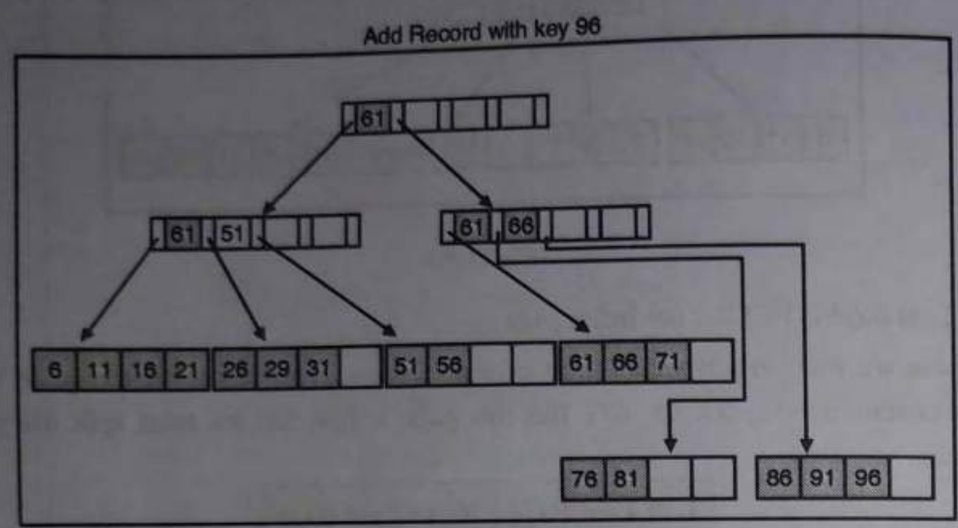


Fig.11.9.5

(c) Deletion of node from B+ Tree

- A key value determines placement of record in a B+ tree.
- To delete a record from B+ tree we must consider below given three scenarios.
- We must consider a Fill Factor (FF) while deletion of records from data file,

Leaf Page	Index Page	Algorithm to follow
Above FF	Above FF	1. Delete the record from the leaf page. 2. Arrange keys in ascending order to fill void. If the key of the deleted record appears in the index page, use the next key to replace it.
Below FF	Above FF	Combine the leaf page and its sibling. Change the index page to reflect the change.
Below FF	Below FF	1. Combine the leaf page and its sibling. 2. Adjust the index page to reflect the change. Combine the index page with its sibling. Continue combining index pages until you reach a page with the correct fill factor or you reach the root page.

Example

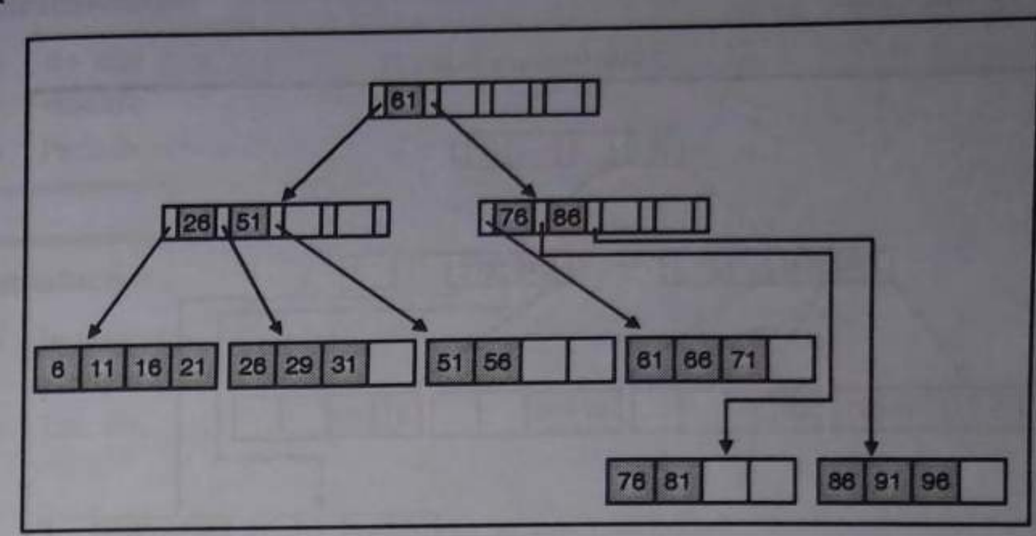


Fig. 11.9.6

Case 1 : Leaf and Index Both are above fill factor

- Suppose we want to delete value 72 to existing B+ tree. In this operation we found position of 72 and delete it from page (62, 67, 72) which will now become (62, 67) only 2 records as our fill factor is 50% or (2 records) we simply delete 72 from the leaf node.

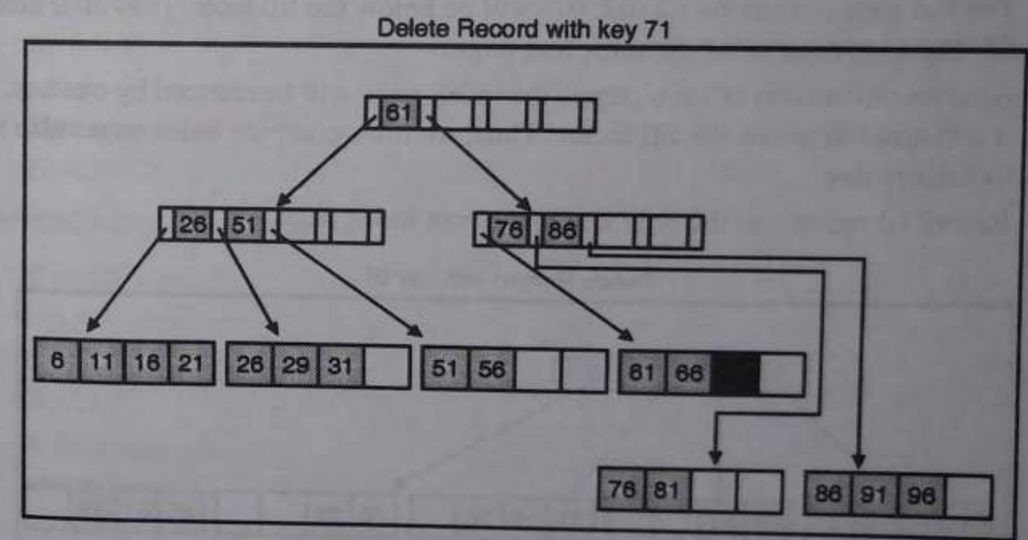


Fig.11.9.7

Case 2 : Leaf page is below fill factor

- Suppose we want to delete record 27 from existing B+ tree. This record is found in the leaf node containing (27, 30, 32) in above diagram. The fill factor will be 50% after the

Scanned by CamScanner

deletion; 27 appear in the index page. Thus, when we delete 27 we must replace 27 with 30 in the index page.

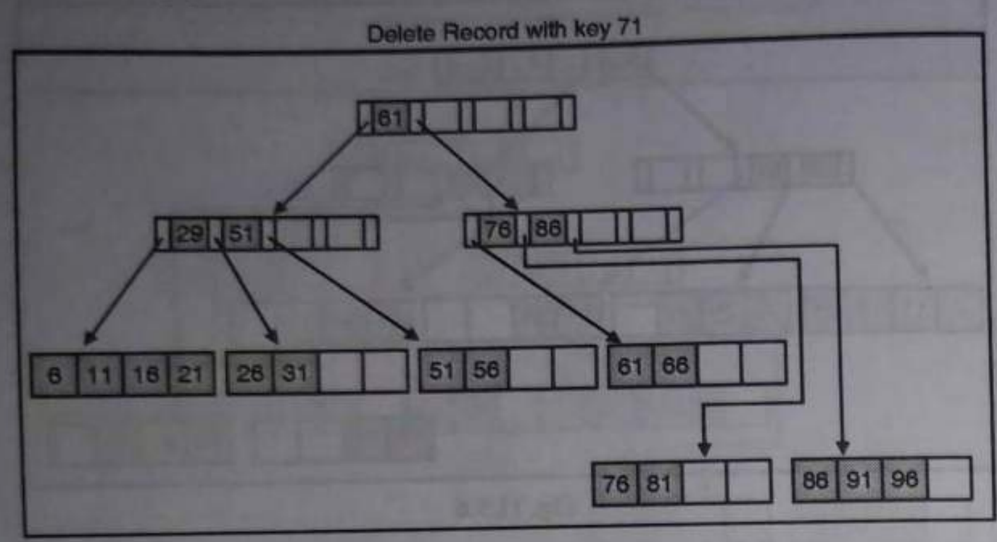


Fig 11.9.8

Case 3 : Leaf and Index Both are Full

- Suppose we want to delete record 62 from the B+ tree. We need to go through following steps,
  1. The leaf page containing 62 (62, 67) will be below the fill factor (FF) after deletion of 62. Thus, we must combine some leaf pages.
  2. After reconstruction of such pages, the index page will be reduced by one key. Hence, it will also fall below the fill factor. Thus, we must combine index pages also in order to balance tree.
  3. Record 62 appears as the only key in the root index page.

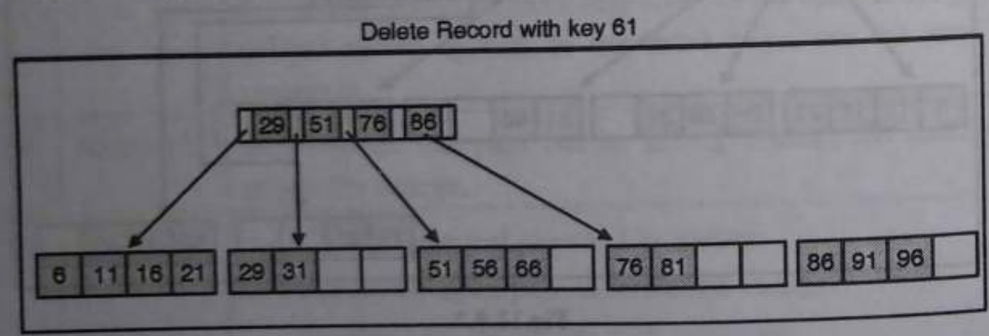


Fig.11.9.9

5. Advantages

- B+ tree index automatically reorganizes itself in the case of insertions and deletions.

- Reorganization of entire file is not required to maintain performance.

6. Disadvantages

- B+ tree index degrades performance as file grows. (As many overflow blocks get created)
- Periodic reorganization of entire file is required.

Syllabus Topic : Indexes on Multiple Keys

1. Introduction

- In sequential file organization and fetching of records is based on the primary key which is used in the query.
- The real world queries are not restricted to conditions on primary key or single column.
- If we only have a single key access it will degrade the performance of the query.
- For example employee table with large number of employees in different department.
- Select all the employees whose salary is 3000 and works for department ID 60.

```
SELECT    *
FROM      EMPLOYEE
WHERE     DEPT_ID = 20
AND SALARY = 3000;
```

- Since there is index on DEPT_ID, all the employees who are working for department 20 will be fetched quickly. But it will return large number of employees. The filter to process the employees with salary = 3000 will be time consuming part of the query as there are no indexes defined.
- To solve above problem we need to introduce the indexes on multiple keys.

2. Ordered Index on Multiple keys

- If multiple attributes are frequently used in queries it is beneficial to set up a multiple key access structure to provide efficient access.
- It is expected to put columns with highest unique value first. That is if create (SALARY, DEPT_ID) index than (DEPT_ID, SALARY) is most efficient.
- A lexicographic ordering of these tuple values performs an order on this composite search key.
- An index on a composite key of *n* attributes works same as any other index.

3. Partitioned Hashing

- Partitioned hashing is an extension of static external hashing which allows access on multiple keys.
- This type of hashing index is only for equality comparisons.

Scanned by CamScanner

- In partitioned hashing, for a key consisting of n components, the hash function is designed to produce a result with n separate hash addresses.
- The bucket address is acting as concatenation of these n addresses.
- The main drawback of partitioned hashing is that it cannot handle range queries on any of the component attributes.

3. Advantages

- This index will works efficiently when all the columns of the index are used in the query.
- Index is designed to works efficiently even on larger tables. As, all records are sorted based on the index.

4. Disadvantages

- It works in lexicographic ordering so if query not using the index then cost of the query is high.
- It does not work efficiently if few columns of index are used in query, then it will not use this index to retrieve the data.

Review Questions

Q. 1 What are different file organizations ?

Q. 2 What is difference between Indexing and Hashing ?

Q. 3 Explain B+ tree and its advantages.

Q. 4 Explain single level indexes with example.

Q. 5 What is hashing ? Explain different types of hashing methods.

Q. 6 Write short note on : Hashing technique.

Q. 7 Explain the insertion of entry in a B^+ tree.

Q. 8 Write short note on : Hashing technique.

Q. 9 Write insertion of entry in B+ tree.

Q. 10 Write short note on : Hashing technique.

Q. 11 Write short note on : B+ tree.

□□□

Appendix A

PLSQL Basics

A.1 Introduction to PLSQL

- PL/SQL is Oracle Corporation's procedural language which is 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.
- **Sections**
  - A PL/SQL block made up to 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.
- **Block Structure**

```
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
```

Scanned by CamScanner

- **Executing PL/SQL Blocks**
  - 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.

Example

Simple PL/SQL Block to increment salary of all employees.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
-- Variable declarative section
SQL> DECLARE
  v_salary      NUMBER(6);
-- Executable Section
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;

SQL> /
The salary of all employees are Updated
PL/SQL procedure successfully completed.
```

A.2 Declaring and Initializing

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 PLSQL 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 specifies that either value of variable is mandetary or optional
DEFAULT / :=	This clause is used to specify default or initial value of variable.':=' is an assignment operator in PLSQL.
Expr	Initial value of a variable

5) Example

```
SQL> DECLARE
    v_dob        DATE;
    v_branchno   NUMBER(5) NOT NULL := 50;
    v_address    VARCHAR2(50) := 'Unknown';
    v_fees       CONSTANT NUMBER(10,2) := 50000.00;
```

A.3 PLSQL 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*.

Scanned by CamScanner

- 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 PLSQL.
- 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.

```
SQL> BEGIN
DBMS_OUTPUT.PUT_LINE('Output Tested ');
END;
SQL> /
Output Tested
PL/SQL procedure successfully completed.
```

4) Printing PLSQL Variables

To print variable no need to write single quotation.

```
SQL> DECLARE
    Var_number Number :=1;
BEGIN
DBMS_OUTPUT.PUT_LINE(Var_number);
END;

SQL> /
1
PL/SQL procedure successfully completed.
```

5) Appending String and Variables in output

To print text string we use single quotation not for integer or number value.

```
SQL> DECLARE
Var_number Number :=1;
BEGIN
DBMS_OUTPUT.PUT_LINE('Number Accepted = ' ||Var_number);
END;
```



```
SQL> /
Number Accepted = 1
PL/SQL procedure successfully completed.
```

A.4 PLSQL Inputs using Substitution Variable

1) Introduction

We use substitution variable for accepting input using PLSQL.Substitution variable is prefixed with '&'.

2) Working

- Whenever PLSQL 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) Accpting text String

To accept text string we use single quotation.

```
SQL> DECLARE
Var_N VARCHAR(50) := '&Name';
BEGIN
DBMS_OUTPUT.PUT_LINE(' Accepted Name = '||Var_N);
END;

SQL> /
Enter value for Name: VisionIT
Accepted Name = VisionIT
PL/SQL procedure successfully completed.
```

Scanned by CamScanner

5) Accepting number variables

To print variable no need to write single quotation.

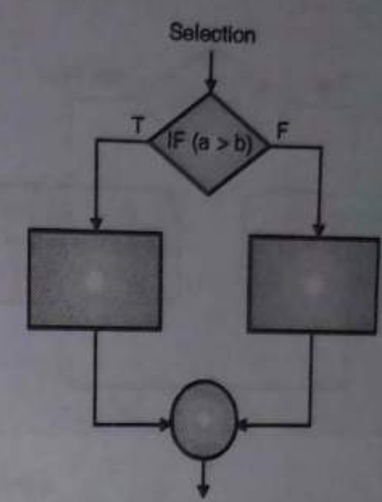
```
SQL> DECLARE
Var_Number number:= &Number;
BEGIN
DBMS_OUTPUT.PUT_LINE(' Accepted Number = '||Var_Number);
END;

SQL> /
Enter value for number: 12
Accepted Number = 12
PL/SQL procedure successfully completed.
Product ID  : 1
Zone        : S
Total Sales : 16
PL/SQL procedure successfully completed.
```

A.5 Introduction to Control Structures in PLSQL

- A control structure is a primary concept in all most all the high-level programming languages.
- PLSQL 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 below,
  1. Decision making Statements (IF statement and CASE statement)
  2. Iterative statements (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.

A.5.1 IF Statement



(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

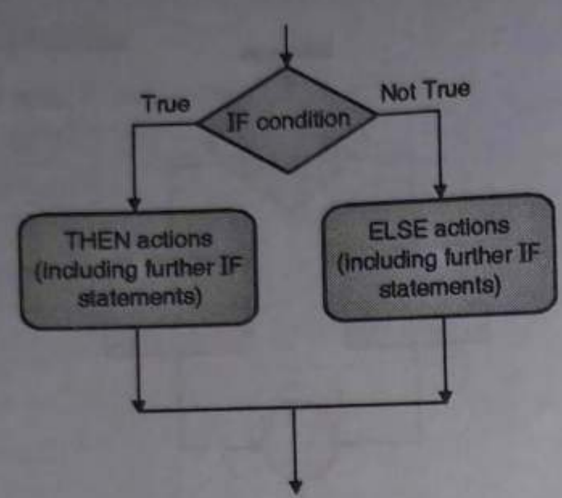
```
IF condition THEN
   Statements;
[ELSEIF condition THEN
   Statements;]
[ELSE
   Statements;]
END IF;
```

A.5.2 IF..ELSE Statement

(a) Introduction

- 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.

Scanned by CamScanner



(b) Syntax

```
IF condition THEN
   Statements;
ELSE
   Statements;
END IF;
```

(c) Example

Write a PLSQL block to check given number is greater than 100 or not.

```
SQL> SET SERVEROUTPUT ON;
-- Variable declarative section

SQL> DECLARE
 v_number      NUMBER(6):=&Number;
-- Executable Section
BEGIN
IF v_number > 100 THEN
-- the following displays output from the PL/SQL section
 DBMS_OUTPUT.PUT_LINE(v_number || ' is greater then 100');
ELSE
-- the following displays output from the PL/SQL section
 DBMS_OUTPUT.PUT_LINE(v_number || ' is not greater then 100');
END IF;
END;
```



```
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 then 100
PL/SQL procedure successfully completed.
```

A.6 Iterative Statements

- 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

- Basic loop to provide repetitive actions without overall conditions.
- FOR loops to provide iterative control of actions based on a count.
- WHILE loops to provide iterative control of actions based on a condition.

A.7 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
- You can terminate a basic loop using the EXIT statement.
- Flow of control is passes to the next statement after the END LOOP statement.

Scanned by CamScanner

(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) Example

Write a PLSQL block to generate 10 odd numbers using Basic loop.

```
SQL> 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;

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.
```

Create table part_details then write a PLSQL 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 .

```
SQL> CREATE TABLE part_details(part_id VARCHAR(10), subpart_id VARCHAR(10))
SQL> /
Table created.

SQL> 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;

SQL> /
PL/SQL procedure successfully completed.

SQL> SELECT * FROM part_details
SQL> /
PART_ID          SUBPART_ID
----------       ---------
50                    1
50                    2
50                    3
50                    4
50                    5
```

Scanned by CamScanner

```
50          6
50          7
50          8
50          9
50          10
10 rows selected
```

A.8 FOR LOOP Statement

(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 PLSQL 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) Example

Write a PLSQL block to generate 10 odd numbers using for loop.

```
SQL> 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;

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.
```

Scanned by CamScanner

Create table part_details then write a PLSQL 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 .

```
SQL> CREATE TABLE part_details(part_id VARCHAR(10),subpart_id VARCHAR(10))
SQL> /
Table created.

SQL> 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;

SQL> /
PL/SQL procedure successfully completed.

SQL> SELECT * FROM part_details
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
```


Demonstration of FOR REVERSE Loop to generate 3 numbers.

```
SQL>
BEGIN
FOR i IN REVERSE 1..3
LOOP
    DBMS_OUTPUT.PUT_LINE(TO_CHAR(i));
END LOOP;
END;

SQL> /
3
2
1
PL/SQL procedure successfully completed.
```

A.9 WHILE LOOP Statement

(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.

Scanned by CamScanner

(c) Example

Write a PLSQL block to generate 10 odd numbers using while loop.

```
SQL> 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;

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.
```


Create table part_details then write a PLSQL 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.

```
SQL> CREATE TABLE part_details(part_id VARCHAR(10),subpart_id VARCHAR(10))

SQL> /
Table created.

SQL> 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;

SQL> /
PL/SQL procedure successfully completed.

SQL> SELECT * FROM part_details
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
```

Scanned by CamScanner

A.10 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 then one EXIT statements.

(b) Syntax

```
LOOP          -- Start of Loop
Statement 1; -- Statement

IF v_Counter > 10 THEN
    EXIT;
END IF;

END LOOP;    -- Start of Loop
```

(c) Example

Create table part_details then write a PLSQL block to insert 10 rows to part_details table.

```
SQL> CREATE TABLE part_details (part_id VARCHAR(10),subpart_id VARCHAR(10));

SQL> /
Table created.

SQL> 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;

SQL> /
PL/SQL procedure successfully completed.

SQL> SELECT * FROM    part_details;
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
```

A.11 Sample Programs

Write a PLSQL block to print given number of odd numbers.

```
-- Option to Print Output
SQL> SET SERVEROUTPUT ON;
-- Variable declarative section
SQL> 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);
```

Scanned by CamScanner

```
  v_count := v_count + 1;
 END LOOP;
END;
SQL> /
Enter value for num: 5
1
3
5
7
9
PL/SQL procedure successfully completed.
```

Scanned by CamScanner