SUBJECT CODE : 18CS53

As per Revised Syllabus of

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Outcome Based Education (OBE) & Choice Based Credit System (CBCS)
Semester - V (CSE / ISE)

DATABASE MANAGEMENT SYSTEM

Anuradha A. Puntambekar
M.E. (Computer)
Formerly Assistant Professor in
P.E.S. Modern College of Engineering,
Pune

TECHNICAL PUBLICATIONS
SINCE 1993
An Up-Thrust for Knowledge

DATABASE MANAGEMENT SYSTEM

Subject Code : 18CS53

Semester - V (CSE / ISE)

Published by :

TECHNICAL PUBLICATIONS
An Up-Thrust for Knowledge

Amit Residency, Office No.1, 412, Shaniwar Peth,
Pune - 411030, M.S. INDIA, Ph.: +91-020-24495496/97
Email : sales@technicalpublications.org Website : www.technicalpublications.org

Printer :
Yogiraj Printers & Binders
Sr.No. 10/1A,
Ghule Industrial Estate, Nanded Village Road,
Tal. - Haveli, Dist. - Pune - 411041.

ISBN 978-81-947993-0-6

VTU 18

PREFACE

The importance of **Database Management System** is well known in various engineering fields. Overwhelming response to my books on various subjects inspired me to write this book. The book is structured to cover the key aspects of the subject **Database Management System.**

The book uses plain, lucid language to explain fundamentals of this subject. The book provides logical method of explaining various complicated concepts and stepwise methods to explain the important topics. Each chapter is well supported with necessary illustrations, practical examples and solved problems. All the chapters in the book are arranged in a proper sequence that permits each topic to build upon earlier studies. All care has been taken to make students comfortable in understanding the basic concepts of the subject.

Representative questions have been added at the end of each section to help the students in picking important points from that section.

The book not only covers the entire scope of the subject but explains the philosophy of the subject. This makes the understanding of this subject more clear and makes it more interesting. The book will be very useful not only to the students but also to the subject teachers. The students have to omit nothing and possibly have to cover nothing more.

I wish to express my profound thanks to all those who helped in making this book a reality. Much needed moral support and encouragement is provided on numerous occasions by my whole family. I wish to thank the **Publisher** and the entire team of **Technical Publications** who have taken immense pain to get this book in time with quality printing.

Any suggestion for the improvement of the book will be acknowledged and well appreciated.

Author
A. A. Puntambekar

Dedicated to God.

SYLLABUS

Database Management System - (18CS53)

Credits : 04	CIE Marks : 40	SEE Marks : 60	Exam Hours : 03

Module - 1

Introduction to Databases : Introduction, Characteristics of database approach, Advantages of using DBMS approach, History of database applications. **Overview of Database Languages and Architectures** : Data Models, Schemas and Instances, Three schema architecture and data independence, Database languages and interfaces, The database system environment.

Conceptual Data Modelling using Entities and Relationships : Entity types, Entity sets, attributes, roles and structural constraints, Weak entity types, ER diagrams, examples, Specialization and Generalization. **(Chapter - 1)**

Module - 2

Relational Model : Relational Model Concepts, Relational Model Constraints and Relational Database Schemas, Update Operations, Transactions and Dealing with Constraint Violations. **Relational Algebra** : Unary and Binary Relational Operations, Additional Relational Operations (Aggregate, Grouping etc.), Examples of Queries in Relational Algebra, **Mapping Conceptual Design into a Logical Design** : Relational Database Design using ER-to-Relational Mapping.

SQL : SQL data definition and data types, specifying constraints in SQL, retrieval queries in SQL, INSERT, DELETE and UPDATE statements in SQL, Additional features of SQL.

(Chapters - 2, 3)

Module - 3

SQL : Advances Queries : More complex SQL retrieval queries, Specifying constraints as assertions and action triggers, Views in SQL, Schema change statements in SQL. **Database Application Development** : Accessing database from applications, An introduction to JDBC, JDBC classes and interfaces, SQLJ, Stored procedures, Case study : The internet bookshop.

Internet Applications : The three-tier application architecture, The presentation layer, The middle tier. **(Chapter - 4)**

Module - 4

Normalization : Database Design Theory : Introduction to Normalization using Functional and Multivalued Dependencies : Informal Design Guidelines for Relation Schema, Functional Dependencies, Normal Forms, based on Primary Keys, Second and Third Normal Forms, Boyce-Codd Normal Form, Multivalued Dependencies and Fourth Normal Form, Join dependencies and Fifth Normal Form. **Normalization Algorithms** : Inference Rules, Equvalence, and Minimal Cover, Properties of Relational Decompositions, Algorithms for Relational Database Schema Design, Nulls, Dangling Tuples and Alternate Relational Designs, Further discussion of Multivalued dependencies and 4NF, Other Dependencies and Normal Forms. **(Chapter - 5)**

Module - 5

Transaction Processing : Introduction to Transaction Processing, Transaction and system concepts, Desirable properties of Transactions, Characterizing schedules based on recoverability, Characterizing schedule based on Serializibility, Transaction support in SQL, **Concurrency Control in Databases** : Two Phase Locking Techniques for Concurrency Control, Concurrency Control Based on Timestamp Ordering, Multi-version Concurrency control techniques, Validation Concurrency control techniques, Granularity of data items and multiple granularity locking. **Introduction to Database Recovery Protocols** : Recovery Concepts, NO-UNDO/REDO recovery based on deferred update, Recovery techniques based on immediate update, shadow paging, Database backup and recovery from catastrophic failures. **(Chapter - 6)**

TABLE OF CONTENTS

Module - 1

Module - 2

Part III : Mapping Conceptual Design into Logical Design

Part II : Concurrency Control in Databases

Part III : Introduction to Database Recovery Protocol

1 Introduction to Databases and Data Modelling

Syllabus

Introduction to Databases : *Introduction, Characteristics of database approach, Advantages of using DBMS approach, History of database applications.*

Overview of Database Languages and Architectures : *Data Models, Schemas and Instances, Three schema architecture and data independence, Database languages and interfaces, The database system environment.*

Conceptual Data Modelling using Entities and Relationships : *Entity types, Entity sets, attributes, roles and structural constraints, Weak entity types, ER diagrams, examples, Specialization and Generalization.*

Contents

Part I : Introduction to Databases

1.1 Introduction

- **Definition :** A database management system (DBMS) is collection of **interrelated data** and various **programs** that are used to handle the data.
- The **primary goal** of DBMS is to provide a way to **store and retrieve** the required information from the database in convenient and efficient manner.
- For managing the data in the database two important **tasks** are conducted -
  - **Define the structure** for storage of information.
  - Provide mechanism for **manipulation of information.**
- In addition, the database systems must ensure the **safety of information** stored.

Database System Applications

There are wide range of applications that make use of database systems. Some of the applications are -

1) **Accounting :** Database systems are used in maintaining information employees, salaries, and payroll taxes.
2) **Manufacturing :** For management of supply chain and tracking production of items in factories database systems are maintained.
3) For maintaining customer, product and purchase information the databases are used.
4) **Banking :** In banking sector, for customer information, accounts and loan and for performing banking applications the DBMS is used.
5) For purchase on credit cards and generation of monthly statements database systems are useful.
6) **Universities :** The database systems are used in universities for maintaining student information, course registration, and accounting.
7) **Reservation systems :** In airline/railway reservation systems, the database is used to maintain the reservation and schedule information.
8) **Telecommunication :** In telecommunications for keeping records of the calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about communication networks the database systems are used.

1.2 Characteristics of Database Approach

Following are the characteristics of database system :

1) Representation of some aspects of **real world applications.**
2) Systematic **management of information.**

3) Representing the data by **multiple views.**
4) Efficient and easy implementation of various **operations** such as insertion, deletion and updation.
5) It maintains **data for** some specific **purpose.**
6) It represents **logical relationship** between records and data.

1.3 Advantages of using DBMS Approach

VTU : July-18, 19, Jan.-20, Marks 8

Following are the advantages of DBMS :

1) DBMS **removes the data redundancy** that means there is no duplication of data in database.
2) DBMS allows to **retrieve the desired data** in required format.
3) **Data** can be **isolated** in separate tables for convenient and efficient use.
4) Data can be **accessed efficiently** using a simple query language.
5) The **data integrity** can be maintained. That means – the constraints can be applied on data and it should be in some specific range.
6) The **atomicity** of data can be maintained. That means, if some operation is performed on one particular table of the database, then the change must be reflected for the entire database.
7) The DBMS allows **concurrent access** to multiple users by using the synchronization technique.
8) The **security policies** can be applied to DBMS to allow the user to access only desired part of the database system.

Disadvantages of Database Systems

1) **Complex design :** Database design is complex, **difficult and time consuming.**
2) **Hardware and software cost : Large amount of investment** is needed to setup the required hardware or to repair software failure.
3) **Damaged part :** If one part of database is corrupted or damaged, then **entire database** may get affected.
4) **Conversion cost :** If the current system is in conventional file system and if we need to convert it to database systems then large amount of cost is incurred in purchasing different tools, and adopting different techniques as per the requirement.
5) **Training :** For designing and maintaining the database systems, the people need to be trained.

1.3.1 File Processing System Vs. DBMS

- Earlier database systems are created in response to manage the commercial data. These data is typically stored in files. To allow users to manipulate these files, various programs are written for
  1) Addition of new data
  2) Updating the data
  3) Deleting the data.
- As per the need for addition of new data, separate application programs were required to write. Thus as the time goes by, the system acquires more files and more application programs.
- This typical **file processing system** is supported by conventional operating system. Thus the file processing system can be described as -
- "The system that stores the permanent records in files and it needs different application programs to extract or add the records".
- Before introducing database management system, this file processing system was in use. However, such a system has many drawbacks. Let us discuss them.

Disadvantages of Traditional File Processing System

The traditional file system has following disadvantages :

1) **Data redundancy :** Data redundancy means duplication of data at several places. Since different programmers create different files and these files might have different structures, there are chances that some information may appear repeatedly in some or more format at several places.
2) **Data inconsistency :** Data inconsistency occurs when various copies of same data may no longer get matched. For example changed address of an employee may be reflected in one department and may not be available (or old address present) for other department.
3) **Difficulty in accessing data :** The conventional file system does not allow to retrieve the desired data in efficient and convenient manner.
4) **Data isolation :** As the data is scattered over several files and files may be in different formats, it becomes difficult to retrieve the desired data from the file for writing the new application.
5) **Integrity problems :** Data integrity means data values entered in the database fall within a specified range and are of specific format. With the use of several files enforcing such constraint on the data becomes difficult.

6) **Atomicity problems :** An atomicity means particular operation must be carried out entirely or not at all with the database. It is difficult to ensure atomicity in conventional file processing system.
7) **Concurrent access anomalies :** For efficient execution, multiple users update data simultaneously, in such a case data need to be synchronized. As in traditional file systems, data is distributed over multiple files, one cannot access these files concurrently.
8) **Security problems :** Every user is not allowed to access all the data of database system. Since application program in file system are added in an ad hoc manner, enforcing such security constraints become difficult.

Database systems offer solutions to all the above mentioned problems.

Difference between Database System and Conventional File System

Sr. No.	Database systems	Conventional file systems
1.	Data redundancy is **less**.	Data redundancy is **more**.
2.	**Security is high.**	Security is very **low**.
3.	Database systems are used when security **constraints are high.**	Conventional file systems are used where there is **less demand** for **security constraints.**
4.	Database systems define the data in a **structured** manner. Also there is well defined co-relation among the data.	File systems define the data in **un-structured** manner. Data is usually in isolated form.
5.	Data **inconsistency** is **less** in database systems.	Data **inconsistency** is **more** in file systems.
6.	**User** is **unknown** to the physical address of the data used in database systems.	User locates the **physical address of file** to access the data in conventional file systems.
7.	We can retrieve the data in any **desired format** using database systems.	We cannot retrieve the data in any desired format using file systems.
8.	There is ability to access the data **concurrently** using database systems.	There is **no ability to concurrently** access the data using conventional file system.

University Questions

1. *Discuss the main characteristics of database approach and how it differs from traditional file system* **VTU : July-18, Marks 4**
2. *Define DBMS. Discuss the advantages of DBMS over the traditional file system.* **VTU : July-19, Marks 8**
3. *Compare DBMS and early file systems, bringing out the major advantages of the database approach.* **VTU : Jan.-20, Marks 6**

1.4 Actors

VTU : Jan.-19, 20, Marks 6

People who use and maintain the database systems are called actors. Normally various types of actors that maintain the database system are :

1) Database Administrator

- Database administrator is a person who manages the resources.
- The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.
- The DBA also handles the problems such as security breaches and poor system response time.

2) Database Designer

- Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
- The responsibility of database designers is to communicate with all database users in order to understand their requirements and create a design that meets these requirements.
- The database designers interact with the potential group of users and develop the **views of database**. With the help of these views, the processing of the requirements of these groups can be performed. Each view is analyzed and integrated with the views of other users group.
- Finally the database design must be created in such a way that the requirements of all the users of the group may get satisfied.

3) End Users

- End users are the users who interact with the database system. There are different types of end users and those are :
  - **Casual End Users :** These are the type of users who occasionally access the database and each time they require different type of information. These users use sophisticated query language to access the database systems.
  - **Naïve or Parametric Users :** These type of users constantly access or update the database system using standard queries called **canned transactions**. For example - the users for the ATM system will check the balance and withdraw some amount, the passenger may see the seat reservation status and may book the seats.
  - **Sophisticated End Users :** The engineers, scientists, business analysts are some sophisticated users who can easily understand the facilities of DBMS in order to implement their own applications to meet the complex requirements.

  - **Standalone Users :** These are the users who maintain personal databases by using ready-made program packages. For example – the user of tax package can store variety of personal financial data using specific software package.

4) System Analysts and Application Programmers

- System analysts identify the requirements of end users and develop **specifications** for standard canned transactions that meet their requirements.
- Application Programmers implement these **specifications as programs**. Then they test and debug these programs to maintain the canned transactions. These programmers are also called as **software developers** or **software engineers**.

5) Workers Behind the Scene : The workers behind the scene are the users who are not interested in database contents itself. There are following types of such users :

- **DBMS System Designers and Implementers :** These are the actors who design and implement the DBMS modules and interfaces as a software package.
- **Tool Developers :** Tools are software packages for software design, performance monitoring, graphics interfaces and so on. These are used for improved performance. Tool developers are the software developers who develop the software tools for the specific requirements.
- **Operators and Maintenance Personnel :** These are responsible for the actual running and maintenance of the hardware and software environment for the database system.

University Question

1. *What are the responsibilities of the DBA and database designer ?*

VTU : Jan.-19, 20, Marks 6

1.5 History of Database Application

1) Early Database Applications using Hierarchical and Network system :

- Many early database applications maintained records in large organizations such as corporations, universities, hospitals, and banks. In many of these applications, there were large numbers of records of similar structure.
- These early system applications were based on **hierarchical and network system**.
- In these applications, there was intermixing of conceptual relationships with the physical storage and placement of records on disk. This creates two major problems :
  - It **did not provide enough flexibility** to access records efficiently when new queries and transactions were identified.

- These type of applications were provided with only **programming language interfaces**, which is very **time consuming and expensive** to implement new queries and transactions.

2) Providing Data Abstraction and Application Flexibility with Relational Databases

- Relational database systems eliminated the drawbacks of the early database applications. That is, in relational database systems, **physical storage of data is separated from the conceptual representation.**
- The relational data models also introduced the **high level query language** instead of using programming language interfaces. This helps in **faster execution of new queries.**
- The relational database system implementation was started in 1970s and 1980s. Later on indexing techniques added to the implementation of relational database systems help in better query processing and optimization.

3) Object Oriented Applications and the Need for More Complex Databases

- After invention of object oriented programming language in 1980s, the development of object oriented databases has started. But due to complexity of this model and lack of early standards led to limited use of object oriented databases.
- The object oriented database systems are now used only in specialized applications such as engineering design, multimedia publishing and manufacturing systems.

4) Interchanging Data on Web for E-Commerce using XML

- The World Wide Web (WWW) provides a large network of interconnected computers. Users can create documents using **Hyper Text Markup Language (HTML)**, and store these documents on Web servers where other users can access them. Documents can be linked through **hyperlinks**, which are pointers to other documents.
- In 1990s E-commerce is a major application that is developed on Web.
- Currently, **eXtended Markup Language (XML)** is considered to be the primary standard for interchanging data among various types of databases and Web pages.

5) Extending Database Capabilities for New Applications

- Database systems now offer extension to better support the specialized requirements for some of the applications. These applications are –
  - **Images :** Storage and retrieval of images including, personal images, satellite images or images from X-rays and MRIs.

  - **Videos :** Storage and retrieval of videos such as movies and videos from cameras.
  - **Scientific Applications :** Scientific applications such as mapping of the human genome, and the discovery of protein structures.
  - **Data Mining :** Data mining applications that analyze large amounts of data searching for the occurrences of specific patterns.
  - **Time Series Applications : Time series** applications that store information such as economic data at periodic times.
  - **Spatial Applications :** Spatial applications that store spatial locations of data, such as weather information, maps used in geographical information systems, and in automobile navigational systems.

6) Information Retrieval (IR) Based Systems

- Database technology is heavily used in manufacturing, retail, banking, insurance, finance, and health care industries, where structured data is collected through forms, such as invoices or patient registration documents. An area related to database technology is Information Retrieval (IR).
- Data is indexed, cataloged, and annotated using keywords.IR is concerned with searching for material based on these keywords.
- Retrieval of information on the Web is a new major concern that requires techniques from databases and IR.

Part II : Overview of Database Languages and Architectures

1.6 Data Models, Schema and Instances

VTU : Jan.-20, Marks 5

1.6.1 Data Models

The data model in database applications describe the logical structure of database, relationship between the database stored in database and various constraints on data.

Importance of Data Model

1. End users have different view for data.
2. Data model organizes data for different users.

- There are three types of data models that are used commonly :

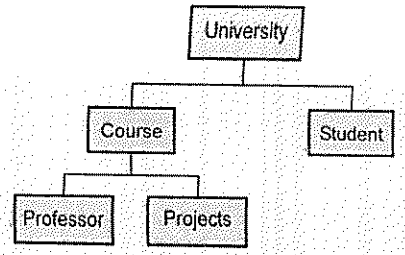


Fig. 1.6.1 Hierarchical model

1. Hierarchical Model :

- In this model each entity has only one parent but can have several children. At the top of hierarchy there is only one node called **root**. Refer Fig. 1.6.1.

- This model represents the relationship in 1 : N types. That means one university can have multiple courses. One course can have multiple projects and so on.

Advantage

1. This model groups the data into tables and defines the relationship between the tables.

Disadvantages

1. For searching any data, we have to start form the root and move downwards and visit each child node. Thus traversing through each node is required.
2. For addition of some information about child node, sometimes the parent information need to be modified.
3. It fails to handle many to many relationship (M : N) efficiently.
   It can cause duplication and data redundancy.

2. Network Model :

- This is enhanced version of hierarchical model. It overcomes the drawback of hierarchical model. It helps to address M : N relationship. That means, this model is not having single parent concept. Any child in this model can have multiple parents. Refer Fig. 1.6.2.
- The **main difference** between network model and hierarchical model is to allow many to many relationship.

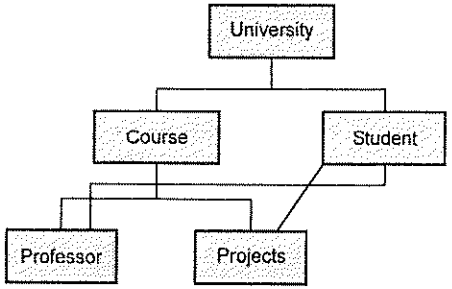


Fig. 1.6.2 Network model

Advantages

1. **Capability to handle more Relationships :** Since the network model allows many to many relationship, it helps in modeling the real life situations.
2. **Ease of data access :** The data access is easier and flexible than hierarchical model.
3. **Data Integrity :** In network model every member is associated with some other member in the model.
4. **Conformance to Standards :** The network model structure can be designed as per the standards.

Disadvantages

1. **Complex to implement :** For all the records the pointers need to be maintained, hence the database structure becomes complex.

2. **Complicated Operations :** The simple operations such as insertion, deletion and modification becomes complex due to adjustment of multiple pointer.
3. **Difficult to change structure :** The structural changes are difficult.

3. Relational Model

- It is developed by Codd in 1970. In relational model, the data is stored in the form of tables. The database systems based on it are called relational database systems.
- These tables are related with each other. Refer Fig. 1.6.3.

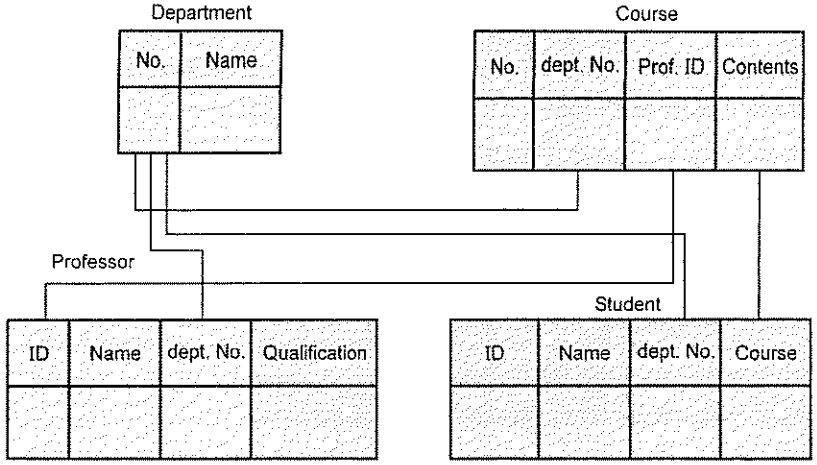


Fig. 1.6.3 Relational model

Advantages

1. The database design is simple to implement and manage.
2. In this model, the table is an entity which is primarily used. Hence insertion, deletion and retrieval of data becomes easy.
3. The data can be easily linked with other table, using the table attributes.
4. It facilitates the support for SQL languages.
5. It ensures the structural independence.
6. Even after creation of database, new categories of data can be inserted without making changes in the existing application.

Disadvantages

1. It has substantial hardware and software overhead.
2. It can facilitate to poor design and implementation.
3. It can not handle the data in the form of images, audio or video.

4. E-R Model :

- This model is basically used to design the relational database systems. The primary purpose of E-R Model is to show the relationship between various data objects.
- The object relationship pair can be graphically represented by a diagram called **Entity Relationship Diagram (ERD)**.
- The ERD was originally proposed by Peter Chen for design of relational database systems.
- Various **components of ERD** are -

Entity

- Drawn as a rectangle.
- An entity is an object that exists and is distinguishable.
- Similar to a record in a programming language with attributes.

Relationship

- Drawn as a diamond.
- An association among several entities.
- Relationships may have attributes.
- Relationships have cardinality (e.g., one-to-many).

Attribute

- Drawn as ellipses.
- Similar to record fields in a programming language.
- Each attribute has a set of permitted values, called the domain.
- Primary key attributes may be underlined.

Advantages

1. **Simple to design :** Conceptually creating ER model is conceptually very simple when one knows the various types of entities, attributes and relationships among them.
2. **Visual Representation :** This model gives clear graphical and diagrammatical representation of various entities, attributes and the relationships.
3. **Easy Conversion :** Converting an ER Model to tables is very simple. Also ER model can be easily converted to hierarchical model, network model, or relational model.

Disadvantages

1. **Limited Specification :** Using ER Model only binary relationship can be represented. Thus the model represents minimum specification.
2. **For High Level Design :** This model is popularly used for high level design only.
3. **No Industry Standard :** There is no industry standard notation for developing ER Model.

1.6.2 Schema and Instances

Schema : The overall design of the database is called schema

For example - In a program we do variable declaration and assignment of values to the variable. The variable declaration is called schema and the value assigned to the variable is called instance. The schema for the student record can be

RollNo	Name	Marks

Instances : When information is inserted or deleted from the database then the database gets changed. The collection of information at particular moment is called **instances**. For example - following is an instance of **student** database.

RollNo	Name	Marks
10	AAA	43
20	BBB	67

Types of Schema : The database has several schema based on the levels of abstraction.

1. **Physical Schema :** The physical schema is a database design described at the physical level of abstraction.
2. **Logical Schema :** The logical schema is a database design at the logical level of abstraction.
3. **Subschema :** A database may have several views at the view level which are called subschemas.

University Question

1. Define the following terms –
1) Data Model 2) Schema 3) instance 4) Canned transactions
VTU : Jan.-20, Marks 8

1.7 Three Schema Architecture and Data Independence

VTU : July-18, Jan.-19, 20, Marks 10

Three Schema Architecture

- **Definition :** Database schema is a collection of database objects like tables, views, indexes and so on associated with one particular database username. This username is called the **schema owner**.
- For example Student Schema can be owner of STUDENT and MARKS tables. The Course schema can be the owner of SUBJECT table.
- The goal of three-schema architecture is to separate the user application from the physical database.
- The architecture of database is divided into **three levels** based on three types of schema - internal schema, conceptual schema or external schema.

1. Internal level :

- It contains **internal schema**.
- This schema represents the **physical storage structure of database**.
- This schema is maintained by the software and user is not allowed to modify it.
- This level is closest to the physical storage. It typically describes the record layout of the files and types of files, access paths etc.

2. Conceptual level :

- It contains **conceptual schema**.
- This schema **hides the details** of internal level.
- This level is also called as logical level as it contains the constructs used for designing the database.
- It contains information like table name, their columns, indexes and constraints, database operations.
- A representational data model is used to describe conceptual schema when a database system is implemented.

3. External level :

- It contains the external schema or user views.
- At this level, the user will get to see only the data stored in the database. Either they will see whole data values or any specific records. They will not have any information about how they are stored in the database.

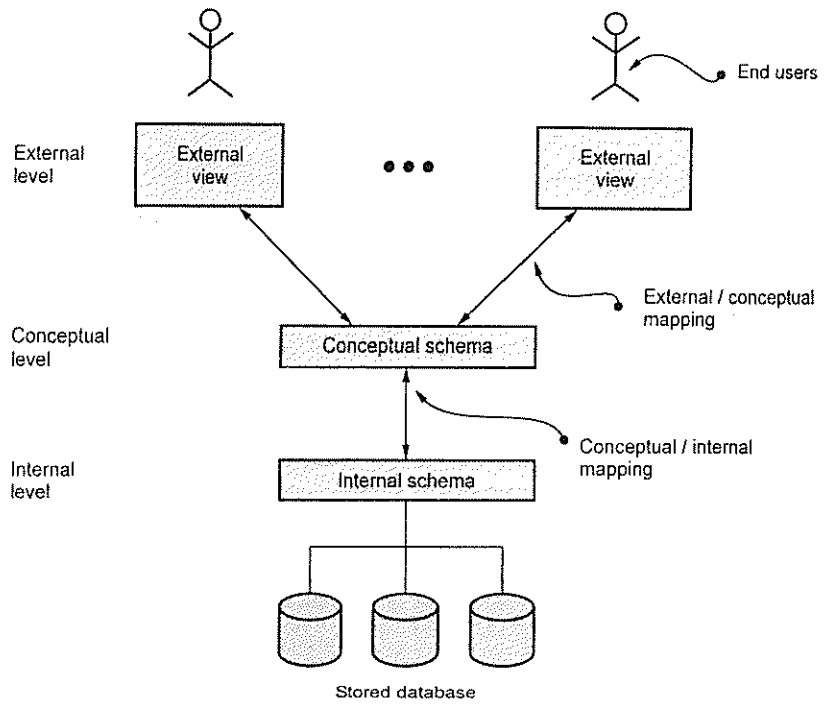


Fig. 1.7.1 Three schema architecture

- The processes of transforming requests and results between levels are called **mappings**.
- In the three schema architecture there are two mappings –

  1) External - Conceptual Mapping and

  2) Conceptual - Internal Mapping

Data Independence

- **Definition :** Data independence is an ability by which one can change the data at one level without affecting the data at another level. Here level can be physical, conceptual or external.
- Data independence is one of the important characteristics of database management system.
- By this property, the structure of the database or the values stored in the database can be easily modified by without changing the application programs.

- There are two types of data independence.

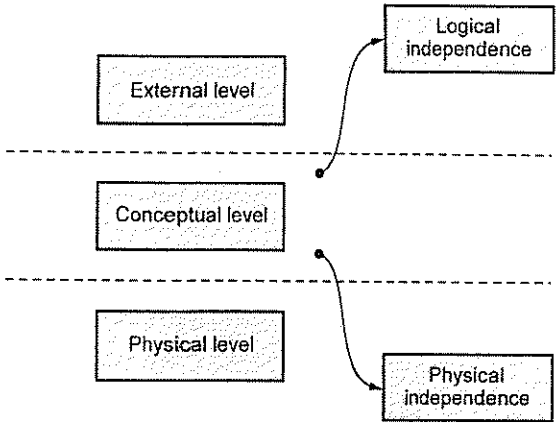


Fig. 1.7.2 Data independence

1. Physical independence :

- This is a kind of data independence which allows the modification of physical schema without requiring any change to the conceptual schema.
- For example - If there is any change in memory size of database server then it will not affect the logical structure of any data object.

2. Logical independence :

- This is a kind of data independence which allows the modification of conceptual schema without requiring any change to the external schema.
- For example - Any change in the table structure such as addition or deletion of some column does not affect user views.
- By these data independence the time and cost acquired by changes in any one level can be reduced and abstract view of data can be provided to the user.

University Questions

1. Describe the three schema architecture. Why do we need mapping among schema levels?
VTU : July-18, Jan.-19, Marks 4

2. With neat block diagram, explain the architecture of a typical DBMS.
VTU : Jan.-20, Marks 10

1.8 Database Languages and Interfaces

1.8.1 Database Languages

- Query is a statement used for requesting the retrieval of information. This retrieval of information using some specific language is called **query language**.
- There are three types of database languages.

1) Data Definition Language

- Data Definition Language (DDL) is a specialized language used to specify a database schema by a set of definitions.
- It is a language which is used **for creating** and modifying the structures of tables, views, indexes and so on.
- DDL is also used to specify additional properties of data.
- Some of the common commands used in DDL are - **CREATE, ALTER, DROP.**
- The main use of CREATE command is to build a new table. Using ALTER command, the users can add up some additional column and drop existing columns. Using DROP command, the user can delete table or view.

2) Data Manipulation Language

- This Data Manipulation Language (DML) enables users to access or manipulate data as organized by appropriate data model.
- The types of access are -
  - Retrieval of information stored in the database
  - Insertion of new information into the database.
  - Deletion of information from the database.
  - Modification of information stored in database.
- There are two types of DML -
  - **Procedural DML** - Require a user to specify what data are needed and how to get those data.
  - **Declarative DML** - Require a user to specify what data are needed without specifying how to get those data.

3) Data Control Language

- The Data Control Language (DCL) is used to control access to data stored in the database. This is also called as authorization.
- The typical command used in DCL are GRANT and REVOKE.

- **GRANT :** This command is used to give access rights or privileges to the database.
- **REVOKE :** The revoke command removes user access rights or privileges to the database objects.

1.8.2 DBMS Interfaces

There are various ways by which the user friendly interfaces can be created. Following are the types of DBMS interfaces :

1) Menu based Interfaces for Web Clients

- Menus are basically the lists of commands that user wants to use while accessing the information from the database. Due to menu based interface, user need not have to remember formal queries, simply by selecting the options from the menu particular activity for the database can be performed.
- **Pull down menus** are popularly used in web based user interfaces. They are also used in browsing interfaces.

2) Form Based Interfaces

- In the form based interfaces, the form is presented to each user. User fill up the form by entering new data.
- Forms are usually designed in such a way that any native user can use it easily to perform canned transactions.
- Many DBMS have the form specification languages that help the user to specify the form.
- For example - Oracle forms is a component of Oracle product suite that help in creating the form.

3) Graphical User Interface (GUI)

- Using GUI the graphical user interface can be displayed in diagrammatic form.
- User can specify the query by manipulating the diagram.
- For selection of any part of the diagram GUI supports for the use of pointing device such as mouse.

4) Natural Language Interface

- This kind of interface allow the user to enter the request in English or in some natural language.
- A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words.

- The natural language interface refers to the words in its schema, as well as to the set of standard words in its dictionary, to **interpret the request**.
- If the interpretation is successful, the interface generates the high level query and submits it to DBMS.

5) Speech Input and Output

- Speech can be used to input the query and to get the result from the DBMS.
- Applications such as telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access the information.
- The speech input is detected using a library of **predefined words** and used to set up the parameters that are supplied to the queries.

6) Interfaces for Parametric users

- **Concept of Canned transaction :** These type of users constantly access or update the database system using standard queries called **canned transactions**. For example - the users for the ATM system will check the balance and withdraw some amount.
- These users require small set of operations that are repeatedly performed.
- Systems analysts and programmers design and implement a special interface for each known class of Naive users. Usually a small set of abbreviated commands is included.

7) Interfaces for the DBA

- DBA is a special type of user for any DBMS system because there are some privileged commands that can be used only by DBA.
- Such commands for creating accounts, setting system parameters, granting account authorization, changing schema and so on.

1.9 The Database System Environment

VTU - July-18, 19, Jan.-19, Marks 8

- The environment of typical DBMS is based on relational data model as shown in Fig. 1.9.1. (See Fig. 1.9.1 on next page).
- Consider the top part of Fig. 1.9.1. It shows **application interfaces** used by Naïve users, application programs created by application programmers, query tools used by sophisticated users and administration tools used by database administrator.
- The lowest part of the architecture is for **disk storage**.

- The two important components of database architecture are - **Query processor and storage manager.**

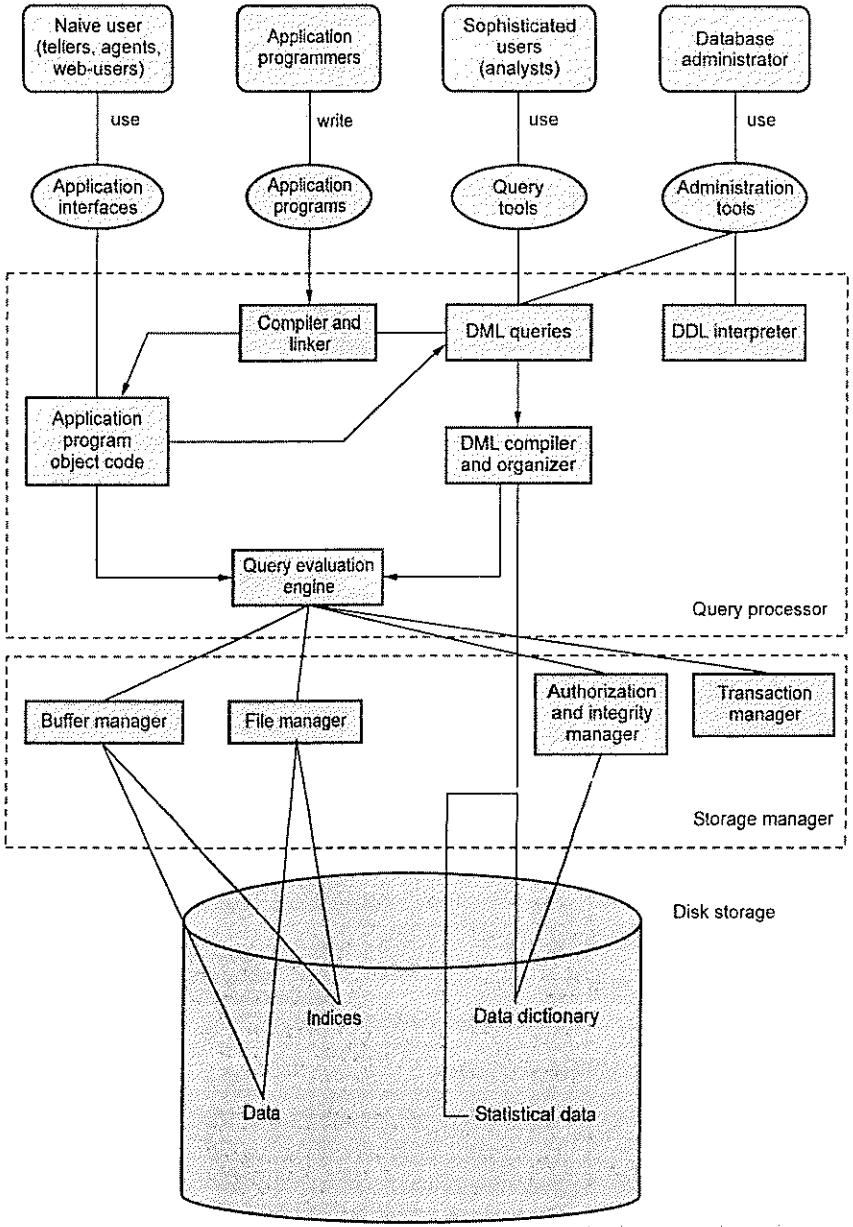


Fig. 1.9.1 Architecture of database

Query processor :

- The interactive query processor helps the database system to simplify and facilitate access to data. It consists of DDL interpreter, DML compiler and query evaluation engine.
- With the following components of query processor, various functionalities are performed -
  - i) **DDL interpreter :** This is basically a translator which interprets the DDL statements in data dictionaries.
  - ii) **DML compiler :** It translates DML statements query language into an evaluation plan. This plan consists of the instructions which query evaluation engine understands.
  - iii) **Query evaluation engine :** It executes the low-level instructions generated by the DML compiler.
- When a user issues a query, the parsed query is presented to a query optimizer, which uses information about how the data is stored to produce an efficient execution plan for evaluating the query. An execution plan is a blueprint for evaluating a query. It is evaluated by **query evaluation engine**.

Storage manager :

- Storage manager is the component of database system that provides interface between the low level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for **storing, retrieving,** and **updating data** in the database. The storage manager components include -
  - i) **Authorization and integrity manager :** Validates the users who want to access the data and tests for integrity constraints.
  - ii) **Transaction manager :** Ensures that the database remains in **consistent state** despite of system failures and concurrent transaction execution proceeds without conflicting.
  - iii) **File manager :** Manages **allocation of space** on disk storage and representation of the information on disk.
  - iv) **Buffer manager :** Manages the fetching of data from **disk storage into main memory.** The buffer manager also decides what data to cache in main memory. Buffer manager is a crucial part of database system.

- Storage manager implements several **data structures** such as -
  - i) **Data files :** Used for **storing database** itself.
  - ii) **Data dictionary :** Used for **storing metadata**, particularly schema of database.
  - iii) **Indices :** Indices are used to provide **fast access** to data items present in the database.

University Questions

1. *Discuss various components of a DBMS with neat diagram*
   VTU : July-18, 19 Marks 8
2. *Discuss the different types of user friendly interfaces and the type of user who typically use each*
   VTU : Jan.-19, Marks 5

Part III : Conceptual Data Modeling using Entities and Relationships

1.10 Introduction to Entity Relationship Modeling

VTU : Jan.-19, Marks 8

Entity Relational model is a model for identifying entities to be represented in the database and representation of how those entities are related.

Let us first understand the design process of database design.

1.10.1 Design Phases

Following are the six steps of database design process. The ER model is most relevant to first three steps.

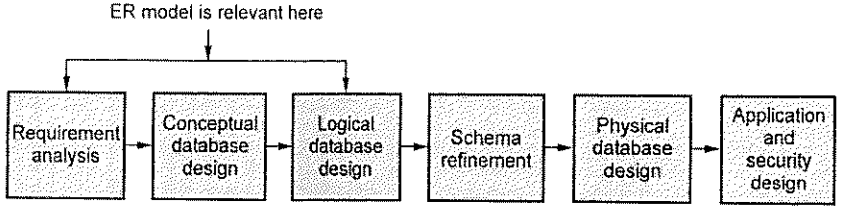


Fig. 1.10.1 Database design process

Step 1 : Requirement analysis :

- In this step, it is necessary to understand what data need to be stored in the database, what applications must be built, what are all those operations that are frequently used by the system.
- The requirement analysis is an **informal** process and it requires proper **communication** with user groups.

- There are several methods for **organizing and presenting information** gathered in this step.
- Some **automated tools** can also be used for this purpose.

Step 2 : Conceptual database design :

- This is a steps in which **E-R Model** i.e. Entity Relationship model is built.
- E-R model is a **high level data model** used in database design.
- The **goal** of this design is to create a simple description of data that matches with the requirements of users.

Step 3 : Logical database design :

- This is a step in which ER model in **converted to relational database schema**, sometimes called as the logical schema in the relational data model.

Step 4 : Schema refinement :

- In this step, **relational database schema is analyzed** to identify the potential problems and to refine it.
- The schema refinement can be done with the help of **normalizing and restructuring the relations**.

Step 5 : Physical database design :

- In this step, the design of database is **refined further**.
- The tasks that are performed in this step are - building **indexes** on tables and **clustering** tables, redesigning some parts of schema obtained from earlier design steps.

Step 6 : Application and security design :

- Using design methodologies like UML (Unified Modeling Language) the design of the database can be accomplished.
- The **role of each entity** in every process must be reflected in the application task.
- For each role, there must be the provision for **accessing** the some part of database and **prohibition of access** to some other part of database.
- Thus some **access rules** must be enforced on the application (which is accessing the database) to protect the **security features**.

University Question

1. *Explain with block diagram the different phases of database design.*
   VTU : Jan.-19, Marks 8

1.11 Entity Types, Entity Sets, Attributes, Roles and Structural Constraints

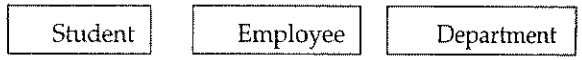
VTU : July-18, Marks 6

The ER data model specifies enterprise schema that represents the overall logical structure of a database.

The E-R model is very useful in mapping the meanings and interactions of real-world entities onto a conceptual schema.

The ER model consists of following basic concepts :

1) Entity : An entity is an object that exists and is distinguishable from other objects. For example - Student named "Poonam" is an entity and can be identified by her name. The entity can be concrete or abstract. The concrete entity can be - Person, Book, Bank. The abstract entity can be like - holiday, concept entity is represented as a box.



2) Entity set : The entity set is a set of entities of the same types. For example - All students studying in class X of the School. The entity set need not be disjoint. Each entity in entity set have the same set of attributes and the set of attributes will distinguish it from other entity sets. No other entity set will have exactly the same set of attributes.

3) Attribute

Attributes define the properties of a data object of entity. For example if student is an entity, his ID, name, address, date of birth, class are its attributes. The attributes help in determining the unique entity. Refer Fig. 1.11.1 for Student entity set with attributes - ID, name, address. Note that entity is shown by rectangular box and attributes are shown in oval. The primary key is underlined.

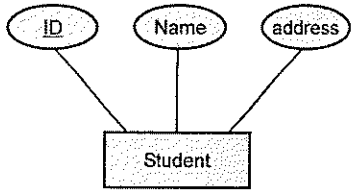


Fig. 1.11.1 Student entity set with attributes

Types of Attributes

1) Simple and Composite Attributes :

1) Simple attributes are attributes that are drawn from the atomic value domains
   For example - Name = {Parth} ; Age = {23}
2) Composite attributes: Attributes that consist of a hierarchy of attributes
   For example - Address may consists of "Number", "Street" and "Suburb"
   → Address = {59 + 'JM Road' + 'ShivajiNagar'}

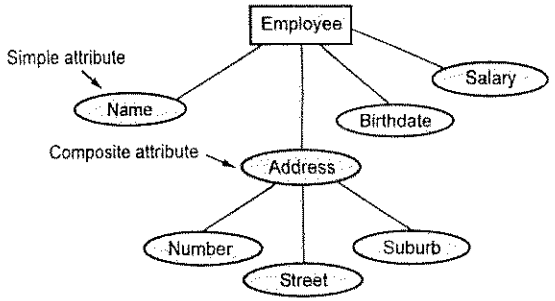


Fig. 1.11.2

2) Single valued and multivalued :

- There are some attributes that can be represented using a single value. For example - StudentID attribute for a Student is specific only one studentID.
- Multivalued attributes : Attributes that have a set of values for each entity. It is represented by concentric ovals.
  For example - Degrees of a person : ' BSc' , 'MTech', 'PhD'.

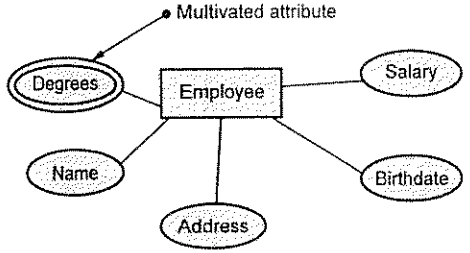


Fig. 1.11.3

3) Derived attribute :

Derived attributes are the attributes that contain values that are calculated from other attributes. To represent derived attribute there is dotted ellipse inside the solid ellipse. For example – Age can be derived from attribute DateOfBirth. In this situation, DateOfBirth might be called Stored Attribute.

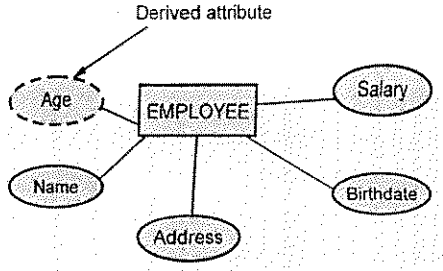


Fig. 1.11.4

4) Entity Types

An entity type defines a collection of entities that have the same attributes

Entities are classified as follows :

1) Strong entity Set.

2) Weak entity Set.

3) Recursive entities.

4) Composite entities.

Let us understand them,

Weak entity set : A weak entity is an entity that cannot be uniquely identified by its attributes alone. The entity set which does not have sufficient attributes to form a primary key is called as weak entity set.

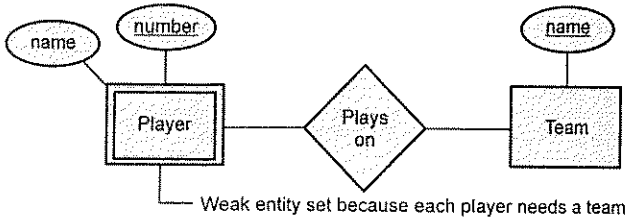


Fig. 1.11.5 Weak entity set

Strong entity set : The entity set that has primary key is called as strong entity set. For example Student is a strong entity set as it has a Roll_no as its primary key.

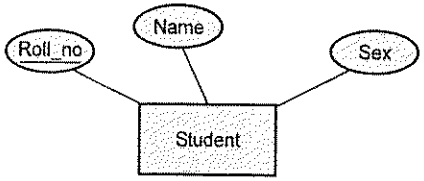


Fig. 1.11.6

Recursive entities : If a relation exists between the same entities, then such entities are called as recursive entity. For example, mapping between manager and employee is recursive entity.

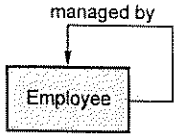


Fig. 1.11.7

- **Composite entities :** Entities participating in the many to many relationships are called composite entity. In case of a composite entity will always have a composite key that is the combination of the other two table's key, For example – if the Class entity has a classID, and the Student entity has a stuID, the composite entity's key will be ClassID and StuID.

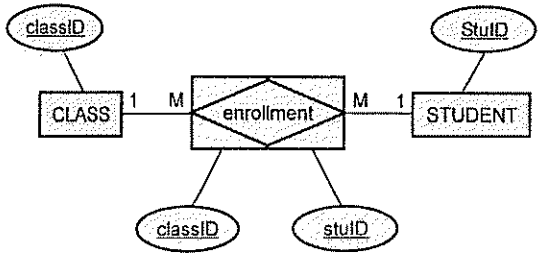


Fig. 1.11.8

5) Relationship sets and Roles :

Relationship is an association among two or more entities.

The **relationship set** is a collection of similar relationships. For example - Following Fig. 1.11.9 shows the relationship **works_for** for the two entities **Employee** and **Departments**.

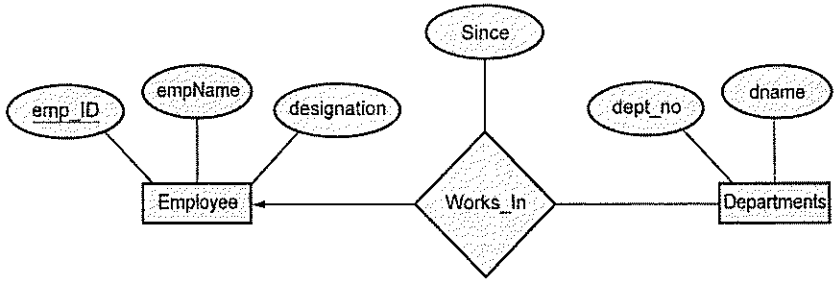


Fig. 1.11.9 Relation set

The association between entity sets is called as **participation**. That is, the entity sets E1, E2, . . . , En participate in relationship set R.

The function that an entity plays in a relationship is called that entity's **role**.

Each entity type that participates in a relationship type plays a particular role in the relationship. The role name signifies the role of that entity. For example - Role can be Employee, Student, Worker, Customer and so on. The role of an entity gives meaning to the relationship.

6) Structural Constraints :

Relationship types have certain rules that limit the possible combination of entities that can take part in relationship. These rules or restrictions are called structural constraints. The common type of structural constraint is represented by the cardinality ratio. The **cardinality ratio** for a binary relationship specifies the **maximum** number of relationship instances that an entity can participate in.

7) Types of Cardinality

Mapping Cardinality represents the number of entities to which another entity can be associated via a relationship set.

The mapping cardinalities are used in representing the binary relationship sets.

Various types of mapping cardinalities are -

1) **One to One :** An entity A is associated with at least one entity on B and an entity B is associated with at one entity on A. This can be represented as

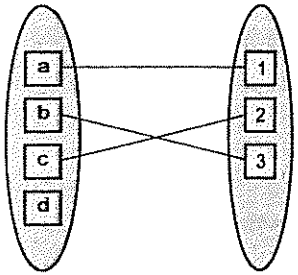


Fig. 1.11.10

2) **One to Many :** An entity in A is associated with any number of entities in B. An entity in B, however, can be associated with at most one entity in A.

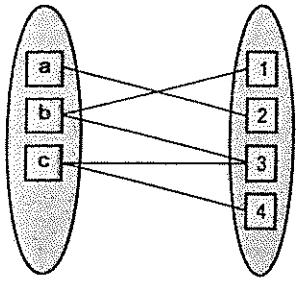


Fig. 1.11.11

3) **Many to One :** An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number of entities in A.

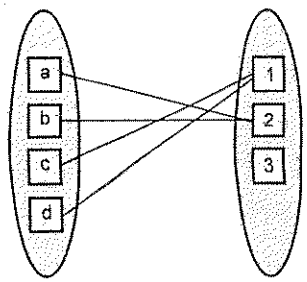


Fig. 1.11.12

4) **Many to many :** An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.

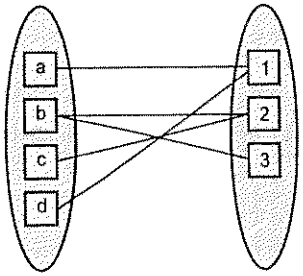


Fig. 1.11.13

University Question

1. *Define an entity and attribute. Explain different types of attributes that occur in an ER diagram model with an example.* **VTU : July-18, Marks 6**

1.12 Weak Entity Types

VTU : July-19, Marks 8

- A weak entity is an entity that cannot be uniquely identified by its attributes alone. The entity set which does not have sufficient attributes to form a primary key is called as weak entity set.

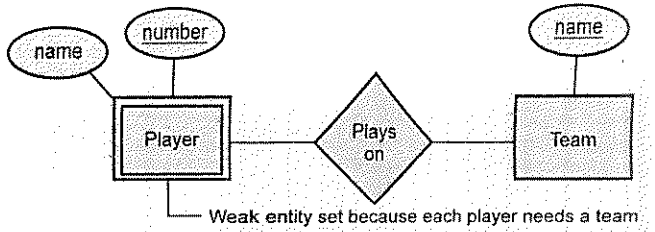


Fig. 1.12.1 : Weak entity set

Strong Entity Set

The entity set that has primary key is called as strong entity set.

Weak entity rules

- A weak entity set has one or more many-one relationships to other (supporting) entity sets.
- The key for a weak entity set is its own underlined attributes and the keys for the supporting entity sets. For example - player-number and team-name is a key for Players.

Difference between Strong and Weak Entity Set

Sr. No.	Strong entity set	Weak entity set
1	It has its own primary key.	It does not have sufficient attribute to form a primary key on its own.
2.	It is represented by rectangle.	It is represented by double rectangle.
3.	It represents the primary key which is underlined.	It represents the partial key or discriminator which is represented by dashed underline.
4.	The member of strong entity set is called as dominant entity set.	The member of weak entity set is called subordinate entity set.
5.	The relationship between two strong entity sets is represented by diamond symbol.	The relationship between strong entity set and weak entity set is represented by double diamond symbol.
6.	The primary key is one of the attributes which uniquely identifies its member.	The primary key of weak entity set is a combination of partial key and primary key of the strong entity set.

University Question

1. Define following with an example :

i) Weak Entity Type *ii) Participation constraints*

iii) Cardinality ratio *iv) Recursive relationship*

VTU : July-19, Marks 8

1.13 ER Diagram

An E-R diagram can express the overall logical structure of a database graphically. E-R diagrams are used to model real-world objects like a person, a car, a company and the relation between these real-world objects.

Features of ER model

i) E-R diagrams are used to represent E - R model in a database, which makes them easy to be converted into relations (tables).

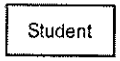
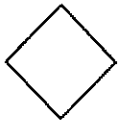
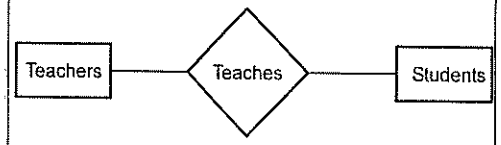
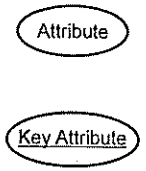
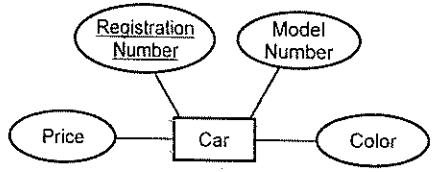
ii) E-R diagrams provide the purpose of real - world modeling of objects which makes them intently useful.

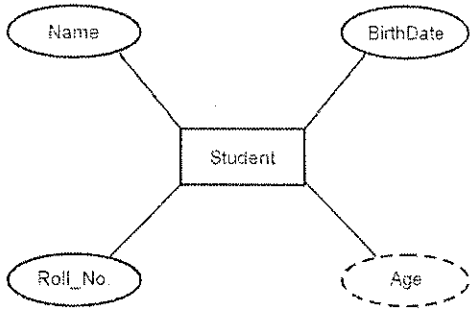
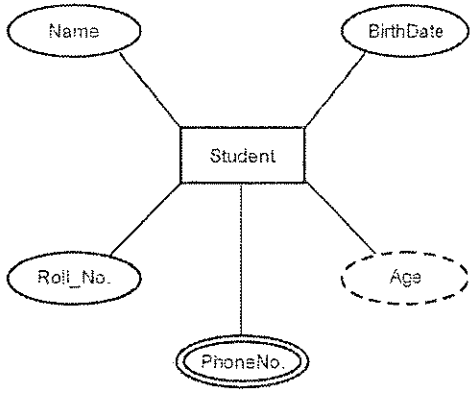
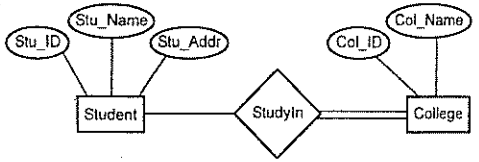
iii) E-R diagrams require no technical knowledge and no hardware support.

iv) These diagrams are very easy to understand and easy to create even by a naive user.

v) It gives a standard solution of visualizing the data logically.

Various Components used in ER Model are -

Component	Symbol	Example
Entity : Any real-world object can be represented as an entity about which data can be stored in a database. All the real world objects like a book, an organization, a product, a car, a person are the examples of an entity.		
Relationship : Rhombus is used to setup relationships between two or more entities.		
Attribute : Each entity has a set of properties. These properties of each entity are termed as attributes. For example, a car entity would be described by attributes such as price, registration number, model number, color etc		

Derived attribute : Derived attributes are those which are derived based on other attributes, for example, age can be derived from date of birth. To represent a derived attribute, another dotted ellipse is created inside the main ellipse		
Multivalued attribute : An attribute that can hold multiple values is known as multivalued attribute. We represent it with double ellipses in an E-R Diagram. E.g. A person can have more than one phone numbers so the phone number attribute is multivalued.		
Total participation : Each entity is involved in the relationship. Total participation is represented by double lines.		

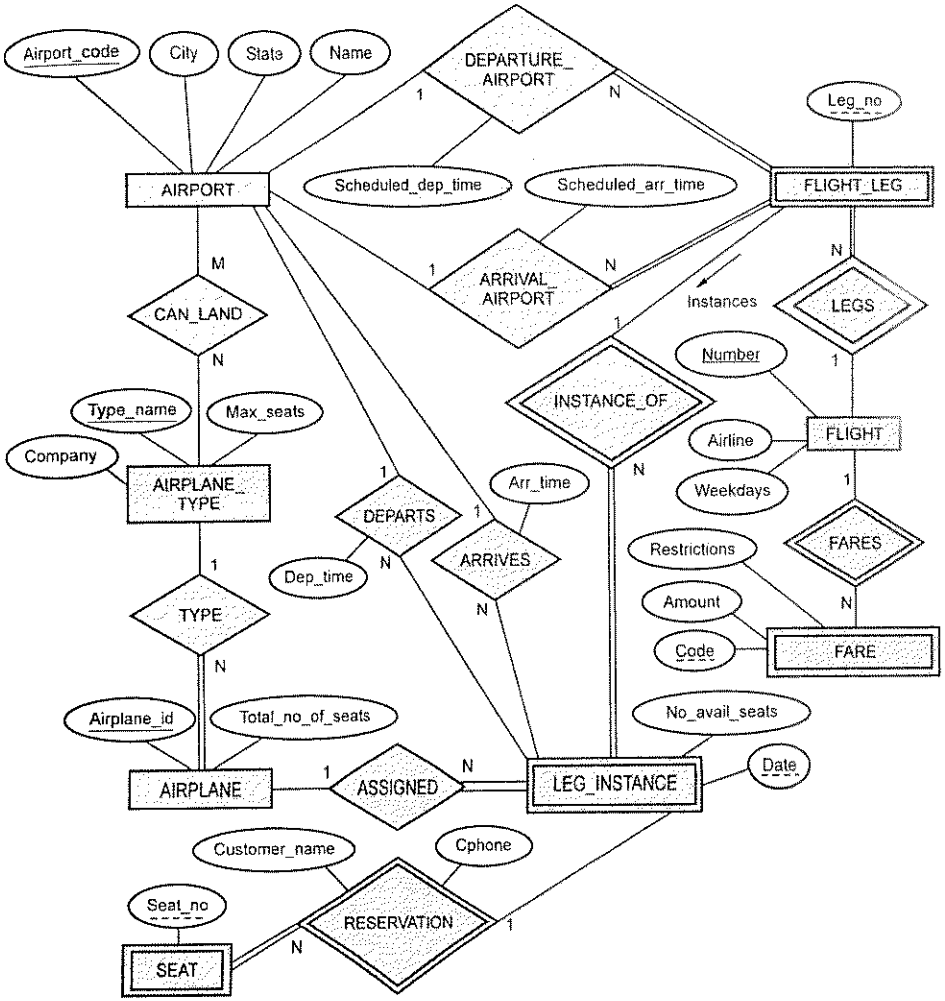
1.14 Examples of ER Diagrams

VTU : July-18,19, Jan.-19,20, Marks 12

Example 1.14.1 *Draw an ER diagram of Airline reservation system taking into account at least five entities. Indicate all keys, constraints and assumptions that are made.*

VTU : July-18, Marks 10

Solution :

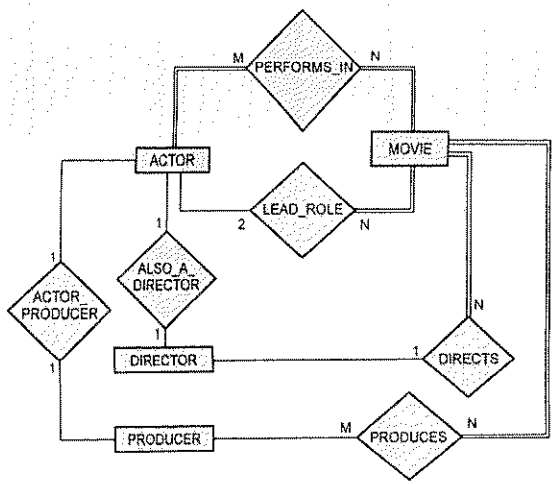


A LEG is nonstop portion of flight. The LEG_INSTANCE is a particular occurrence of a LEG on particular date.

Example 1.14.2 *Draw an ER diagram of movie database. Assume your own entities (minimum 4) attributes and relationships.*

VTU : Jan.-19, Marks 8

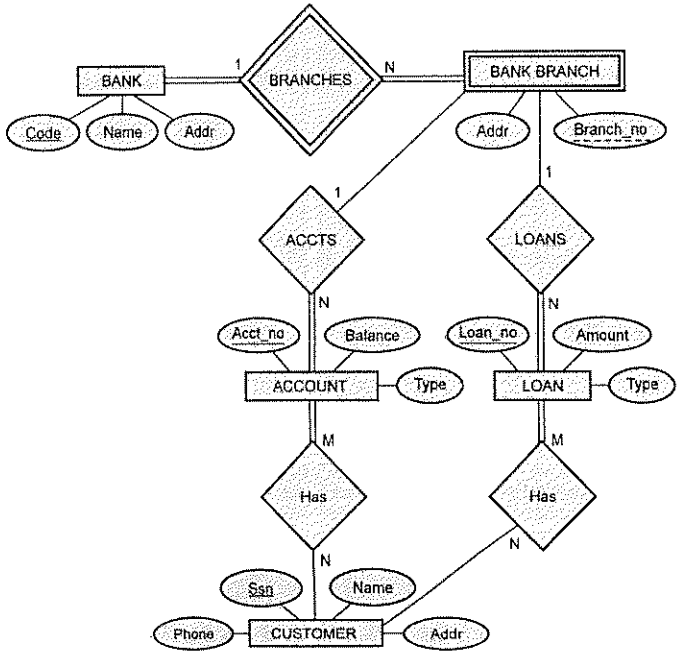
Solution :



Example 1.14.3 *Draw an ER diagram of Banking system taking into account at least five entities, indicate all keys, constraints and assumptions that are made.*

VTU : July-19, Marks 8

Solution :



Example 1.14.4 *Draw an ER diagram to represent the Election Information system based on the following description.*

In Indian national election, a state is divided into a number of constituencies depending upon the population of the state. Several candidates contest elections in each constituency. Record the number of votes obtained by each candidate. The system also maintains the voter list and a voter normally belongs to a particular constituency.

Note that the party details must also be taken care in the design.

VTU : Jan.-20, Marks 12

Solution :

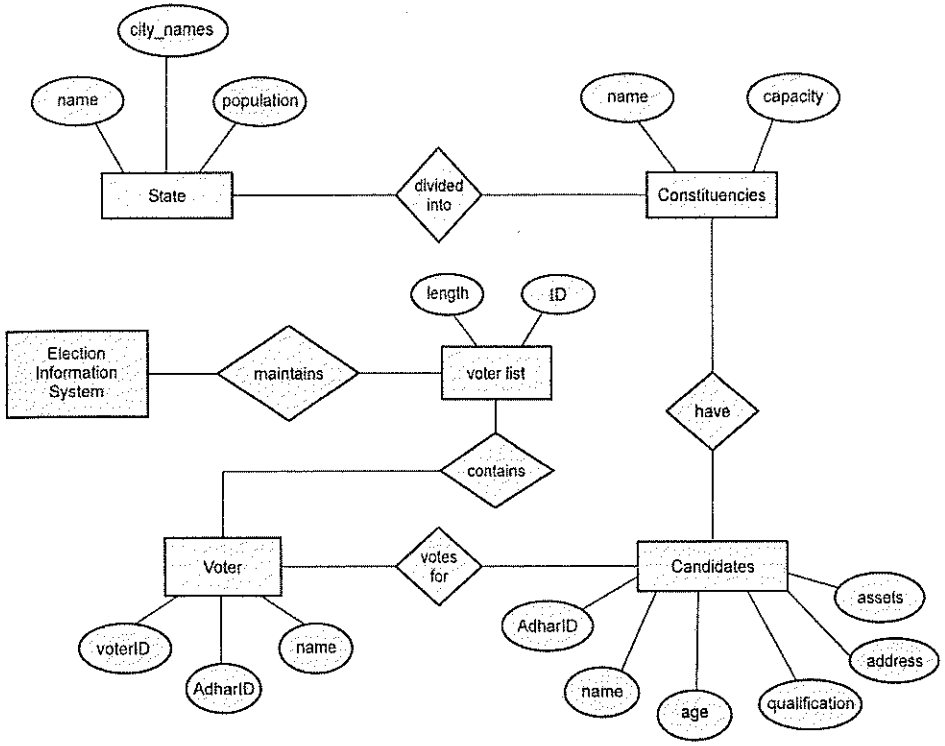
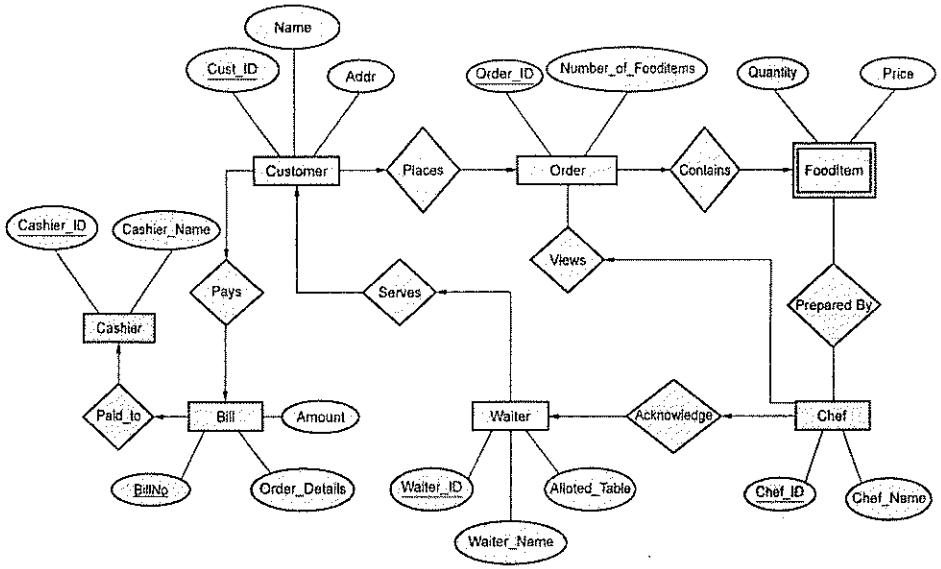


Fig. 1.14.1 Election information system

Example 1.14.5 *Draw E-R diagram for the "Restaurant Menu Ordering System", which will facilitate the food items ordering and services within a restaurant. The entire restaurant scenario is detailed as follows. The customer is able to view the food items menu, call the waiter, place orders and obtain the final bill through the computer kept in their table. The Waiters through their wireless tablet PC are able to initialize a table for customers, control the table functions to assist customers, orders, send orders to food preparation staff (chef) and*

finalize the customer's bill. The Food preparation staffs (chefs), with their touch-display interfaces to the system, are able to view orders sent to the kitchen by waiters. During preparation they are able to let the waiter know the status of each item, and can send notifications when items are completed. The system should have full accountability and logging facilities, and should support supervisor actions to account for exceptional circumstances, such as a meal being refunded or walked out on.

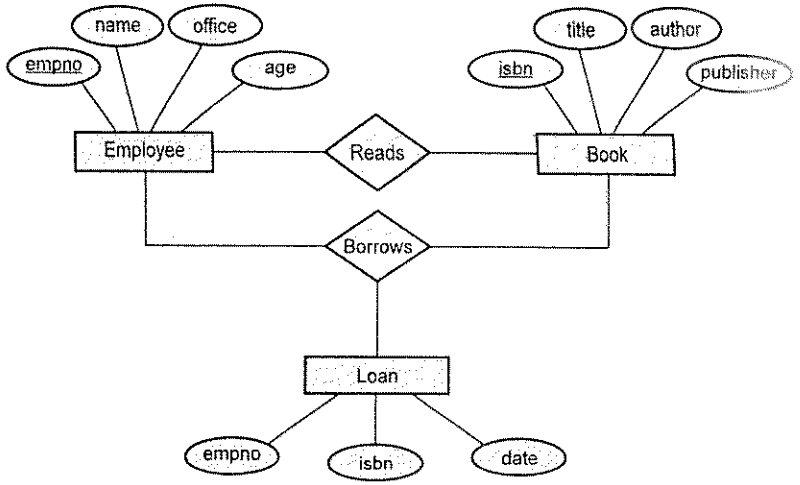
Solution :



Example 1.14.6 *Consider the relation schema given in Figure. Design and draw an ER diagram that capture the information of this schema.*

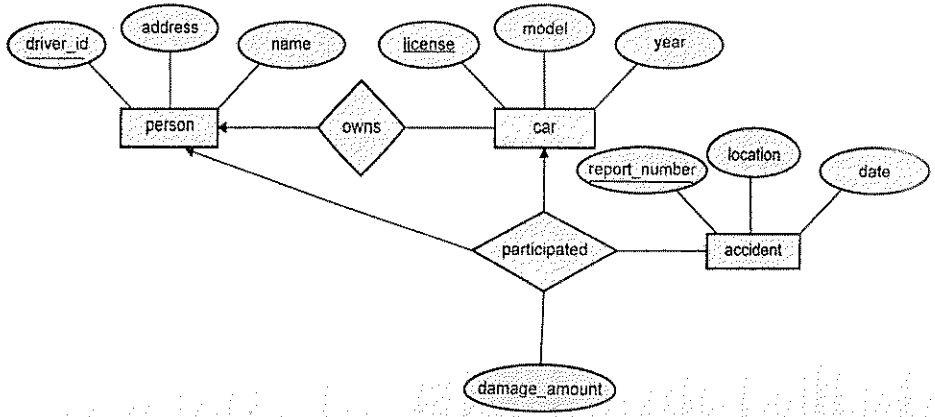
Employee (empno, name, office, age)
Books (isbn, title, authors, publisher)
Loan (empno, isbn, date)

Solution :



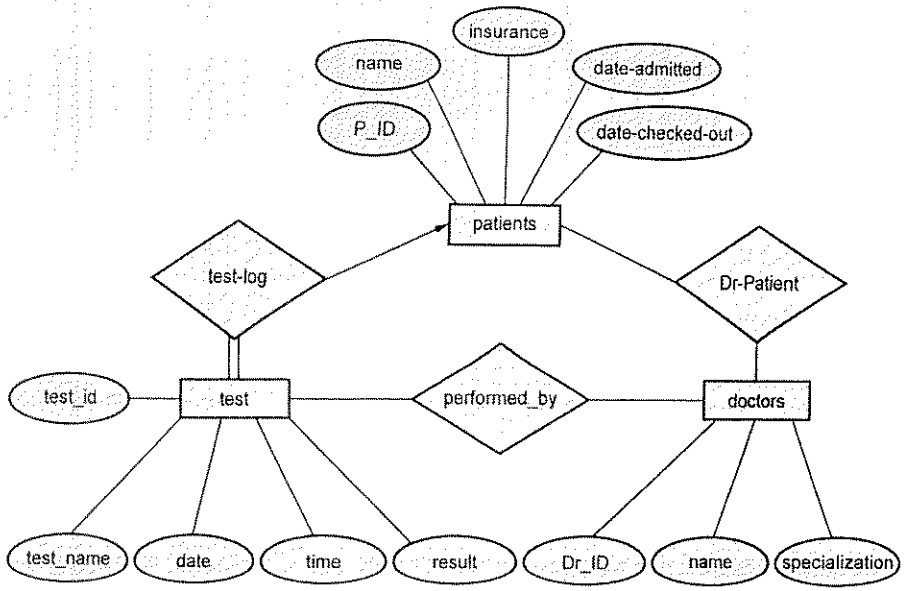
Example 1.14.7 *Construct an E-R diagram for a car insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents. Each insurance policy covers one or more cars and has one or more premium payments associated with it. Each payment is for particular period of time and has an associated due date and date when the payment was received.*

Solution :



Example 1.14.8 *Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.*

Solution :



1.15 Specialization and Generalization

- Some entities have relationships that form hierarchies. For instance, Employee can be an hourly employee or contracted employee.
- In this relationship hierarchy, some entities can act as superclass and some other entities can act as subclass.
- **Superclass :** An entity type that represents a general concept at a high level, is called superclass.
- **Subclass :** An entity type that represents a specific concept at lower levels, is called subclass.
- The subclass is said to inherit from superclass. When a subclass inherits from one or more superclasses, it inherits all their attributes. In addition to the inherited attributes, a subclass can also define its own specific attributes.
- The process of making subclasses from a general concept is called **specialization**. This is top-down process. In this process, the sub-groups are identified within an entity set which have attributes that are not shared by all entities.
- The process of making superclass from subclasses is called **generalization**. This is a **bottom up** process. In this process multiple sets are synthesized into high level entities.

- The symbol used for specialization / Generalization is 

.
- **For example** - There can be two subclass entities namely **Hourly_Emps** and **Contract_Emps** which are subclasses of **Employee** class. We might have attributes **hours_worked** and **hourly_wage** defined for **Hourly_Emps** and an attribute **contractid** defined for **ContractEmps**.

Therefore, the attributes defined for an **Hourly_Emps** entity are the attributes for **Employees** plus **Hourly_Emps**. We say that the attributes for the entity set **Employees** are inherited by the entity set **Hourly_Emps** and that **Hourly-Emps ISA** (read is a) Employees. It can be represented by following Fig. 1.15.1.

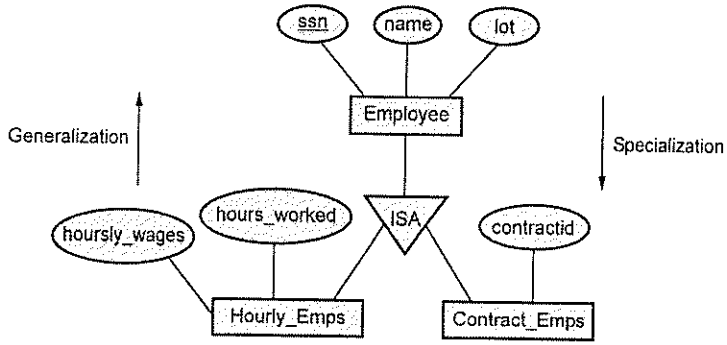


Fig. 1.15.1

1.15.1 Constraints on Specialization/Generalization

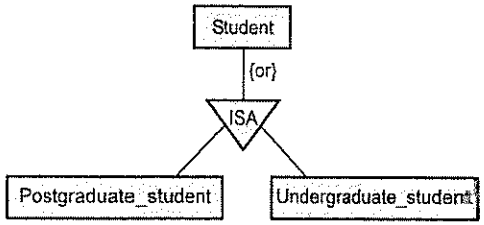
There are four types of constraints on specialization/generalization relationship. These are -

1) **Membership constraints :** This is a kind of constraints that involves determining which entities can be members of a given lower-level entity. There are two types of membership constraints -

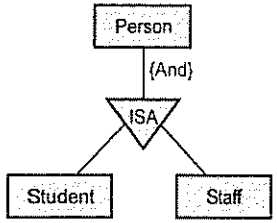
   i) **Condition defined :** In condition-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate. For example - Consider the high-level entity Set **Employee** that has attribute **Employee_type**. All Employee entities are evaluated on defining **Employee_type** attribute. All entities that satisfy the condition student type = "ContractEmployee" are included in Contracted Employee. Since all the lower-level entities are evaluated on the basis of the same attribute this type of generalization is said to be **attribute-defined**.

ii) **User defined :** This is kind of entity set that in which the membership is manually defined.

2) **Disjoint constraints :** The disjoint constraint only applies when a superclass has more than one subclass. If the subclasses are disjoint, then an entity occurrence can be a member of only one of the subclasses. For entity Student has either **Postgraduate_Student** entity or **Undergraduate_Student**.

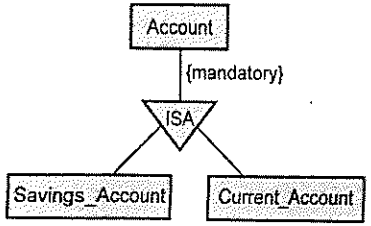


3) **Overlapping :** When some entity can be a member of more than one subclasses. For example - Person can be both a Student or a Staff. The **And can** be used to represent this constraint.

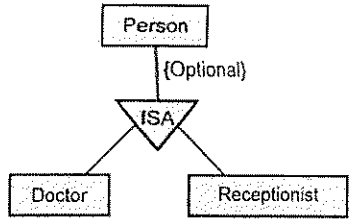


4) **Completeness :** It specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This constraint may be one of the following -

i) **Total generalization or specialization :** Each higher-level entity must belong to a lower-level entity set. For example - Account in the bank must either Savings account or Current Account. The mandatory can be used to represent this constraint.



ii) **Partial generalization or specialization :** Some higher-level entities may not belong to any lower-level entity set.



Example 1.15.1 *A car rental company maintains a database for all vehicles in its current fleet. For all vehicles, it includes the vehicle identification number license number, manufacturer, model, date of purchase and color. Special data are included for certain types of vehicles.*

Trucks : Cargo capacity

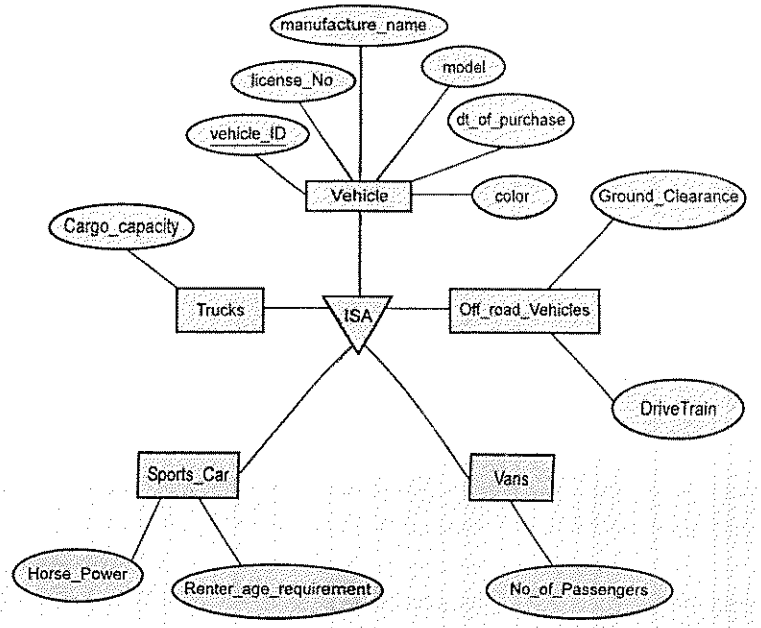
Sports cars : horsepower, renter age requirement

Vans : number of passengers

Off-road vehicles : ground clearance, drivetrain (four-or two-wheel drive)

Construct an ER model for the car rental company database.

Solution :



❑❑❑

Notes

MODULE - 2

2 Relational Model, Algebra and Logical Design

Syllabus

Relational Model : *Relational Model Concepts, Relational Model Constraints and Relational Database Schemas, Update Operations, Transactions and Dealing with Constraint Violations.* ***Relational Algebra :*** *Unary and Binary Relational Operations, Additional Relational Operations (Aggregate, Grouping etc.), Examples of Queries in Relational Algebra,* ***Mapping Conceptual Design into a Logical Design :*** *Relational Database Design using ER-to-Relational Mapping.*

Contents

Part I : Relational Model

2.1 Relational Model Concepts

VTU : Jan.-19, Marks 6

- Relation database is a collection of tables having unique names.
- For example - Consider the example of student table in which the information about the student is stored.

RollNo	Name	Phone
001	AAA	1111111111
002	BBB	2222222222
003	CCC	3333333333

Fig. 2.1.1 Student table

The above table consists of three column headers RollNo, Name and Phone. Each row of the table indicates the information of each student by means of his Roll Number, Name and Phone Number.

Similarly consider another table named Course as follows -

CourseID	CourseName	Credits
101	Mechanical	4
102	Computer Science	6
103	Electrical	5
104	Civil	3

Fig. 2.1.2 Course table

Clearly, in above table the columns are **CourseID, CourseName** and **Credits.** The CourseID **101** is associated with the course named **Mechanical** and associated with the course of mechanical there are **4 credit points.** Thus the relation is represented by the table in the relation model. Similarly we can establish the relationship among the two tables by defining the third table. For example - Consider the table Admission as

RollNo	CourseID
001	102
002	104
003	101

Fig. 2.1.3 Admission

From this third table we can easily find out that the course to which the RollNo 001 is admitted is Computer Science.

2.1.1 Basic Terms used in Relational Model

There are some commonly used terms in Relational Model and those are -

Table or relation : In relational model, table is a collection of data items arranged in rows and columns. The table cannot have duplicate data or rows. Below is an example of student table.

Roll No	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333
004	DDD	67	4444444444

Tuple or Record or Row : The single entry in the table is called tuple. The tuple represents a set of related data. In above **Student** table there are four tuples. One of the tuple can be represented as

001	AAA	88	1111111111

Attribute or Columns : It is a part of table that contains several records. Each record can be broken down into several small parts of data known as attributes. For example the above table consists of four attributes such as **RollNo, Name, Marks** and **Phone.**

Relation schema : A relation schema describes the structure of the relation, with the name of the relation (i.e. name of table), its attributes and their names and type.

Relation instance : It refers to specific instance of relation i.e. containing a specific set of rows. For example - The following is a relation instance - which contains the records with marks above 80.

RollNo	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333

Domain : For each attribute of relation, there is a set of permitted values called domain. For example - in above table, the domain of attribute **Marks** is set of all

possible permitted marks of the students. Similarly the domain of **Name** attribute is all possible names of students.

That means Domain of Marks attribute is (88,83,98)

Atomic : The domain is atomic if elements of the domain are considered to be indivisible units. For example in above **Student** table, the attribute **Phone** is non-atomic.

NULL attribute : A null is a special symbol, independent of data type, which means either unknown or inapplicable. It does not mean zero or blank. For example - Consider a salary table that contains NULL.

Emp#	Job Name	Salary	Commission
E10	Sales	12500	32090
E11	Null	25000	8000
E12	Sales	44000	0
E13	Sales	44000	Null

Degree : It is nothing but total number of columns present in the relational database. In given Student table -

Roll No	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333

The degree is 4.

Cardinality : It is total number of tuples present in the relational database. In above given table the cardinality is 3.

Example 2.1.1 *Find out following for given Staff table*

i) No. of columns ii) No. of tuples iii) Different attributes

iv) Degree v) Cardinality

StaffID	Name	Sex	Designation	Salary	DOJ
S001	John	M	Manager	50000	1 Oct. 2012
S002	Ram	M	Executive	20000	20 Jan. 2015
S003	Meena	F	Supervisor	40000	12 Aug. 2011

Solution :

i) No. of columns = 6

ii) No. of tuples = 3

iii) Different attributes are StaffID, Name, Sex, Designation, Salary, DOJ

iv) Degree = Total number of columns = 6

v) Cardinality = Total number of rows = 3

2.1.2 Characteristics of Relation

1) Ordering of Tuple in Relation

- Tuples in a relation do not have any particular order.
- However, in a file, the records are stored physically on disk; hence there exists some ordering among the records.
- For example - Tuples in Student relation can be ordered based on roll no, name, age or some other attribute.

2) Ordering of Values within a Tuple and Alternate Definition of a Relation

- The ordering of values (attributes) in relation schema is important.
- However, at a logical level, the order of attributes and their values is not that important as long as the correspondence between attributes and values is maintained.
- An alternative definition can be given such that the ordering of tuples in a relation is not necessary.
- For example - Consider following schema -

Roll No	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333
004	DDD	67	4444444444

t = <(RollNo,003),(Name,CCC),(Marks,98),(Phone,3333333333)>

t = <(Name,CCC) ,(RollNo,003), (Phone,3333333333) ,(Marks,98), >

These two tuples are same as the corresponding attribute and value pair is maintained in each tuple.

3) Values and NULLS in Tuple

- Each value within a tuple is atomic. That means it is not divisible into components within the framework of the basic relational model. Hence, composite and multivalued attributes are not allowed. This model is also called the **flat relational model.**
- Nulls are those values which are used to represent the values of attributes that may be unknown or may not apply to a tuple. The null is a special value.

4) Interpretation of a Relation

- The relation schema can be interpreted as a declaration or as a type of assertion. For example the schema of the STUDENT relation as given asserts that Student entity has Roll no, Name, Marks and Phone number. Each tuple in the relation can be interpreted as a fact or a particular instance of the assertion.
- For example - The first tuple in the Student relation is a student whose Roll no is 001, name is AAA, marks are 88 and phone number is 1111111111.

2.1.3 Relation Model Notations

- A relation schema R of degree n is denoted by R(A1, A2, .. . An).
- The uppercase letters Q, R, S denote relation names.
- The letters t, u, v denote tuples.
- The relation STUDENT(Name, Ssn,...) refers to the relation schema only.
- An attribute A can be related to the relation name R to which it belongs by using the dot notation R.A For example : STUDENT.Name or STUDENT. Phone
- All attribute names in a particular relation must be distinct.
- An n-tuple t in a relation r(R) is denoted $t=< v_1 , v_2 , .., v_n >$ where each value v_i $1 \leq i \leq n$ is the value corresponding to attribute A_i . This notation is called **component values** of tuples.
- Both $t[A_i]$ or $t. A_i$ (or t[i]) refer to the value v_i in t for attribute A_i.
- Both $t[A_u , A_v ,...., A_z]$ and $t.(A_u , A_v ,...., A_z)$, where, $A_u , A_v ,...., A_z$ is a list of attributes from R, refer to the sub-tuple of values $< v_u , v_v ,...., v_z >$ from t corresponding to the values of the attributes specified in the list.

University Questions

1. *Discuss the characteristics of relations.* **VTU : Jan.-19, Marks 6**
2. *Define the following : i) Relation state ii) Relation schema iii) Atomic iv) Domain* **VTU : Jan.-19, Marks 4**

2.2 Relational Model Constraints and Relational Database Schemas

VTU : Jan.-20, Marks 5

2.2.1 Keys

Keys are used to specify the tuples distinctly in the given relation.

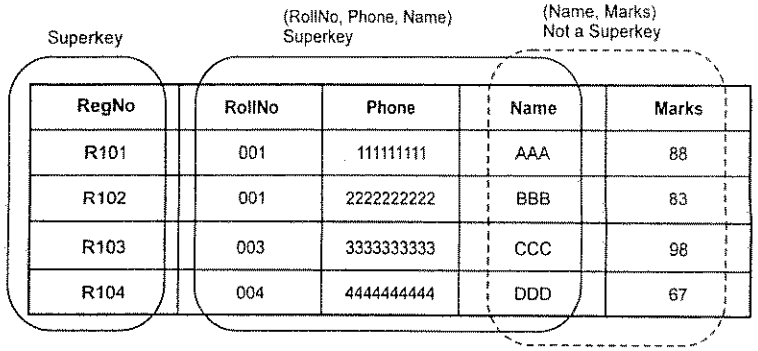
Various types of keys used in relational model are - Super key, Candidate keys, primary keys, foreign keys. Let us discuss them with suitable example.

1) **Super Key (SK) :** It is a set of one or more attributes within a table that can uniquely identify each record within a table. For example - Consider the Student table as follows -

Reg No.	Roll No	Phone	Name	Marks
R101	001	1111111111	AAA	88
R102	002	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

Fig. 2.2.1 Student

The super key can be represented as follows :



Clearly using the (RegNo) and (RollNo,Phone,Name) we can identify the records uniquely but (Name, Marks) of two students can be same, hence this combination not necessarily help in identifying the record uniquely.

2) **Candidate Key (CK) :** The candidate key is a subset of superset. In other words candidate key is a single attribute or least or minimal combination of attributes that uniquely identify each record in the table. For example - In given **Student** table, the candidate key is RegNo, (RollNo,Phone).

The candidate key can be

Candidate key

Candidate key

RegNo	RollNo	Phone	Name	Marks
R101	001	111111111	AAA	88
R102	001	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

Thus every candidate key is a super key but every super key is not a candidate key.

3) **Primary Key (PK) :** The primary key is a candidate key chosen by the database designer to identify the tuple in the relation uniquely. For example - Consider the following representation of primary key in the student table.

Primary key

RegNo	RollNo	Phone	Name	Marks
R101	001	111111111	AAA	88
R102	001	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

Other than the above mentioned primary key, various possible primary keys can be **(RollNo), (RollNo,Name), (RollNo, Phone)**

The relation among super key, candidate key and primary can be denoted by

Candidate key = Super key – Primary key

Rules for Primary Key

i) The primary key may have one or more attributes.

ii) There is only **one primary key** in the relation.

iii) The value of primary key attribute can not be NULL.

iv) The value of primary key attribute does not get changed.

4) **Alternate key :** The alternate key is a candidate key which is not chosen by the database designer to uniquely identify the tuples. For example -

Primary key

Alternate key

RegNo	RollNo	Phone	Name	Marks
R101	001	111111111	AAA	88
R102	001	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

5) **Foreign key :** Foreign key is a single attribute or collection of attributes in one table that refers to the primary key of other table.

- Thus foreign keys refer to primary key.
- The table containing the primary key is called **parent table** and the table containing foreign key is called **child table.**
- Example – Consider two tables Student and Course as follows :

Student

RegNo	RollNo	Phone	Name	Marks
R101	001	111111111	AAA	88
R102	001	2222222222	BBB	83
R103	003	3333333333	CCC	98
R104	004	4444444444	DDD	67

Parent table

Primary key

Course

CoursetD	CourseName	RegNo
C111	ComputerSci	R103
C112	Electrical	R101
C113	Mechanical	R104
C114	Civil	R102

Child table

Foreign key

From above example, we can see that two tables are linked. For instance we could easily find out that the 'Student CCC has opted for ComputerSci course'

Difference between Primary Key and Foreign Key

Primary Key	Foreign Key
Primary key is a column or a set of columns that can be used to uniquely identify a row in a table.	Foreign key is a column or a set of columns that refer to a primary key or a candidate key of another table.
A table can have a single primary key.	A table can have multiple foreign keys that can reference different tables.

2.2.2 Relational Model Constraints

Constraints mean some rules or restrictions that are set on the database.

There are three main types of constraints.

1. Domain Constraint
2. Key Constraint or NULL Constraint
3. Integrity Constraint
   - i) Entity Integrity Constraint
   - ii) Referential Integrity Constraint

1. Domain Constraint

- Domain constraint defines the domain or set of values for an attribute.
- The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.
- For example - Consider the Student table as follows.

Roll No	Name	Marks	Phone
001	AAA	88	111111111
001	BBB	83	2222222222
003	1234	98	3333333333
004	DDD	67	4444444444

Name cannot be numeric value. It must be a string. Hence this is not allowed !!!

The above relation does not satisfy the domain constraint.

2. Key Constraint or Null Constraint

- Keys are used to identify particular record from the table. Primary key is normally used to identify the record uniquely.
- Hence the key constraint can be stated as -
  - All values of primary key must be unique.
  - The value of primary key must not be NULL.
- For example - Consider the Student table as follows. For this relation, the **Roll No** is a primary key. It is expected to find the desired record using this primary key.

Duplicate values ? This is not allowed !!!

Roll No	Name	Marks	Phone
001	AAA	88	111111111
001	BBB	83	2222222222
003	CCC	98	3333333333
004	DDD	67	4444444444

The above relation does not satisfy key constraint as the primary key is not having unique value.

3. Integrity Constraint

- Integrity constraints are rules that are to be applied on database columns to ensure the validity of data.
- For example -
  - i) The Employee ID and Department ID must consist of two digits.
  - ii) Every Employee ID must start with letter.

i) Entity Integrity Constraint

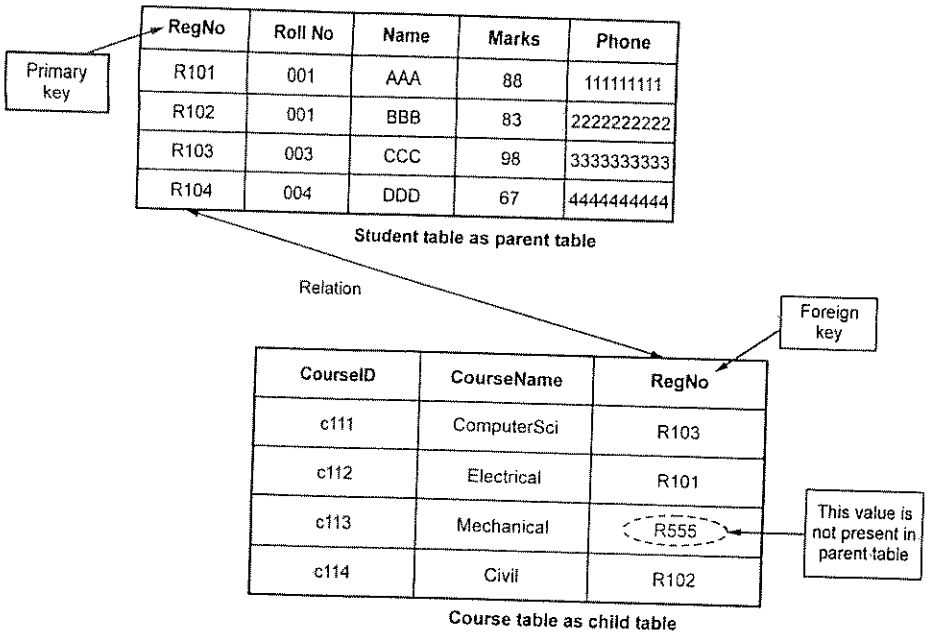
- **This rule states that** "In the relations, the value of attribute of primary key can not be null".
- The NULL represents a value for an attribute that is currently unknown or is not applicable for this tuple. The Nulls are always **to deal with incomplete or exceptional data.**
- The primary key value helps in uniquely identifying every row in the table. Thus if the users of the database want to retrieve any row from the table or perform any action on that table, they must know the value of the key for that row. Hence it is necessary that the primary key should not have the NULL value.
- For example -

Roll No	Name	Marks	Phone
001	AAA	88	111111111
001	BBB	83	2222222222
003	CCC	98	3333333333
	DDD	67	4444444444

NULL is not allowed !!!

ii) Referential Integrity Constraint

- In relationships, data is linked between two or more tables.
- This is achieved by having the foreign key (in the associated table) reference a primary key value (in the primary - or parent - table). Because of this, we need to ensure that data on both sides of the relationship remain intact.
- **The referential integrity rule states that** "whenever **a foreign key** value is used it must **reference a valid, existing primary** key in the parent table".
- For example - Consider two tables



In above relation, the registration no. R555 is not existing still if it is present in the course table, then we say that it is not following referential integrity constraint.

University Question

1. *Define following terms: i) Key ii) Super key iii) Candidate key iv) Primary key v) Foreign key*

VTU : Jan.-20, Marks 5

2.3 Update Operation, Transactions and Dealing with Constraint Violation

VTU : July-18, Marks 6

- The data manipulation operations are - Insert, Delete and Update (or Modify).
- The **insert** operation inserts new data in the database, delete operation deletes some data from the database and update operations makes some changes in the data present in the database. Whenever these operations are applied on the database, the **integrity constraints** specified on the relational database **must not be violated.**

Let us discuss these operations one by one.

1) Insert operation :

Consider following Student table -

Roll No	Name	Marks	Phone
001	AAA	88	1111111111
002	BBB	83	2222222222
003	CCC	98	3333333333
004	DDD	67	4444444444

If we perform

Insert(005, EEE,NULL,5555555555)

Then this operation will be rejected as it violates entity integrity constraint. Because the NULL value is inserted for Marks field.

If insertion violates one or more integrity constraints then the default option is to reject the insert operation.

2) Delete operation : Consider following two tables.

RegNo	Roll No	Name	Marks	Phone
R101	001	AAA	88	1111111111
R102	001	BBB	83	2222222222
R103	003	CCC	98	3333333333
R104	004	DDD	67	4444444444

Student table as parent table

CourseID	CourseName	RegNo
c111	ComputerSci	R103
c112	Electrical	R101
c113	Mechanical	R101
c114	Civil	R102

Course table as child table

If we perform

Delete on Student table where RegNo=R101

This violates the referential integrity constraint. Because there are two tuples in Course table that may get affected if we delete the RegNo=R101 (assuming that we have to delete only one student record)

The deletion operation can violate only referential integrity.

There are several options available if a deletion operation violates referential integrity.

i) One option is called **restrict** - it rejects the delete operation.

ii) Second option is **set null** or **set default.** This option either sets the value to NULL or changes the reference to valid tuples.

iii) Third option is to **cascade or propagate** the deletion by deleting tuples that reference the tuple that is being deleted.

3) Update : This operation makes changes the values of one or more attributes of tuples.

For example - Consider two tables Student table and Course table.

RegNo	Roll No	Name	Marks	Phone
R101	001	AAA	88	1111111111
R102	001	BBB	83	2222222222
R103	003	CCC	98	3333333333
R104	004	DDD	67	4444444444

Student table as parent table

CourseID	CourseName	RegNo
c111	ComputerSci	R103
c112	Electrical	R101
c113	Mechanical	R104
c114	Civil	R102

Course table as child table

If we apply

Update on RegNo of **Course** with CourseID = c112 to R555

The result of above operation is - it is rejected as it violates referential integrity. The R555 registration number is not in existence in the Parent table **Student.**

Updating an attribute that is neither part of a primary key nor of a foreign key usually causes no problems.

The DBMS need only check to confirm that the new value is of the correct data type and domain.

2.3.1 Transaction Concept

- A transaction is an executing program that includes some database operations such as reading from the database or applying insertions, deletions or updates to the database.
- At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all constraints specified in the database scheme.
- A single transaction may involve any number of retrieval operations and any number of update operations. These operations will work together as an atomic unit of work against the database.

University Question

1. *Explain briefly violations in entity integrity constraint, key and referential integrity constraints, with example.* **VTU : July-18, Marks 6**

Part II : Relational Algebra

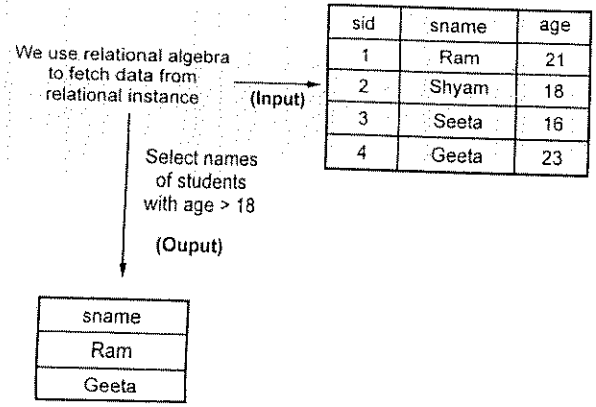
2.4 Introduction to Relational Algebra

Definition :

- Relational algebra is a procedural query language which is used to access database tables to retrieve the information in the form of relational algebra expressions.
- The queries present in the relational algebra are denoted **using operators.**

Properties of Relational Algebra

1) It makes use of operators and use relations (means tables) as operands.
2) In relational algebra, one or two relations are used as input and the single relation is generated as output without name. For example :



3) Each relational algebra is **procedural.** That means each relational query describes a step-by-step procedure for computing the desired answer, based on the order in which operators are applied in the query.
4) The relational algebra is based on set theory and it does not allow duplicate entries.
5) It does not make use of any English statements.

Importance of Relational Algebra

1. It provides **formal foundation** for relational model operations.
2. It is used as a basis for **implementing and optimizing queries** in query processing.
3. Some of the concepts of relational algebra are **incorporated into the SQL.**

There are various operations that can be performed using relational algebra. These operations are mainly classified as -

1. Unary Operation 2. Binary Operation

2.5 Unary Relational Operations

The unary operations are of two types -

1. Selection Operation 2. Projection Operation

Let us discuss them in detail -

1. Selection Operation

- This operation is used to fetch the **rows or tuples** from the table (relation).
- **Syntax :** The syntax is

$\sigma_{predicate}(relation)$

where σ represents the select operation. The predicate denotes some logic using which the data from the relation (table) is selected.

- For example - Consider the relation student as follows :

sid	sname	age	gender
1	Ram	21	Male
2	Shyam	18	Male
3	Seeta	16	Female
4	Geeta	23	Female

Fig. 2.5.1 Student table

Query : Fetch students with age more than 18

We can write it in relational algebra as

$\sigma_{age > 18}$ (Student)

The output will be -

sname
Ram
Geeta

We can also specify conditions using and, or operators.

$\sigma_{age > 18 \text{ and } gender = 'Male'}$ **(Student)**

sname
Ram

2. Projection :

- Project operation is used to project only a certain set of attributes of a relation. That means if you want to see only the names all of the students in the Student table, then you can use Project operation.
- Thus to **display particular column** from the relation, the projection operator is used.
- It will only project or show the columns or attributes asked for, and will also remove duplicate data from the columns.
- **Syntax :**

$\Pi C1, C2... (r)$

where C1, C2 etc. are attribute names (column names).

- **For example -** Consider the Student table given in Fig. 2.5.1.

Query : Display the name and age all the students

This can be written in relational algebra as

$\Pi_{sname, age}(Student)$

Above statement will show us only the Name and Age columns for all the rows of data in Student table.

sname	age
Ram	21
Shyam	18
Seeta	16
Geeta	23

3. Rename Operation

This operation is used to rename the output relation for any query operation which returns result like Select, Project etc. Or to simply rename a relation (table). The operator ρ (rho) is used for renaming.

Syntax : ρ (RelationNew, RelationOld)

For example : If you want to create a relation Student_names with **sid** and **sname** from **Student,** it can be done using rename operator as :

ρ (Student_names, ($\Pi_{sid.sname}$(Student))

2.6 Binary Relational Operation

2.6.1 The Join Operation

The join operation is used to **combine information from two or more relations.** Formally join can be defined as a cross-product followed by selections and projections, joins arise much more frequently in practice than plain cross-products. The join operator is used as ⋈.

There are two types of Join operations - Inner Join and Outer Join.

1. The Inner Join Operation

Inner Join is used to return rows from both tables which satisfy the given condition. It is the most widely used join operation and can be considered as a default join-type.

There are three types of inner joins used in relational algebra.

i) **Theta join :** This is an operation in which information from two tables is combined using **some condition** and this condition is specified along with the join operator.

$$A \bowtie_c B = \sigma_c(A \times B)$$

Thus ⋈ is defined to be a cross-product followed by a selection. Note that the condition c can refer to attributes of both A and B. The condition C can be specified using <,<=,>,<= or = operators.

For example consider two table student and reserve as follows -

Student

sid	sname	age
1	Ram	21
2	Shyam	18
3	Seeta	16
4	Geeta	23

Reserve

sid	isbn	day
1	005	07-07-18
2	005	03-03-17
3	007	08-11-16

If we want the names of students with sid(Student) = sid(Reserve) and isbn = 005, then we can write it using Cartesian product as -

$(\sigma_{((Student.sid = Reserve.sid) \wedge (Reserve.(Isbn) = 005))}(Student \times Reserve))$

Here there are two conditions as

i) (Student.sid = Reserve.sid) and ii) (Reserve.isbn = 005) which are joined by ^ operator.

Now we can use $\bowtie_C$ instead of above statement and write it as -

$(Student \bowtie_{(Student.sid = Reserve.sid) \wedge (Reserve.(Isbn) = 005)} Reserve))$

The result will be -

sid	sname	age	isbn	day
1	Ram	21	005	07-07-18
2	Shyam	18	005	03-03-18

ii) **Equijoin :** This is a kind of join in which there is **equality condition** between two **attributes (columns) of relations (tables).** For example - If there are two table Book and Reserve table and we want to find the book which is reserved by the student having isbn 005 and name of the book is 'DBMS', then :

Book

isbn	bname	Author
005	DBMS	XYZ
006	OS	PQR
007	DAA	ABC

Reserve

sid	isbn	day
1	005	07-07-18
2	005	03-03-17
3	007	08-11-16

$(\sigma_{bname} = \text{'DBMS'} (\text{Book} \bowtie_{(Book.isbn = Reserve.isbn)} \text{Reserve})$

Then we get

isbn	bname	Author	sid	day
005	DBMS	XYZ	1	07-07-18
005	DBMS	XYZ	2	03-03-18

iii) **Natural join :** When there are **common columns** and we have to **equate** these **common columns** then we use natural join. The symbol for natural join is simply ⋈ without any condition or sometimes * is used. For example, consider two tables -

Book

isbn	bname	Author
005	DBMS	XYZ
006	OS	PQR
007	DAA	ABC

Reserve

sid	isbn	day
1	005	07-07-18
2	007	03-03-17
3	006	08-11-16

Now if we want to list the books that are reserved, then that means we want to match **Books.isbn** with **Reserve.isbn.** Hence it will be simply

Books ⋈ Reserve

OR

Books * Reserve

The result of above two tables on natural join will be

isbn	bname	Author	sid	day
005	DBMS	XYZ	1	07-07-18
006	OS	PQR	3	08-11-16
007	DAA	ABC	2	03-03-17

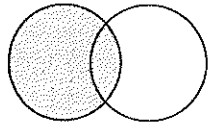
2. Outer Join Operation

An Outer Join doesn't require each record in the two join tables to have a matching record. In this type of join, the table retains each record even if no other matching record exists.

There are three types of outer joins - Left outer join, Right outer join and Full outer join.

i) Left Outer Join

- This is a type of join in which all the records from left table are returned and the matched records from the right table gets returned.
- The result is NULL from the right side, if there is no match.
- The symbol used for left outer join is ⟕.
- This can be graphically represented as follows



All rows from left table and only matching rows from right table

- **For example -** Consider two tables Student and Course as follows -

Student

RollNo	Name
1	AAA
2	BBB
3	CCC

Course

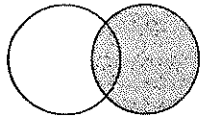
RollNo	CourseName
1	Computer
3	Electrical
5	Civil
6	Mechanical

The result of left outer join will be

RollNo	Name	CourseName
1	AAA	Computer
2	BBB	NULL
3	CCC	Electrical

ii) Right Outer Join

- This is a type of join in which all the records from right table are returned and the matched records from the left table gets returned.
- The result is NULL from the left side, if there is no match.
- The symbol used for right outer join is ⟖.
- This can be graphically represented as follows



All rows from right table and only matching rows from left table

- **For example** - Consider two tables Student and Course as follows -

Student

RollNo	Name
1	AAA
2	BBB
3	CCC

Course

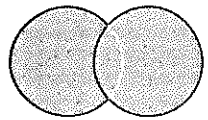
RollNo	CourseNamey
1	Computer
3	Electrical
5	Civil
6	Mechanical

The result of right outer join will be

RollNo	CourseName	Name
1	Computer	AAA
3	Electrical	CCC
5	Civil	NULL
6	Mechanical	NULL

iii) Full Outer Join

- In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.
- It is denoted by ⟗.
- Graphically it can be represented as



All rows from both tables

For example -

- Consider two tables Student and Course as follows -

Student

RollNo	Name
1	AAA
2	BBB
3	CCC

Course

RollNo	CourseNamey
1	Computer
3	Electrical
5	Civil
6	Mechanical

The result of right outer join will be

RollNo	Name	CourseName
1	AAA	Computer
2	BBB	NULL
3	CCC	Electrical
5	NULL	Civil
6	NULL	Mechanical

The symbols used are -

A LEFT OUTER JOIN B ON x=y	$A ⟕_{x=y} B$
A RIGHT OUTER JOIN B ON x=y	$A ⟖_{x=y} B$
A FULL OUTER JOIN B ON x=y	$A ⟗_{x=y} B$

2.6.2 The Division Operation

The division operator is used when we have to evaluate queries which contain the keyword **ALL.**

It is denoted by A/B where A and B are instances of relation.

For example - Find all the customers having accounts in all the branches. For that consider two tables - Customer and Account as

Customer

Name	Branch
A	Pune
B	Mumbai
A	Mumbai
C	Pune

Account

Branch
Pune
Mumbai

Now A/B will give us

Name
A

Here we check all the branches from Account table against all the names from Customer table. We can then find that only customer A has all the accounts in all the branches.

Formal Definition of Division Operation : The operation A/B is define as the set of all x values (in the form of unary tuples) such that for every y value in (a tuple of) B, there is a tuple <x,y> in A.

2.7 Set Operations

VTU : Jan.-19, Marks 8

Various set operations are - union, intersection and set-difference.

Let us understand each of these operations with the help of examples.

i) Union :

- This operation is used to fetch data from two relations (tables) or temporary relation (result of another operation).
- For this operation to work, the relations (tables) specified should have **same number of attributes (columns) and same attribute domain.** Also the **duplicate tuples are automatically eliminated** from the result.
- **Syntax : A ∪ B**

  where A and B are relations.
- **For example :** If there are two tables student and book as follows -

Student

sid	sname	age
1	Ram	21
2	Shyam	18
3	Seeta	16
4	Geeta	23

Book

isbn	bname	Author
005	DBMS	XYZ
006	OS	PQR
007	DAA	ABC

- **Query :** We want to display both the student name and book names from both the tables then

$$\Pi_{Sname}(Student) \cup \Pi_{bname}(Book)$$

ii) Intersection :

- This operation is used to fetch data from both tables which is common in both the tables.
- **Syntax : A ∩ B**

  where A and B are relations.
- Example - Consider two tables - Student and Worker

Student

Name	Branch
AAA	ComputerSci
BBB	Mechanical
CCC	Civil
DDD	Electrical

Worker

Name	Salary
XXX	3000
AAA	2000
YYY	1500
DDD	2500

- **Query :** If we want to find out the names of the students who are working in a company then

$$\Pi_{name}(Student) \cap \Pi_{name}(Worker)$$

Name
AAA
DDD

iii)Set-Difference :

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
- **Syntax : A – B**
- **For Example :** Consider two relations Full_Time_Employee and Part_Time_Employee, if we want to find out all the employee working for Fulltime, then the set difference operator is used -

$$\Pi_{EmpName}(Full\_Time\_Employee) - \Pi_{EmpName}(Part\_Time\_Employee)$$

2.7.1 The Cartesian Product (Cross Product)

- This is used to combine data from two different relations (tables) into one and fetch data from the combined relation.
- **Syntax : A × B**

- **For example :** Suppose there are two tables named Student and Reserve as follows :

Student

sid	sname	age
1	Ram	21
2	Shyam	18
3	Seeta	16
4	Geeta	23

Reserve

sid	isbn	day
1	005	07-07-18
2	005	03-03-17
3	007	08-11-16

- **Query :** Find the names of all the students who have reserved isbn = 005. To satisfy this query we need to extract data from two table. Hence the Cartesian product operator is used as

$(\sigma_{Student.sid = Reserve.sid \wedge Reserve.Isbn = 005}(Student \times Reserve)$

As an output we will get

sid	sname	age	sid	isbn	day
1	Ram	21	1	005	07-07-18
2	Shyam	18	2	005	03-03-18

Note : that although the sid columns is same, it is repeated.

University Question

1. *Discuss various types of set theory operations with example.* **VTU : Jan.-19, Marks 8**

2.8 Additional Relational Operations

2.8.1 Generalized Projection

The generalized projection operation extends the projection operation by allowing functions of attributes to be included in the projection list.

The general form of generalized projection operation is -

$\Pi_{F1,F2,\ldots,Fn}(R)$

Where F1,F2,...,Fn are functions over the attributes in relation R and may involve arithmetic operations and constant values.

For example - Consider following relation for **Student** as

Student(RollNo,Name,Marks1,Marks2,Marks3)

Student

RollNo	Name	Marks1	Marks2	Marks3
1	AAA	40	50	60
2	BBB	55	56	57
3	CCC	70	80	90

Now if we want to have MarkSheet that need to show

Total = Marks1 + Marks2 + Marks3

Then generalized projection combined with renaming may be used as follows -

MARKSHEET $\leftarrow \rho_{(RollNo,Name,Total)}(\Pi_{RollNo,Name,Marks1+Marks2+Marks3}(STUDENT))$

2.8.2 Aggregate Function and Grouping

The aggregate functions are the functions where values of multiple rows are grouped together as input on certain criteria to form a single value. For example finding average, or finding total number of employee, such operations need to use aggregate functions.

Various aggregate functions are

1. Count() 2. Sum() 3. Avg()
4. Min() 5. Max()

The aggregate function is denoted by using the operator $\mathcal{S}$. It is pronounced as script F. The syntax of using aggregate function is -

$_{<grouping\ attributes>}\mathcal{S}_{<function\ list>}(R)$

Where **<grouping attributes>** is a list of attributes of the relation R and **<function list>** is a list of (<function> and <attribute>). In this case <function> denotes any of the aggregate function and <attribute> denote the attribute of relation R.

Let us discuss these aggregate functions one by one.

Suppose the relation Student as follows -

RollNo	Name	Marks
1	AAA	40
2	BBB	55
3	CCC	70
4	DDD	55
5	EEE	NULL

1) **COUNT :** It will return total number of records

$\Im_{COUNT\ RollNo}$ (STUDENT)

Will return 5

2) **SUM :** It will return the sum of the values present in the attribute. For example

$\Im_{SUM\ marks}$ (STUDENT)

Will return

220

3) **AVERAGE :** It will return average value. For example

$\Im_{AVERAGE\ marks}$ (STUDENT)

It will return

44

4) **MAXIMUM :** It will return maximum value present in particular attribute.

For example -

$\Im_{MAXIMUM\ marks}$ (STUDENT)

It will return

70

5) **MINIMUM :** It will return minimum value present in particular attribute.

For example -

$\Im_{MINIMUM\ marks}$ (STUDENT)

It will return

40

2.9 Examples of Queries in Relational Algebra

VTU : July-18, 19, Jan.-19, 20, Marks 8

Example 2.9.1 *Consider following databases reserves(sid, bid, day) sailors (sid, sname, rating, age) boats (bid,bname,color)*

i) Find the names of sailors who have reserved boat number 103

ii) Find the names of sailors who have reserved a red boat

iii) Find the id of sailors with age over 20 who have not reserved red boat

iv) Find the names of sailors who have reserved at least one boat

Solution :

i) $(\Pi_{sname}((\sigma_{bid=103}$ Reserves) $\bowtie$ Sailors)

ii) $(\Pi_{sname}((\sigma_{color='red'}$ Boats) $\bowtie$ Reserves $\bowtie$ Sailors)

iii) $(\Pi_{sid}((\sigma_{age>20}$ Sailors) $- \Pi_{sid}((\sigma_{color='red'}$ Boats) $\bowtie$ Reserves)

(iv) $(\Pi_{sname}$(Sailors $\bowtie$ Reserves)

Example 2.9.2 *Consider the following expressions, which use the result of a relational algebra operation as the input to another operation. For each expression explain in words what the expression does :*

a) $\sigma_{year \geq 2009\ (takes)} \bowtie_{Student}$ *b)* $\sigma_{year \geq 2009\ (takes \bowtie Student)}$

c) $\Pi_{ID,\ name,\ course\text{-}id}$ *(student* $\bowtie$ *takes)*

Solution :

a. Select each student who takes at least one course in 2009, display the student information along with the information about what the courses the student took.

b. Select each student who takes at least one course in 2009, display the student information along with the information about what the courses the student took but the selection must be before join operation.

c. Display the ID, Name and Course_id of all the students who took any course in the university.

Example 2.9.3 *Consider following relational database*

branch(branch_name, branch_city, assets)

customer (customer_name, customer_street, customer_city)

loan (loan_number, branch_name, amount)

borrower (customer_name, loan_number)

account (account_number, branch_name, balance)

depositor (customer_name, account_number)

i) Find the names of all branches located in "Chennai".

ii) Find the names of all borrowers who have a loan in branch "ABC".

Solution :

i) Π_{branch_name} $(\sigma_{branch\_city\ =\ 'Chennai'})$ (branch))

ii) $\Pi_{customer\_name}$ $(\sigma_{branch\_name\ ='ABC'})$ (borrower $\bowtie$ loan))

Example 2.9.4 *author (author_id, first_name, last_name)*

author_pub (author_id, pub_id, author_position)

book (book_id, book_title, month, year, editor)

pub (pub_id, title, book_id)

i) Give the relational algebra expression that returns names of all the authors that are book editors

ii) Give the relational algebra expression that returns names of all the authors that are not book editors

iii) Write a relational algebra expression that returns the names of all authors who have at least one publication in the database.

Solution :

i) $(\Pi_{first\_name,last\_name}$ (author $_{author\_id = editor} \bowtie$ book)

ii) $(\Pi_{first\_name,last\_name}((\Pi_{author\_id}$ (author) $- \Pi_{editor}$ (book)) × author)

iii) $(\Pi_{first\_name,last\_name}$ (author × author_pub)

Example 2.9.5 *Consider the following schema :*

Supplier (sid, sname, address)

Parts (pid, pname, color)

Cataloge (sid,pid,cost)

Write the relational algebraic queries for the following :

i) Find the sids of supplier who supply some red or some green parts

ii) Find the sids of supplier who supply every red or some green parts

iii) Find the pids of parts supplied by at least two different suppliers.

Solution :

i) $\rho(R1, \Pi_{sid}((\Pi_{pid}\ \sigma_{color='red'}$ Parts) ⋈ Cataloge))

$\rho(R2, \Pi_{sid}((\Pi_{pid}\ \sigma_{color='red'}$ Parts) ⋈ Cataloge))

R1 ∪ R2

ii) $\rho(R1, \Pi_{sid,\ pid}$Cataloge)/ $(\Pi_{pid}\ \sigma_{color='red'}$ Parts))

$\rho(R2, \Pi_{sid}((\Pi_{pid}\ \sigma_{color='red'\ Parts})$ ⋈ Cataloge))

R1 ∪ R2

iii) ρ (R1, Cataloge)

ρ (R2, Cataloge)

$(\Pi_{R1.pid}\ \sigma_{R1.pid = R2.pid\ \wedge\ R1.sid!=R2.sid}$ (R1× R2)

Example 2.9.6 *Consider the relational database*

employee (person-name, street, city)

works (person-name, company-name, salary)

company (company-name, city)

manages (person-name, manager-name)

where primary keys are underlined.

a) Find the names of all employees who work for First Bank Corporation

b) Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than 200,000 per annum.

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

Solution :

a) $\Pi_{person\text{-}name}(\sigma_{company\text{-}name = \text{"First Bank Corporation"}}$(works))

b) $\Pi_{person\text{-}name,\ street,\ city}$ $(\sigma_{company\text{-}name = \text{"First Bank Corporation"}\ \wedge\ salary > 200000}$ (works ⋈ employee))

c) $\Pi_{\ person\text{-}name}$ (works ⋈ employee ⋈ company))

Example 2.9.7 *Solve the queries for the following database using relational algebra*

branch (branch-name, branch-city, assets)

customer (customer-name, customer-street, customer-only)

account (account-number, branch-name, balance)

loan (loan-number, branch-name, amount)

depositor (customer-name, account-number)

borrower (customer-name, loan-number)

1) Find all loans over $1200

2) Find the loan number for each loan of an amount greater than $1200

3) Find the names of all customers who have a loan, an account, or both, from the bank

4) Find the names of all customers who have a loan and an account at bank.

5) Find the names of all customers who have a loan at the Perryridge branch.

6) Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

7) Find the names of all customers who have a loan and an account at the Perryridge branch.

Solution :

1) $\sigma_{amount>1200}$ (loan))

2) $\Pi_{\ loan\text{-}number}(\sigma_{amount>1200}$ (loan))

3) $\Pi_{\ customer\text{-}name}$(borrower) ∪ $\Pi_{customer\text{-}name}$ (depositor)

4) $\Pi_{\ customer\text{-}name}$(borrower) ∩ $\Pi_{\ customer\text{-}name}$ (depositor)

5) $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{"Perryridge"}}(\sigma_{borrower.loan\text{-}number=loan.loan\ number}$(borrower ⋈ loan)))

6) $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{"Perryridge"}}(\sigma_{borrower.loan\text{-}number=loan.loan\text{-}number}$(borrower ⋈ loan)))

$- \Pi_{customer\text{-}name}$(depositor)

7) $\Pi_{customer\text{-}name}(\sigma_{branch\text{-}name=\text{"Perryridge"}}(\sigma_{borrower.loan\text{-}number=loan.loan\text{-}number}$(borrower ⋈ loan))

∪ customer-name(depositor))

Example 2.9.8 *Consider the relational database as given below. Give an expression in relational algebra to express each of the following queries*

Employee (person-name, street, city)

Works (person-name, company-name, salary)

Company (company-name, city)

Manages (person-name, manager-name)

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

b) Find the names, street address, and cities of residence of all employees who work for HCL Corporation and earn more than $10,000 per annum.

Solution :

(1) $\Pi_{\text{person-name}}$(Works ⋈ Employee ⋈ Company))

(2) $\Pi_{\text{person-name, street, city}}(\sigma_{\text{company-name = "HCL" ^ salary > 10000}}$ (Works ⋈ Employee))

Example 2.9.9 *Consider following Schema and represent given statements in relation algebra form*

**Branch(branch_name, branch_city)*

**Account(branch_name, acc_no, balance)*

**Depositor(customer_name, acc_no)*

i) Find out list of customer who have account at 'abc' branch.

ii) Find out all customer who have account in 'Ahemedabad' city and balance is greater than 10,000.

iii) Find out list of all branch name with their maximum balance.

Solution :

i) $\Pi_{\text{customer\_name}}(\sigma_{\text{branch\_name='abc'}}$(Account ⋈ Depositor))

ii) $\Pi_{\text{customer\_name}}(\sigma_{\text{branch\_city='ahemedabad' ^ balanace>10000}}$ (Branch ⋈Account ⋈ Depositor))

iii) branch_name $\Im_{\text{max(balance)}}$ (Account)

Example 2.9.10 *Consider the following relational database, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries :*

employee(ssn,name,dno,salary,hobby,gender)

department(dno,dname,budget,location,mgrssn)

works_on(ssn,pno)

project(pno,pname,budget,location,goal)

1) List all pairs of employee names and the project numbers they work on

2) List out department number, department name and department budget

3) List all projects that Raj Yadav works on by project name

Solution :

1) $\Pi_{\text{name,pno}}$(employee ⋈ works_on)

2) $\Pi_{\text{dno,dname,budget}}$(department)

3) RajYadav $\leftarrow \sigma_{\text{name='Raj Yadav'}}$(employee)

Result $\leftarrow \Pi_{\text{pname}}$(RajYadav ⋈ works_on ⋈ project))

Example 2.9.11 *Consider following relational database, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries :*

course(course-id, title, dept_name, credits)

instructor(id, name, dept_name, salary)

section(course-id, sec-id, semester, year, building, room-no, time_slot_id)

teaches(id, course-id, sec-id, semester, year)

1. Find the names of all instructors in the physics department.

2. Find all the courses taught in the fall 2009 semester but not in Spring semester

3. Find the names of all instructors in Com.Sci.department together with course titles of all the courses that the instructors teach.

4. Find the average salary in each department.

Solution:

1) $\Pi_{\text{name}}(\sigma_{\text{dept\_name='physics'}}$(instructor))

2) $\Pi_{\text{course-id}}(\sigma_{\text{semester='Fall' ^ year=2009}}$ (section)) - $\Pi_{\text{course-id}}(\sigma_{\text{semester='Sprint'}}$ (section))

3) $\Pi_{\text{name,title}}(\sigma_{\text{dept\_name='Comp.Sci.department'}}$(course ⋈ instructor ⋈ teaches))

4) dept_name $\Im_{\text{avg(salary)}}$(instructor)

Example 2.9.12 *Consider the following relations for a database that keeps track of student enrolled in courses and the books adopted for each course :*

STUDENT(ssn,Name,Major, bdate)

COURSE(Courseno, Cname,dept)

ENROLL(ssn,Courseno,Quarter,grade)

BOOK_ADOPTION(Courseno, Quarter, book_isbn)

TEXT(book_isbn,book_title, Publisher, Author)

Write the following queries in relational algebra on the database schema :

i) List the number of courses taken by all students named John smith in winter 2009 (i.e. Quarter = W09).

ii) Produce a list of text books(include courseno, book_isbn,book_title) for courses offered by the CS department that have used more than two books.

iii) List any department that has all its adopted books published by Pearson publishing.

VTU : July-18, Marks 6

Solution :

i) $\Pi_{\text{courseno}}(\sigma_{\text{Quarter= W09}}((\sigma_{\text{Name='John Smith'}}$ (STUDENT) ⋈ ENROLL))

ii) $\Pi_{\text{courseno,book\_isbn,book\_title}}(\sigma_{\text{dept='cs'^2<count(book\_isbn)}}$(COURSE ⋈ TEXT ⋈ BOOK_ADOPTION))

iii) OtherDept = Π_{Dept} (($\sigma_{\text{Publisher <> 'Pearson Publishing'}}$ (BOOK_ADOPTION ⋈ TEXT)) ⋈ COURSE)

AllDept = Π_{Dept} (BOOK_ADOPTION ⋈ COURSE)

result = AllDept - OtherDept

Example 2.9.13 *Consider the following Movie database :*

Movie(Title, director, Myear,Rating)

Actors(Actor, Aage)

Acts(Actor, title)

Directors(Director, dage)

Write the following queries in relational algebra on the database given :

i) Find movies made by Hanson after 1997

ii) Find all actors and directors

iii) Find "Coen's" movie with "McDormand"

iv) Find(director,actor) pairs where the director is younger than the actor.

VTU : July-19, Marks 8

Solution :

i) $\sigma_{Myear>1997 \wedge director='Hanson'}(Movies)$

ii) $\sigma_{actor(Actors)} \cup \Pi_{director}(Directors)$

iii) $e1 = \Pi_{title(\sigma_{actor='McDormand'}}(Acts))$

$e2 = \pi_{title}(\sigma_{director='Coen'}(Movies))$

$result = e1 \cap e2$

iv) $\Pi_{director,actor}(Directors \bowtie_{dAge<aAge} Actors)$

Example 2.9.14 *Consider the two tables, show the results of the following :*

T1		
A	B	C
10	a	5
15	b	8
25	a	6

T2		
P	Q	R
10	b	6
25	c	3
10	b	5

i) $T1 \rhd\!\lhd_{T1.B = T2.Q} T2$

ii) $T1 \rhd\!\lhd_{T1.A=T2.P} T2$

iii) $T1 \bowtie_{(T1.A=T2.P)AND(T1.C=T2.A)} T2$

iv) T1 – T2

VTU : Jan.-19, Marks 8

Solution :

i) $T1 \rhd\!\lhd_{T1.B = T2.Q} T2$

A	B	C	P	Q	R
15	b	8	10	b	6
-	-	-	25	c	3
15	b	8	10	b	5

ii) $T1 \rhd\!\lhd_{T1.A=T2.P} T2$

A	B	C	P	Q	R
10	a	5	10	b	6
10	a	5	10	b	5
15	b	8	-	-	-
25	a	6	25	c	3

iii) $T1 \bowtie_{(T1.A=T2.P)AND(T1.C=T2.A)} T2$

A	B	C	P	Q	R
10	a	5	10	b	5

iv) T1 - T2

A	B	C
10	a	5
15	b	8
25	a	6

Example 2.9.15 *Consider the schema*

Sailors(sid, sname, rating,age)

Boats(bid,bname, color)

Reserves (sid, bid,day)

Write relational algebraic queries for the following :

i) Find names of sailors who have reserved boat #103
ii) Find the names of sailors who have reserved a red boat.
iii) Find the names of sailors who have reserved a red boat or green boat
iv) Find the names of sailors who have reserved all boats.

VTU : Jan.-20, Marks 8

Solution :

i) $(\Pi_{sname}((\sigma_{bid=103}$ Reserves) ⋈ Sailors)

ii) $(\Pi_{sname}((\sigma_{color='red'}$ Boats) ⋈ Reserves ⋈ Sailors)

iii) $\Pi_{sname}(\sigma_{color='red' \text{ or } color = 'green'}$ Boats ⋈ Reserves ⋈ Sailors)

iv) $\Pi_{sname}($ Boats ⋈ Reserves ⋈ Sailors)

Part III : Mapping Conceptual Design into Logical Design

2.10 Relational Database Design using ER-to-Relational Mapping

VTU : July-18, Jan.-19, 20, Marks 6

In this section we will discuss how to map various ER model constructs to Relational Model construct.

Basic Steps to Convert the Basic ER Model to Relational Database Schema

Step 1 : Mapping of Regular Entity Types

Step 2 : Mapping of Weak Entity Types

Step 3 : Mapping of Binary 1:1 Relation Types

Step 4 : Mapping of Binary 1:N Relationship Types.

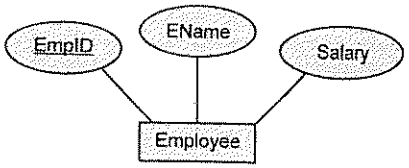
Step 5 : Mapping of Binary M:N Relationship Types.

Step 6 : Mapping of Multivalued Attributes.

Step 7 : Mapping of N-ary Relationship Types.

2.10.1 Mapping of Regular Entity to Relationship

- An entity set is mapped to a relation in a straightforward way.
- Each attribute of entity set becomes an attribute of the table.
- The primary key attribute of entity set becomes an entity of the table.
- For example - Consider following ER diagram.



The converted employee table is as follows -

EmpID	EName	Salary
201	Poonam	30000
202	Ashwini	35000
203	Sharda	40000

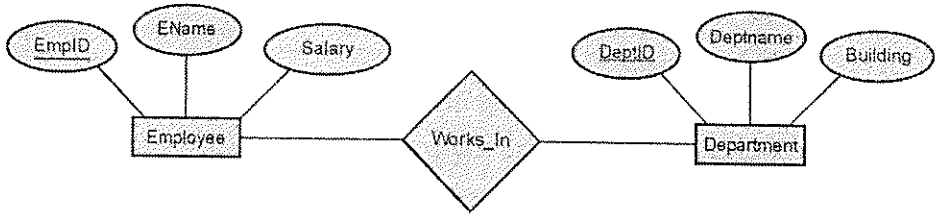
The SQL statement captures the information for above ER diagram as follows -

```
CREATE TABLE Employee( EmpID CHAR(11),
    EName CHAR(30),
    Salary INTEGER,
PRIMARY KEY(EmpID))
```

2.10.2 Mapping Relationship Sets (without Constraints) to Tables

- Create a table for the relationship set.
- Add all primary keys of the participating entity sets as fields of the table.
- Add a field for each attribute of the relationship.
- Declare a primary key using all key fields from the entity sets.
- Declare foreign key constraints for all these fields from the entity sets.

For example - Consider following ER model.



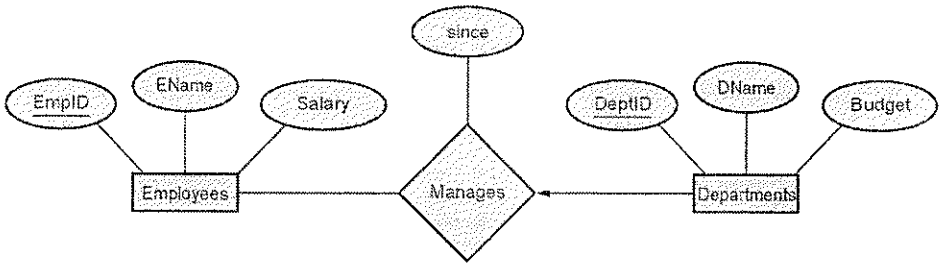
The SQL statement captures the information for relationship present in above ER diagram as follows -

```
CREATE TABLE Works_In  (EmpID CHAR(11),
                DeptID CHAR(11),
                EName CHAR(30),
                Salary INTEGER,
                DeptName CHAR(20),
                Building CHAR(10),
                PRIMARY KEY(EmpID,DeptID),
```

```
FOREIGN KEY (EmpID) REFERENCES Employee,
FOREIGN KEY (DeptID) REFERENCES Department
)
```

2.10.3 Mapping Relationship Sets (with Constraints) to Tables

- If a relationship set involves n entity sets and some m of them are linked **via arrows** in the ER diagram, the key for anyone of these **m** entity sets **constitutes a key** for the relation to which the relationship set is mapped.
- Hence we have **m** candidate keys, and one of these should be designated as the **primary key.**
- There are two approaches used to convert a relationship sets with key constraints into table.
- **Approach 1 :**
  - By this approach the relationship associated with more than one entity is separately represented using a table. For example - Consider following ER diagram. Each **Dept has at most one manager, according to the key constraint on Manages.**



Here the constraint is each department has at the most one manager to manage it. Hence no two tuples can have same DeptID. Hence there can be a separate table named **Manages** with DeptID as Primary Key. The table can be defined using following SQL statement.

```
CREATE TABLE Manages(EmpID CHAR(11),
        DeptID INTEGER,
        Since DATE,
        PRIMARY KEY(DeptID),
        FOREIGN KEY (EmpID) REFERENCES Employees,
        FOREIGN KEY (DeptID) REFERENCES Departments)
```


- **Approach 2 :**
  - In this approach, it is preferred **to translate a relationship set with key constraints.**
  - It is a superior approach because; it avoids creating a distinct table for the relationship set.
  - The idea is to include the information about the relationship set in the table corresponding to the entity set with the key, taking advantage of the key constraint.
  - This approach eliminates the need for a separate "manager" relation, and queries asking for a department's manager can be answered without combining information from two relations.
  - The only drawback to this approach is that space could be wasted if several departments have no managers.
  - The following SQL statement, defining a **Dep_Mgr** relation that captures the information in both **Departments** and **Manages,** illustrates the second approach to translating relationship sets with key constraints :

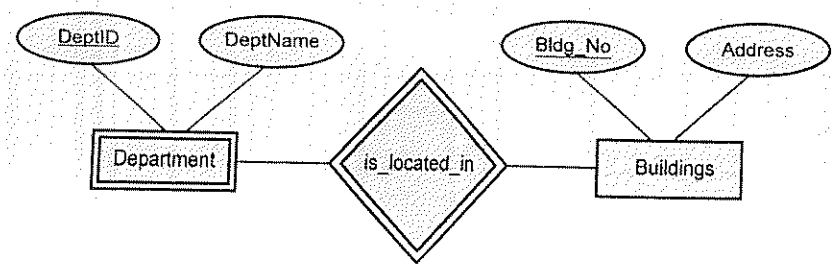
```
CREATE TABLE Dep_Mgr ( DeptID INTEGER,
        DName CHAR(20),
        Budget REAL,
        EmpID CHAR (11),
        since DATE,
        PRIMARY KEY (DeptID),
        FOREIGN KEY (EmpID) REFERENCES Employees)
```

2.10.4 Mapping Weak Entity Sets to Relational Mapping

A weak entity can be identified uniquely only by considering the primary key of another (owner) entity. Following steps are used for mapping Weka Entity Set to Relational Mapping.

- Create a table for the weak entity set.
- Make each attribute of the weak entity set a field of the table.
- Add fields for the primary key attributes of the identifying owner.
- Declare a foreign key constraint on these identifying owner fields.
- Instruct the system to automatically delete any tuples in the table for which there are no owners.

For example - Consider following ER model

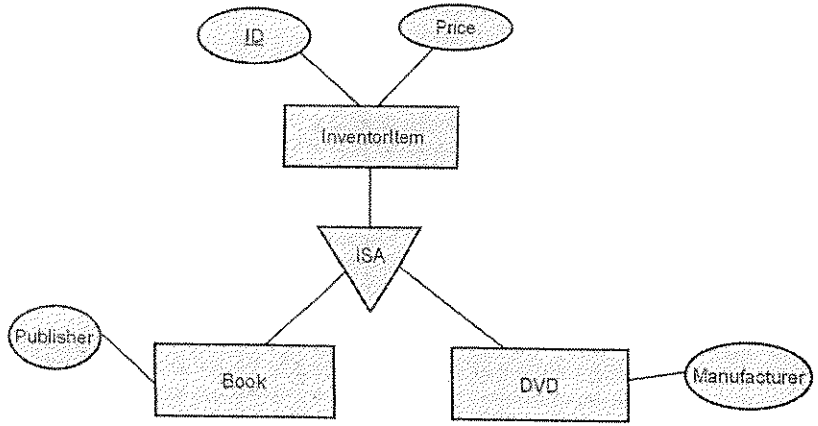


Following SQL Statement illustrates this mapping

```
CREATE TABLE Department(DeptID CHAR(11),
        DeptName CHAR(20),
        Bldg_No CHAR(5),
        PRIMARY KEY (DeptID,Bldg_No),
        FOREIGN KEY(Bldg_No) References Buildings on delete cascade
        )
```

2.10.5 Mapping of Specialization / Generalization (EER Construct) to Relational Mapping

The specialialization/generalization relationship (Enhanced ER Construct) can be mapped to database tables (relations) using three methods. To demonstrate the methods, we will take the - InventoryItem, Book, DVD.



Method 1 : All the entities in the relationship are mapped to individual tables.

```
InventoryItem(ID , name)
Book(ID,Publisher)
DVD(ID, Manufacturer)
```

Method 2 : Only subclasses are mapped to tables. The attributes in the superclass are duplicated in all subclasses. For example -

```
Book(ID,name,Publisher)
DVD(ID, name,Manufacturer)
```

Method 3 : Only the superclass is mapped to a table. The attributes in the subclasses are taken to the superclass. For example -

```
InventoryItem(ID, name,Publisher,Manufacturer)
```

This method will introduce **null** values. When we insert a **Book** record in the table, the **Manufacturer** column value will be null. In the same way, when we insert a **DVD** record in the table, the **Publisher** value will be null.

University Questions

1. *Give an example of mapping of generalization or specialization into relation schemas.* **VTU : July-18, Marks 6**
2. *Outline the steps to convert the basic ER model to Relational Database schema.* **VTU : Jan.-19, 20, Marks 6**

❑❑❑

Notes

MODULE - 2

3 SQL

Syllabus

SQL : SQL data definition and data types, specifying constraints in SQL, retrieval queries in SQL, INSERT, DELETE and UPDATE statements in SQL, additional features of SQL.

Contents

3.1 SQL Data Definition and Data Types

VTU - July -18, Marks 4

- SQL stands for **Structured Query Language.**
- It is the language of databases and almost all companies use databases to store their data.
- SQL makes use of query. A Query is a set of instruction given to the database management system. It tells any database what information we would like to get from the database.
- SQL is **case-insensitive**. However it is become standard in SQL community to use all capital letters for SQL keywords.
- SQL is a standard language for **Relational Database Management System (RDBMS).**
- There are various RDBMS software that are popularly used. It includes MySQL, Oracle, MS ACCESS, Microsoft SQL Server, Sybase and so on.
- There are three categories of SQL commands

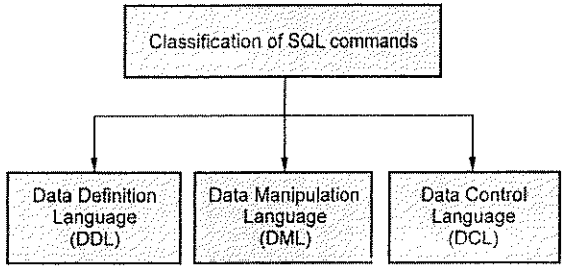


Fig. 3.1.1

- **Data definition language**

Sr.No.	Command	Purpose
1.	CREATE	This command is used to create database, tables, views or any other database objects.
2.	ALTER	It modifies the existing tables.
3.	DROP	This command deletes complete table, view or any other database object.

- **Data manipulation language**

Sr. No.	Command	Purpose
1.	SELECT	This command is used to retrieve either all or desired records from one or more tables.
2.	INSERT	For inserting the records in the table, this command is used.
3.	UPDATE	For updating one or more fields of the table, this command is used.
4.	DELETE	This command is used for deleting the desired record

- **Data control language**

Sr. No.	Command	Purpose
1.	GRANT	This command is used to give access rights or privileges to the database.
2.	REVOKE	The revoke command removes user access rights or privileges to the database objects.

- **Terminology :** In relational model we use the terms relation tuple and attribute. Same terms are used in SQL as follows

Relational Model	SQL
relation	table
tuple	row
attribute	column

- CREATE statement is a main SQL command for data definition

3.1.1 Schema and Catalog Concepts in SQL

- SQL schema :
  - It is also known as **database.**
  - It is identified by **Schema name.**
  - It includes **authorization identifier** to indicate owner of the schema.
  - It also includes **descriptors** such as GRANT for each element.
- Schema Elements include tables, constraints, views, domains and some other constructs.
- Only specific users are allowed to create schema and schema elements.
- Each statement in SQL ends with **semicolon.**

- **Syntax for creating a schema**

CREATE SCHEMA Statement

- **Example**

CREATE SCHEMA Student AUTHORIZATION 'admin_user'

3.1.2 The CREATE Statement in SQL Command

- A database can be considered as a container for tables and a table is a grid with rows and columns to hold data.
- Individual statements in SQL are called **queries**.
- We can execute SQL queries for various tasks such as creation of tables, insertion of data into the tables, deletion of record from table, and so on.

In this section we will discuss how to create a table.

Step 1 : We normally create a database using following SQL statement

Syntax

CREATE DATABASE database_name;

Example

CREATE DATABASE Person_DB

Step 2 : The table can be created inside the database as follows –

```
CREATE TABLE table_name (
 col1_name datatype,
 col2_name datatype,
 ...
 coln_name datatype
);
```

Example

```
CREATE TABLE person_details (
   AadharNo int,
   FirstName VARCHAR(20),
   MiddleName VARCHAR(20),
   LastName VARCHAR(20),
   Address VARCHAR(30),
   City VARCHAR(10)
   )
```

The blank table will be created with following structure.

Person_details

AadharNo	FirstName	MiddleName	LastName	Address	City

3.1.3 Data Types in SQL

Various data types used in SQL are -

1) Numeric data types

- Integer numbers : INT, INTEGER, SMALLINT, BIGINT
- Floating-point (real) numbers : REAL, DOUBLE , FLOAT
- Fixed-point numbers : DECIMAL(n,m), DEC(n,m), NUMERIC(n,m), NUM(n,m)

2) Character-string data types

- Fixed length : CHAR(n), CHARACTER(n)
- Varying length : VARCHAR(n), CHAR VARYING(n), CHARACTER VARYING(n), LONG VARCHAR

3) Large object data types

- Characters : CLOB, CHAR LARGE OBJECT , CHARACTER LARGE OBJECT
- Bits : BLOB, BINARY LARGE OBJECT

4) Boolean data type

- Values of TRUE or FALSE or NULL

5) DATE data type

- Ten positions
- Components are YEAR, MONTH, and DAY in the form YYYY-MM-DD

6) Additional Data type

a) TIMESTAMP data type

It includes the DATE and TIME fields.

b) INTERVAL data type

It specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp.

University Question

1. *Explain the data types available for attribute specification in SQL.* **VTU : July -18, Marks 4**

3.2 Specifying Constraints in SQL

VTU : July-18, 19, Jan.-19, 20, Marks 12

- We can specify rules for data in a table.
- When the table is created at that time we can define the constraints.
- The constraint can be column level i.e. we can impose constraint on the column and table level i.e we can impose constraint on the entire table.
- **Basic constraints :** Relational Model has 3 basic constraint types that are supported in SQL :
  1. **Key constraint :** A primary key value cannot be duplicated or primary keys must be unique.
  2. **Entity integrity constraint :** A primary key value cannot be null.
  3. **Referential integrity constraints :** The "foreign key " must have a value that is already present as a primary key, or may be null.
- Restrictions on attribute domains and NULLs :
  - The attribute value must be within the required range.
  - The primary key may not necessarily be the only attribute not allowed to be NULL.

3.2.1 Specifying Key and Referential Integrity Constraints

1) Primary key clause

The primary key constraint is defined to uniquely identify the records from the table.

The primary key must contain unique values. Hence database designer should choose primary key very carefully.

For example

Consider that we have to create a perons_details table with **AadharNo, FirstName, MiddleName, LastName, Address** and **City.**

Now making **AadharNo** as a primary key is helpful here as using this field it becomes easy to identify the records correctly.

The result will be

```
CREATE TABLE person_details (
AadharNo int,
FirstName VARCHAR(20),
MiddleName VARCHAR(20),
LastName VARCHAR(20),
Address VARCHAR(30),
City VARCHAR(10),
PRIMARY KEY(AadharNo)
);
```


We can create a composite key as a primary key using CONSTRAINT keyword. For example

```
CREATE TABLE person_details (
AadharNo int NOT NULL,
FirstName VARCHAR(20),
MiddleName VARCHAR(20),
LastName VARCHAR(20) NOT NULL,
Address VARCHAR(30),
City VARCHAR(10),
CONSTRAINT PK_person_details PRIMARY KEY(AadharNo, LastName)
);
```

2) Unique clause

Unique constraint is used to prevent same values in a column. In the EMPLOYEE table, for example, you might want to prevent two or more employees from having an identical **designation**. Then in that case we must use unique constraint.

We can set the constraint as unique at the time of creation of table, or if the table is already created and we want to add the unique constraint then we can use ALTER command.

For example -

```
CREATE TABLE EMPLOYEE(
 EmpID  INT   NOT NULL,
 Name VARCHAR (20) NOT NULL,
 Designation  VARCHAR(20) NOT NULL UNIQUE,
 Salary  DECIMAL (12, 2),
 PRIMARY KEY (EmpID)
);
```

If table is already created then also we can add the unique constraint as follows –

```
ALTER TABLE EMPLOYEE
MODIFY Designation VARCHAR(20) NOT NULL UNIQUE;
```

3) Foreign key

- Foreign key is used to link two tables.
- Foreign key for one table is actually a primary key of another table.
- The table containing foreign key is called child table and the table containing candidate primary key is called parent key.
- Consider following two tables.

Employee table

EmpID	LastName	FirstName	Age
1	Khanna	Rajesh	30
2	Joshi	Sharman	23
3	Kapoor	Tushar	20

Dept table :

DeptID	DeptName	EmpID
1	Accounts	3
2	Production	3
3	Sales	2
4	Purchase	1

- Notice that the "EmpID" column in the "Dept" table points to the "EmpID" column in the "Employee" table.
- The "EmpID" column in the "Employee" table is the PRIMARY KEY in the "Employee" table.
- The "EmpID" column in the "Dept" table is a FOREIGN KEY in the "Dept" table.
- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.
- The FOREIGN KEY constraint also prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the table it points to.
- The purpose of the foreign key constraint is to enforce referential integrity.
- The table Dept can be created as follows with foreign key constraint.

```
CREATE TABLE DEPT (
 DeptID    int
 DeptName VARCHAR(20),
 EmpID int,
 PRIMARY KEY(DeptID),
 FOREIGN KEY(EmpID) REFERENCES EMPLOYEE(EmpID)
);
```

- Default operation is reject update on violation.
- This is an implementation of **referential integrity constraint.**
- Attach **referential triggered action clause** in case referenced tuple is deleted
  - Options include SET NULL, CASCADE, and SET DEFAULT

  - **On Delete SET NULL :** – If the tuple containing the primary key is deleted, then the corresponding value in another table where that attribute is a foreign key is set to null.
  - **On Update CASCADE :-** – If the tuple has its primary key changed, that change cascades into the records where it is a foreign key.

3.2.2 Specifying Constraints on Tuple using CHECK

- The CHECK constraint is used to limit the value range that can be placed in a column.
- This clause is at the end of CREATE TABLE statement.
- Apply this constraint to each tuple individually.

For example

```
CREATE TABLE parts (
  part_no int PRIMARY KEY,
  description VARCHAR(40),
  price DECIMAL(10 , 2 ) NOT NULL CHECK(cost > 0)
);
```

University Questions

1. *Explain how constraints are specified in SQL during table creation, with suitable example.* **VTU : July-18, Marks 4**
2. *What is meant by integrity constraint? Explain the importance of referential integrity constraint. How referential integrity constraint is implemented in SQL* **VTU : July-19, Marks 8**
3. *How does SQL implement the entity integrity constraints of the relational data model ? Explain with an example.* **VTU : Jan.-19, Marks 4**
4. *Explain with examples, the basic constraints that can be specified when database table is created in SQL.* **VTU : Jan.-20, Marks 12**

3.3 Retrieval Queries in SQL

- The Select statement is used to fetch the data from the database table.
- The result returns the data in the form of table. These result tables are called resultsets.
- The **Basic Form** of SELECT statement is

```
SELECT  <attribute list>
FROM <table list>
WHERE <condition>;
```

Here

  - <attribute list> is a list of attributes whose values are to be retrieved by query.
  - <table list> is a list of relation names required to process the query
  - <condition> is a conditional(Boolean) expression that identifies the tuples to be retrieved by query.

- For example

```
SELECT AadharNo, FirstName, Address, City from person_details
```

The result of above query will be

AadharNo	FirstName	City
111	AAA	Pune

- The SELECT clause of SQL specifies the attributes whose values are to be retrieved, which are called the **projection attributes.**
- The WHERE clause specifies the Boolean condition that must be true for any retrieved tuple, which is known as the **selection condition.**
- If we want to select all the records present in the table we make use of * character.

Syntax

```
SELECT * FROM table_name;
```

Example

```
SELECT * FROM person_details;
```

The above query will result into

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune

- The WHERE command is used to specify some condition. Based on this condition the data present in the table can be displayed or can be updated or deleted.

Syntax

```
SELECT  col1,col2, ...,coln
FROM table_name
WHERE condition;
```

Example

Consider following table

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute the following query

```
SELECT AadharNo
FROM person_details
  WHERE city='Pune';
```

The result will be –

AadharNo	City
111	Pune
222	Pune

If we want records of all those person who live in city Pune then we can write the query using WHERE clause as

```
SELECT *
FROM person_details
WHERE city='Pune';
```

The result of above query will be

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune

3.3.1 Logical Operators

- Using WHERE clause we can use the operators such as AND, OR and NOT.
- AND operator displays the records if all the conditions that are separated using AND operator are true.
- OR operator displays the records if any one of the condition separated using OR operator is true.
- NOT operator displays a record if the condition is NOT TRUE.

Consider following table

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

Syntax of AND

```
SELECT col1, col2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

Example of AND

If we execute following query -

```
SELECT AadharNo, FirstName, City
FROM person_details
WHERE AadharNo=222 AND City='Pune';
```

The result will be -

AadharNo	FirstName	City
222	DDD	Pune

Syntax of OR

```
SELECT col1, col2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;
```

Example of OR

```
SELECT AadharNo, FirstName, City
FROM person_details
WHERE City='Pune' OR City='Mumbai';
```

The result will be -

AadharNo	FirstName	City
111	AAA	Pune
222	DDD	Pune
444	JJJ	Mumbai

Syntax of NOT

```
SELECT col1, col2, ...
FROM table_name
WHERE NOT condition;
```

Example of NOT

```
SELECT AadharNo, FirstName, City
FROM person_details
WHERE NOT City='Pune';
```

The result will be

AadharNo	FirstName	City
333	GGG	Delhi
444	JJJ	Mumbai

3.3.2 The Order By Clause

- Many times we need the records in the table to be in sorted order.
- If the records are arranged in increasing order of some column then it is called **ascending order**.
- If the records are arranged in decreasing order of some column then it is called **descending order**.
- For getting the sorted records in the table we use **ORDER BY** command.
- The ORDER BY keyword sorts the records in ascending order by default.

Syntax

```
SELECT col1, col2,...,coln
FROM table_name
ORDER BY col1,col2,... ASC|DESC
```

Here ASC is for ascending order display and DESC is for descending order display.

Example

Consider following table

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

```
SELECT *
FROM person_details
ORDER BY AadharNo DESC;
```

The above query will result in

AadharNo	FirstName	MiddleName	LastName	Address	City
444	JJJ	KKK	LLL	Viman nagar	Mumbai
333	GGG	HHH	III	Chandani chowk	Delhi
222	DDD	EEE	FFF	Shivaji nagar	Pune
111	AAA	BBB	CCC	M.G. road	Pune

Example 3.3.1 *Find the age of youngest sailor with age>=18 for each rating with at least two such sailors. Consider sailor instance as*

Sid	Sname	Rating	Age
1	AAA	5	21
2	BBB	2	32
3	CCC	3	19
4	DDD	3	27
5	EEE	10	14
6	FFF	2	47
7	GGG	5	29
8	HHH	4	33
9	III	10	26

Write the SQL query and possible output. Make use of group by and having clause

Solution : The query will be

```
SELECT S.rating, MIN (S.age)
      AS Younger
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

The output will be as follows –

Rating	Younger
5	21
2	32
3	19

3.3.3 The Built in Functions

In SQL a built-in function is a piece for programming that takes zero or more inputs and returns a value. An example of a built-in functions is ABS(), which when given a value calculates the absolute (non-negative) value of the number.

Query

SELECT ABS(-9) Result;

Output

Result
9

Mathematical functions

Function	Value returned
ABS (m)	Absolute value of m
MOD (m, n)	Remainder of m divided by n
POWER (m, n)	m raised to the nth power
ROUND (m [, n])	m rounded to the nth decimal place
TRUNC (m [, n])	m truncated to the nth decimal place
SIN (n)	sine (n)
COS (n)	cosine (n)
TAN (n)	tan (n)
SQRT (n)	positive square root of n
EXP (n)	e raised to the power n
LOG (n2, n1)	logarithm of n1, base n2
CEIL (n)	smallest integer greater than or equal to n
FLOOR (n)	greatest integer smaller than or equal to n
SIGN (n)	-1 if n < 0, 0 if n = 0, and 1 if n > 0

String functions

Function	Value returned
INITCAP (s)	First letter of each word is changed to uppercase and all other letters are in lower case.
LOWER (s)	All letters are changed to lowercase.
UPPER (s)	All letters are changed to uppercase.
CONCAT (s1, s2)	Concatenation of s1 and s2. Equivalent to s1 || s2
LTRIM (s [, set])	Returns s with characters removed up to the first character not in set; defaults to space
RTRIM (s [, set])	Returns s with final characters removed after the last character not in set; defaults to space
REPLACE (s, search_s [, replace_s])	Returns s with every occurrence of search_s in s replaced by replace_s; default removes search_s
SUBSTR (s, m [, n])	Returns a substring from s, beginning in position m and n characters long; default returns to end of s.
LENGTH (s)	Returns the number of characters in s.

For example

```
SELECT CONCAT('Bill', 'Gates') as "NAME"
```

Date and time functions

Function	Value returned
ADD_MONTHS (d, n)	Date d plus n months
LAST_DAY (d)	Date of the last day of the month containing d
MONTHS_BETWEEN (d, e)	Number of months by which e precedes d
NEW_TIME (d, a, b)	The date and time in time zone b when date d is for time zone a
NEXT_DAY (d, day)	Date of the first day of the week after d
SYSDATE	Current date and time
GREATEST (d1, d2, ..., dn)	Latest of the given dates
LEAST (d1, d2, ..., dn)	Earliest of the given dates

For example

```
SELECT SYSDATE();
```

Will result in

2019-06-27 10:02:39

3.3.4 String Operations

- For string comparisons, we can use the comparison operators =, <, >,<=,>=,<> with the ordering of strings determined alphabetically as usual.
- SQL also permits a variety of functions on character strings such as concatenation using operator||, extracting substrings, finding length of string, converting strings to upper case(using function **upper(s))** and lowercase(using function **lower(s))**, removing spaces at the end of string(using function(trim(s)) and so on.
- Pattern matching can also be performed on strings using two types of special characters –
  - Percent(%): It matches zero, one or multiple characters
  - Underscore(_): The _ character matches any single character.
- The percentage and underscore can be used in combinations.
- Patterns are case sensitive. That means upper case characters do not match lowercase characters or vice versa.

- **For instance :**
  - 'Data%' matches any string beginning with "Data", For instance it could be with "Database", "DataMining","DataStructure"
  - '_ _ _' matches any string of exactly three characters.
  - '_ _ _ %'matches any string of at least length 3 characters.
- The **LIKE** clause can be used in **WHERE** clause to search for specific patterns.
- **For example** – Consider following **employee database**

EmpID	EmpName	Department	Date_of_Join
1	Sunil	Marketing	1-Jan
2	Mohsin	Manager	2-Jan
3	Supriya	Manager	3-Jan
4	Sonia	Accounts	4-Jan
5	Suraj	Sales	5-Jan
6	Archana	Purchase	6-Jan

1) Find all the employee with EmpName starting with "s"

SQL Statement :

```
SELECT * FROM Employee
WHERE EmpName LIKE 's%'
```

Output

EmpID	EmpName	Department	Date_of_Join
1	Sunil	Marketing	1-Jan
3	Supriya	Manager	3-Jan
4	Sonia	Accounts	4-Jan
5	Suraj	Sales	5-Jan

2) Find the names of employee whose name begin with S and end with a

SQL Statement :

```
SELECT EmpName FROM Employee
WHERE EmpName LIKE 'S%a'
```

Output

EmpName
Supriya
Sonia

3) Find the names of employee whose name begin with S and followed by exactly four characters

```
SELECT EmpName FROM Employee
WHERE EmpName LIKE 'S_ _ _ _'
```

Output

EmpName
Sunil
Sonia
Suraj

3.4 INSERT, DELETE and UPDATE Statements in SQL

The commands used to modify the database are - INSERT, DELETE, and UPDATE

1) INSERT

We can insert one or more tuples in the relation. The syntax is as follows

Syntax

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

Example

```
INSERT INTO Person_details(AadharNo,FirstName,MiddleName,Address,City)
VALUES(555,'MMM','NNN','PPP','Model Colony','Pune');
```

2) DELETE

We can delete one or more records based on some condition. The syntax is as follows -

Syntax

```
DELETE FROM table_name WHERE condition;
```

Example

```
DELETE FROM person_details
WHERE AadharNo=333;
```


The result will be -

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
444	JJJ	KKK	LLL	Viman nagar	Mumbai

We can delete all the records from table. But in this deletion, all the records get deleted without deleting table. For that purpose the SQL statement will be

```
DELETE FROM person_details;
```

3) UPDATE

- For modifying the existing record of a table, update query is used.

Syntax

```
UPDATE table_name
SET col1=value1, col2=value2,...
WHERE condition;
```

Example

Consider following table

Person_details table

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani Chowk	Delhi
444	JJJ	KKK	LLL	Viman Nagar	Mumbai

If we execute following query

```
UPDATE person_details
SET city='Chennai'
WHERE AadharNo=333;
```

The result will be

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Chennai
444	JJJ	KKK	LLL	Viman nagar	Mumbai

3.5 Additional Features of SQL

Additional features of SQL are enlisted below -

1) It has techniques for specifying the complex retrieval queries.
2) It is possible to write the programs in various programming languages that include SQL statements
3) It supports for the set of commands for specifying physical database design parameters, file structures for relations and access paths
4) The SQL also contains the GRANT and REVOKE commands for privileges to the users.
5) Using SQL, it is possible to create triggers.
6) It support for enhanced relational systems known as object relational system.
7) SQL and relational databases can interact new technologies such as XML and OLAP.

□□□

MODULE - 3

4 Advanced SQL and Application Development

Syllabus

SQL : Advances Queries : *More complex SQL retrieval queries, Specifying constraints as assertions and action triggers, Views in SQL, Schema change statements in SQL.*
Database Application Development : *Accessing database from applications, An introduction to JDBC, JDBC classes and interfaces, SQLJ, Stored procedures, Case study : The internet bookshop.*
Internet Applications : *The three-tier application architecture, The presentation layer, The middle tier.*

Contents

Part I : Advanced SQL

4.1 More Complex SQL Retrieval Queries

VTU : Jan.-20, Marks 3

4.1.1 NULL and Three Valued Logic

- In SQL NULL is an important value. NULL is nothing but a missing value. But it has three different interpretations -
  1) **Unknown Value :** If student's age is not known then the NULL represents the unknown value.
  2) **Unavailable Value :** If a persons' phone number is not available then it can be represented by NULL value in the database
  3) **Not applicable Attribute :** If some value is not applicable to current database systems then it can be represented by NULL. For instance - Parent's designation is not applicable in student database system.
- When a record with NULL in one of its attributes is involved in a comparison operation, result is UNKNOWN.
- SQL uses a **three-valued** logic with values TRUE, FALSE, and UNKNOWN instead of the standard two-valued (Boolean) logic with values TRUE or FALSE. Using the logical operators AND, OR and NOT the result of these three values is represented in the following table -

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

- For example - The result of FALSE AND TRUE is FALSE

- SQL allows queries that check whether an attribute value is NULL. For that, instead of using =, <> (not equal to), it compares attribute value to NULL using the operators IS or IS NOT.
- For instance - Retrieve names of all students who do not opt for any sports.

```
SELECT FName, LName
FROM STUDENT
WHERE sports_type IS NULL
```

4.1.2 Nested Queries, Tuples and Set/Multiset Comparisons

- **Nested Queries :** Some queries require that existing values in the database be fetched and then used in a comparison condition. Such queries can be conveniently formulated by using nested queries, which are complete select-from-where blocks within the WHERE clause of another query. That other query is called the outer query.
- For example - Consider following query

Query : Display list of student who has not given any exam.

```
SELECT name
FROM student NOT IN
(SELECT rollno
FROM student, exam
WHERE sudent.rollno=exam.rollno)
```

4.1.3 Correlated Nested Queries

- Correlated nested queries are used to select data from a table referenced in the outer query. The subquery is known as a correlated because the subquery is related to the outer query.
- With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query, whereas a correlated subquery, executes once for each candidate row considered by the outer query. In other words, the inner query is driven by the outer query.
- For example -

Query : Retrieve the information of all employees who work in 'Marketing' department

```
SELECT E.EName,E.Designation
FROM employee E
WHERE E.emp_code=(
SELECT D.emp_code
FROM departments D WHERE D.dept_name='Marketing');
```

4.1.4 The EXISTS and UNIQUE Functions in SQL

- The EXISTS operator is used to test for the existence of any record in a subquery. The EXISTS operator returns true if the subquery returns one or more records.
- Syntax

```
SELECT column_name(s)
FROM table_name
WHERE EXISTS
(SELECT column_name FROM table_name WHERE condition);
```

- Example - Consider two tables namely Student and Student_Score table as follows -

Student Table

RollNo	FName	LName
1.	Anil	Kumble
2.	Virat	Kohli
3.	Suresh	Raina

Student_Score Table

Subject	TestNo	Grade	RollNo
Maths	1	10	1
Maths	2	9.5	1
Maths	3	9.75	1
Science	4	9.5	1
Science	5	8.5	1
Maths	6	9.25	2
Maths	7	9.8	2
Science	8	7	2

Let's say we want to get all students that have received a 10 grade in the subject 'Maths'.

The SQL query using EXISTS can be written as follows –

```
SELECT
  RollNo, FName, LName
FROM
  Student
WHERE EXISTS (
  SELECT Student_Score.RollNo
  FROM
    Student_Score
  WHERE
    Student_Score.RollNo = Student.RollNo AND
    Student_Score.grade = 10 AND
    Student_Score.Subject = 'Maths'
)
```


After running the above query the result will be -

RollNo	FName	LName
1.	Anil	Kumble

The outer query selects the student row columns we are interested in returning to the client. However, the WHERE clause is using the EXISTS operator with an associated inner subquery.

The EXISTS operator returns true if the subquery returns at least one record and false if no row is selected.

- The NOT EXISTS operator returns true if the underlying subquery returns no record. However, if a single record is matched by the inner subquery, the NOT EXISTS operator will return false, and the subquery execution can be stopped.
- The UNIQUE(Q) function returns TRUE if there are no duplicate tuples in the result of query Q.
- Syntax

```
SELECT UNIQUE column_name1, column_name2, ..., column_namen
FROM table_name;
```

- Example - Suppose there is a Student table as follows –

Student

RollNo	Name	City	Age
1	AAA	Pune	19
2	BBB	Mumbai	19
3	CCC	Chennai	20
4	DDD	Delhi	20

If we execute following query

```
SELECT Age FROM Student;
```

Then the result will be

Age
19
19
20
20

Now if we execute following query

```
SELECT UNIQUE Age FROM Student;
```

The result will be

Age
19
20

4.1.5 Renaming Attributes

The SQL **AS** is used to assign temporarily a new name to a table column or table(relation) itself. One reason to rename a relation is to replace a long relation name with a shortened version that is more convenient to use elsewhere in the query. For example - "Find the names of students and isbn of book who reserve the books".

Student

Reserve

sid	sname	age
1	Ram	21
2	Shyam	18
3	Seeta	16
4	Geeta	23

sid	isbn	day
1	005	07-07-18
2	007	03-03-18
3	009	05-05-18

```
Select S.sname,R.isbn
From Student as S, Reserve as R
Where S.sid=R.sid
```

In above case we could shorten the names of tables Student and Reserve as S and R respectively.

Another reason to rename a relation is a case where we wish to compare tuples in the same relation. We then need to take the Cartesian product of a relation with itself. For example –

If the query is - Find the names of students who reserve the book of isbn 005. Then the SQL statement will be -

```
Select S.sname,R.isbn
From Student as S, Reserve as R
Where S.sid=R.sid and S.isbn=005
```


4.1.6 Join Tables

The SQL Joins clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each.

Example : Consider two tables for using the joins in SQL. Note that **cid** is common column in following tables.

Student

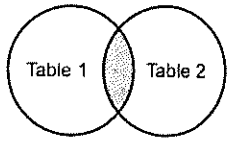
sid	cid	sname
1	101	Ram
2	101	Shyam
3	102	Seeta
4	NULL	Geeta

Reserve

cid	cname
101	Pune
102	Mumbai
103	Chennai

1) Inner Join :

- The most important and frequently used of the joins is the INNER JOIN. They are also known as an EQUIJOIN.
- The INNER JOIN creates a new result table by combining column values of two tables (Table1 and Table2) based upon the join-predicate.
- The query compares each row of table1 with each row of Table2 to find all pairs of rows which satisfy the join-predicate.
- When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row. It can be represented as :



- **Syntax :** The basic syntax of the INNER JOIN is as follows.

```
SELECT Table1.column1, Table2.column2...
FROM Table1
INNER JOIN Table2
ON Table1.common_field = Table2.common_field;
```

- **Example :** For above given two tables namely Student and City, we can apply inner join. It will return the record that are matching in both tables using the common column cid.

The query will be

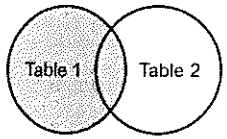
```
SELECT *
FROM Student Inner Join City on Student.cid=City.cid
```

The result will be

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai

2) Left Join(Outer Join) :

- The SQL LEFT JOIN returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still return a row in the result, but with NULL in each column from the right table.
- This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.
- It can be represented as –



- **Syntax :** The basic syntax of a LEFT JOIN is as follows.

```
SELECT
SELECT Table1.column1, Table2.column2...
FROM Table1
LEFT JOIN Table2
ON Table1.common_field = Table2.common_field;
```

- **Example :** For above given two tables namely Student and City, we can apply Left join. It will Return all records from the left table, and the matched records from the right table using the common column cid. The query will be

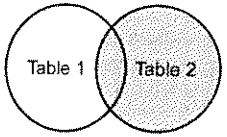
```
SELECT *
FROM Student Left Join City on Student.cid=City.cid
```

The result will be

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
4	NULL	Geeta	NULL	NULL

3) Right Join(Outer Join) :

- The SQL RIGHT JOIN returns all rows from the right table, even if there are no matches in the left table.
- This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.
- This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.
- It can be represented as follows :



- **Syntax :** The basic syntax of a RIGHT JOIN is as follow -

```
SELECT Table1.column1, Table2.column2...
FROM Table1
RIGHT JOIN Table2
ON Table1.common_field = Table2.common_field;
```

- **Example :** For above given two tables namely Student and City, we can apply Right join. It will return all records from the right table, and the matched records from the left table using the common column cid. The query will be

```
SELECT *
FROM Student Right Join City on Student.cid=City.cid
```

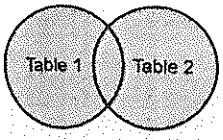
The result will be –

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
NULL	NULL	NULL	103	Chennai

4) Full Join (Outer Join) :

- The SQL FULL JOIN combines the results of both left and right outer joins.
- The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

- It can be represented as,



- **Syntax :** The basic syntax of a FULL JOIN is as follows :

```
SELECT Table1.column1, Table2.column2...
FROM Table1 FULL JOIN Table2 ON Table1.common_field = Table2.common_field;
```

The result will be -

- **Example :** For above given two tables namely **Student and City**, we can apply full join. It will return returns rows when there is a match in one of the tables using the common column cid. The query will be -

```
SELECT *
FROM Student Full Join City on Student.cid=City.cid
```

The result will be -

sid	cid	sname	cid	cname
1	101	Ram	101	Pune
2	101	Shyam	101	Pune
3	102	Seeta	102	Mumbai
4	NULL	Geeta	NULL	NULL
NULL	NULL	NULL	103	Chennai

4.1.7 Aggregate Functions

- An aggregate function allows you to perform a calculation on a set of values to return a single scalar value.
- SQL offers five built-in aggregate functions :

1. Average : avg
2. Minimum : min
3. Maximum : max
4. Total : sum
5. Count :
6. The aggregate functions that accept an expression parameter can be modified by the keywords DISTINCT or ALL. If neither is specified, the result is the same as if ALL were specified.

DISTINCT	Modifies the expression to include only distinct values that are not NULL
ALL	Includes all rows where expression is not NULL

7. Syntax of all the Aggregate Functions

```
AVG( [ DISTINCT | ALL ] expression)
COUNT(*)
COUNT( [ DISTINCT | ALL ] expression )
MAX( [ DISTINCT | ALL ] expression)
MIN( [ DISTINCT | ALL ] expression)
SUM( [ DISTINCT | ALL ] expression)
```

8. The **avg** function is used to compute average value. For example – To compute average marks of the students we can use

SQL Statement

```
SELECT AVG(marks)
FROM Students
```

9. The **Count** function is used to count the total number of values in the specified field. It works on both numeric and non-numeric data type. COUNT (*) is a special implementation of the COUNT function that returns the count of all the rows in a specified table. COUNT (*) also considers Nulls and duplicates. For example Consider following table

Test

id	value
11	100
22	200
33	300
NULL	400

SQL Statement

```
SELECT COUNT(*)
FROM Test
```

Output

4

```
SELECT COUNT(ALL id)
FROM Test
```

Output

3

10. The **min** function is used to get the minimum value from the specified column. For example – Consider the above created **Test table**

SQL Statement

```
SELECT Min(value)
```

FROM Test
Output
100

11. The max function is used to get the maximum value from the specified column. For example - Consider the above created **Test** table
SQL Statement
SELECT Max(value)
FROM Test
Output
400

12. The sum function is used to get total sum value from the specified column. For example - Consider the above created **Test** table
SQL Statement
SELECT sum(value)
FROM Test
Output
1000

4.18 The Group By and Having Clause

i) Group By :

- The GROUP BY clause is a SQL command that is used to group rows that have the same values.
- The GROUP BY clause is used in the SELECT statement.
- Optionally it is used in conjunction with aggregate functions.
- The queries that contain the GROUP BY clause are called grouped queries
- This query returns a single row for every grouped item.
- **Syntax :**

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
```

- **Example :** Consider the Student table as follows -

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai

Query : Find the total marks of each student in each city


```
SELECT SUM(marks), city
FROM Student
GROUP BY city
```

The result will be as follows –

SUM(marks)	city
150	Pune
125	Mumbai

(ii) Having :

- HAVING filters records that work on summarized GROUP BY results.
- HAVING applies to summarized group records, whereas WHERE applies to individual records.
- Only the groups that meet the HAVING criteria will be returned.
- HAVING requires that a GROUP BY clause is present.
- WHERE and HAVING can be in the same query.
- **Syntax :**

```
SELECT column-names
 FROM table-name
WHERE condition
GROUP BY column-names
HAVING condition
```

- **Example :** Consider the Student table as follows -

sid	sname	marks	city
1	AAA	60	Pune
2	BBB	70	Mumbai
3	CCC	90	Pune
4	DDD	55	Mumbai
5	EEE	84	Chennai

Query : Find the total marks of each student in the city named 'Pune' and 'Mumbai' only

```
SELECT SUM(marks), city
FROM Student
GROUP BY city
HAVING city IN('Pune','Mumbai')
```

- The result will be as follows –

SUM(marks)	city
150	Pune
125	Mumbai

4.1.9 Set Operations

1) **Union :** To use this UNION clause, each SELECT statement must have
   i) The same number of columns selected
   ii) The same number of column expressions
   iii) The same data type and
   iv) Have them in the same order

This clause is used to **combine two tables** using UNION operator. It replaces the OR operator in the query. The union operator **eliminates duplicate** while the union all query will retain the duplicates.

Syntax :

The basic syntax of a UNION clause is as follows -

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
UNION
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Here, the given condition could be any given expression based on your requirement.

Example : Find the names of the students who have reserved the 'DBMS' book or 'OS' Book

The query can then be written by considering the Student, Reserve and Book table as

```
SELECT S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='DBMS'
UNION
SELECT S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='OS'
```

2) **Intersect :** The common entries between the two tables can be represented with the help of Intersect operator. It replaces the AND operator in the query.

Syntax :

The basic syntax of a INTERSECT clause is as follows-

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
INTERSECT
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Example : Find the students who have reserved both the 'DBMS' book and 'OS' Book

The query can then be written by considering the Student, Reserve and Book table as

```
SELECT S.sid, S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='DBMS'
INTERSECT
SELECT S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='OS'
```

3) **Except :** The EXCEPT clause is used to represent the set-difference in the query. This query is used to represent the entries that are present in one table and not in other.

Syntax :

The basic syntax of a EXCEPT clause is as follows-

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
EXCEPT
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Example : Find the students who have reserved both the 'DBMS' book but not reserved 'OS' Book

The query can then be written by considering the Student, Reserve and Book table as

```
SELECT S.sid, S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='DBMS'
EXCEPT
SELECT S.sname
FROM Student S, Reserve R, Book B
WHERE S.sid=R.sid AND R.isbn=B.isbn AND B.bname='OS'
```

University Question

1. *With real world examples explain - correlated queries.* **VTU : Jan.-20, Marks 3**

4.2 Examples

VTU : July-18,19, Jan.-19,20, Marks 10

Example 4.2.1 *For the following database, identify primary key and foreign key where-ever applicable and solve the given queries in SQL.*

Item (ino, description, unit_price)

Supplier (sno, sname, address)

Supplied (sno, ino, sdate, qty, per_unit_discount)

1. Find supplier name for the suppliers who supply every item.

2. Find distinct item names for the items supplied with total discount > 500.

3. Pair of supplier names supplying same items on same dates.

Solution :

- The primary key for Item table is **ino.**
- The primary key for supplier table is **sno**
- The primary key for supplied table is **(sno,ino)**

1) Find supplier name for the suppliers who supply every item.

```
SELECT S.sname
FROM Supplier S
WHERE NOT EXISTS (
      ( SELECT * FROM Item I
        WHERE NOT EXISTS
       ( SELECT *
        FROM Supplied SP
        WHERE SP.sno = S.sno
        AND SP.ino=I.ino )
      )
```

2) Find distinct item names for the items supplied with total discount > 500.

```
SELECT DISTINCT description
      FROM Item, Supplied
      WHERE Item.ino=Supplied.ino
      AND Supplied.per_unit_discount >500
```

3) Pair of supplier names supplying same items on same dates.

```
SELECT sname
FROM Supplier
WHERE Supplied.sno=Supplier.sno
      AND
      Item.ino=Supplied.ino
```


Example 4.2.2 *Write SQL statements for the following (any five)*

Consider the following database

pilot (pid, pname)

flight (fid, ftype, capacity)

route (pid, fid, from_city, to_city)

i) List the details of flights having capacity more than 300.

ii) List the flights between 'Surat' and 'Mumbai'.

iii) List the names of the pilots who fly from 'Pune'.

iv) List the route on which, pilot named 'Mr Kapoor' flies

v) List the pilots whose names, starts with letter 'A' %' but does not end with letter 'A'.

vi) List the name of pilots who fly 'boing 737' type of flights.

Solution :

(i) List the details of flights having capacity more than 300.

```
SELECT * FROM flight
WHERE capacity>300
```

(ii) List the flights between 'Surat' and 'Mumbai'.

```
SELECT fid
FROM flight, route
WHERE
filight.fid=route.fid AND
route.from_city='Surat' AND route.to_city='Mumbai';
```

(iii) List the names of the pilots who fly from 'Pune'.

```
SELECT pname
FROM pilot,route
WHERE pilot.pid=route.pid
AND route.from_city='Pune';
```

(iv) List the route on which, pilot named 'Mr Kapoor' flies

```
SELECT from_city, to_city
FROM route, pilot
WHERE pilot.pid=route.pid
AND pilot.pname='Mr.Kapoor';
```

(v) List the pilots whose names, starts with letter 'A' %' but does not end with letter 'A'.

```
SELECT pname
FROM pilot
WHERE pname LIKE 'A%' MINUS
SELECT pname FROM pilot
WHERE pname LIKE '%A';
```

(vi) List the name of pilots who fly 'boing 737' type of flights.

```
SELECT pname
FROM pilot, flight, route
WHERE flight.ftype = 'boing 737' AND
pilot.pid = route.pid AND
route.fid = flight.fid;
```

Example 4.2.3 *Consider the relation Database.*

Person (SSN, Name, city)

Car (License_no, year, Model, SSN)

Accident(drive_no, SSN, license_no, accidentyear, damage_Amt)

Query :

1. Find out total no of cars that had accident in 1988.

2. Find the Name of driver who did not have an accident in 'Delhi'.

3. Find the car, who don't have total damage of more than ₹1000.

4. Find the cars sold in 2006 and whose owner are from 'Vadodara'

5. How many different models of car are used by 'Mr.abc'

6. Find the lucky persons who have not met any accident yet.

Solution :

1. Find out total no of cars that had accident in 1988.

```
SELECT count(License_no)
FROM Car, Accident
WHERE Accident.accidentyear=1988
```

2. Find the Name of driver who did not have an accident in 'Delhi'.

```
SELECT Name
FROM Person,Accident
WHERE Person.SSN=Accident.SSN AND Peson.city<> 'Delhi'
```

3. Find the car, who don't have total damage of more than ` 1000.

```
SELECT License_no
FROM Car, Accident
WHERE Accident.damage_amt<1000 AND
Car.License_no=Accident.licence_no;
```

4. Find the cars sold in 2006 and whose owner are from 'Vadodara'

```
SELECT license_no
FROM Car, Person
WHERE Car.SSN = Person. SSN AND
Car.year = 2006 and Person.city = 'Vadodara';
```


5. How many different models of car are used by 'Mr.abc'

```
SELECT count(model)
FROM Car, Person
WHERE Car.SSN = Person. SSN AND
Person.Name = 'Mr.abc';
```

6. Find the lucky persons who have not met any accident yet.

```
SELECT Name
FROM Person
WHERE person.SSN NOT IN (SELECT SSN FROM Accident);
```

Example 4.2.4 *We have following relations :*

Supplier (S#, sname, status, city)

Parts (P#, pname, color, weight, city)

SP(S#, P#, quantity)

Answer the following queries in SQL.

i) Find name of supplier for city = 'Delhi'.

ii) Find suppliers whose name start with 'AB'.

iii) Find all suppliers whose status is 10, 20 or 30.

iv) Find total number of city of all suppliers.

v) Find s# of supplier who supplies 'red' part.

vi) Count number of supplier who supplies 'red' part.

vii) Sort the supplier table by sname.

Solution :

i) Find name of supplier for city = 'Delhi'.

```
SELECT sname
FROM Supplier
WHERE city='Delhi'
```

ii) Find suppliers whose name start with 'AB'.

```
SELECT sname
FROM Supplier
WHERE sname='AB%'
```

iii) Find all suppliers whose status is 10, 20 or 30.

```
SELECT sname
FROM Supplier
WHERE status BETWEEN 10 AND 30
```

iv) Find total number of city of all suppliers.

```
SELECT count(*) FROM (SELECT DISTINCT CITY FROM Supplier);
```

v) Find s# of supplier who supplies 'red' part.

```
SELECT DISTINCT supplier.S#
FROM Supplier,Parts, SP
WHERE Supplier.S# = SP.S#
AND SP.P# = Parts.P# AND Parts.color = 'red';
```

vi) Count number of supplier who supplies 'red' part.

```
SELECT count(*)
FROM (SELECT DISTINCT Supplier.S#
FROM Supplier,Parts, SP
WHERE Supplier.S# = SP.S#
AND SP.P# = Parts.P# AND Parts.color = 'red');
```

vii) Sort the supplier table by sname.

```
SELECT *
FROM Supplier
ORDER BY sname;
```

Example 4.2.5 *We have following relations :*
Supplier (S#, sname, status, city)
Parts (P#, pname, color, weight, city)
SP(S#, P#, quantity)
Answer the following queries in SQL,
i) Delete records in supplier table whose status is 40.
ii) Add one field in supplier table.

Solution :

i) Delete records in supplier table whose status is 40.

```
DELETE FROM Supplier
WHERE status=40
```

ii) Add one field in supplier table.

```
ALTER TABLE Supplier
ADD(PhoneNo number);
```

Example 4.2.6 *We have following relations :*
Supplier (S#, sname, status, city)
Parts (P#, pname, color, weight, city)
SP(S#, P#, quantity)
Answer the following queries in SQL,
i) Find name of parts whose color is 'red'.
ii) Find parts name whose weight less than 10 kg.
iii) Find all parts whose weight from 10 to 20 kg.

iv) Find average weight of all parts.
v) Find S# of supplier who supply part 'p2'.
vi) Find name of supplier who supply maximum parts.
vii) Sort the parts table by pname.

Solution :

i) Find name of parts whose color is 'red'.

```
SELECT pname
FROM Parts
WHERE Parts.color = 'red';
```

ii) Find parts name whose weight less than 10 kg.

```
SELECT pname
FROM Parts
WHERE Parts.weight < 10;
```

iii) Find all parts whose weight from 10 to 20 kg.

```
SELECT pname, weight
FROM Parts
WHERE Parts.weight BETWEEN 10 AND 20;
```

iv) Find average weight of all parts.

```
SELECT AVG (weight)
FROM Parts;
```

v) Find S# of supplier who supply part 'p2'.

```
SELECT S#
FROM SP
WHERE p# = 'p2';
```

vi) Find name of supplier who supply maximum parts.

```
SELECT sname, MAX(quantity)
FROM Supplier, Parts,SP
WHERE Supplier.S# = SP.S#
AND SP.P# = P.P#
GROUP BY sname;
```

vii) Sort the parts table by pname.

```
SELECT *
FROM Parts
ORDER BY pname;
```

Example 4.2.7 *We have following relations :*
Supplier (S#, sname, status, city)
Parts (P#, pname, color, weight, city)

SP(S#, P#, quantity)
Answer the following query in SQL,
Delete records in parts table whose color is 'blue'.

Solution : (i) DELETE FROM Parts
WHERE color='blue'

Example 4.2.8 *Consider following schema and write SQL for given statements.*
student (rollno, name, branch)
exam(rollno, subject_code, obtained_marks, paper_code)
papers(paper_code, paper_satter_name, university)
i) Display name of student who got first class in subject '130703'.
ii) Display name of all student with their total mark.
iii) Display list number of student in each university.
iv) Display list of student who has not given any exam.

Solution :

i) Display name of student who got first class in subject '130703'.

```
SELECT name
FROM student,exam
WHERE exam.obtained_marks>60 AND exam.subject_code='130703' AND
student.rollno=exam.rollno
```

ii) Display name of all student with their total mark.

```
SELECT name,SUM(obtained_marks)
FROM student,exam
WHERE student.rollno=exam.rollno
```

iii) Display list number of student in each university.

```
SELECT COUNT(Rollno)
FROM student, exam, papers
WHERE student.rollno=exam.rollno AND exam.paper_code=papers.paper_code;
```

iv) Display list of student who has not given any exam.

```
SELECT name
FROM student NOT IN
(SELECT rollno
FROM student, exam
WHERE student.rollno=exam.rollno)
```

Example 4.2.9 *Write down the query for the following table where primary keys are underlined.*

Person(ss#, name, address)
Car(license, year, model)

Accident(date, driver, damage-amount)
Owns(ss#, license)
Log(license, date, driver)
1) Find the total number of people whose cars were involved in accidents in 2009.
2) Find the number of accidents in which the cars belonging to "S.Sudarshan"
3) Add a new customer to the database.
4) Add a new accident recorded for the santro belonging to "KORTH"

Solution :

1) Find the total number of people whose cars were involved in accidents in 2009.

```
SELECT count(ss#)
FROM person ,owns,accident,log
WHERE person.ss#=owns.ss#  AND
owns.licence=log.licence AND
log.driver=accident.driver AND
accident.date >= '01-jan-2009' and accident.date < ='31-dec-09;
```

2) Find the number of accidents in which the cars belonging to "S.Sudarshan"

```
SELECT COUNT(accident.date)
FROM person ,owns,accident,log
WHERE log.date=accident.date AND log.driver=accident.driver AND
own.licence=log.licence AND
person.ss#=own.ss#  AND
person.name = 's.sudarshan';
```

3) Add a new customer to the database.

```
INSERT INTO person (ss#, name, address) values (101, 'Madhav', 'Pune');
INSERT INTO owns (ss#, license) values (101, 'L101');
INSERT INTO car (license, year, model) values ('L101', 2017, 'Honda');
INSERT INTO log (license, date, driver) values ('L101', NULL, 'Shankar');
```

4) Add a new accident recorded for the santro belonging to "KORTH"

```
INSERT INTO accident (date, driver, damage-amount) VALUES ('31-Dec-2016','Shankar',10000);
INSERT INTO log (license, date, driver) VALUES ('L111','31-Dec-2016','Shankar');
INSERT INTO car (license, year, model) VALUES ('L111','2005','SANTRO');
INSERT INTO person (ss#, name, address) VALUES (111, 'Korth', 'Mumbai');
INSERT INTO owns (ss#, license) VALUES (111, 'L111');
```

Example 4.2.10 *Consider the employee data.Give an expression in SQL for the following query :*

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

1) Find the name of all employees who work for State Bank.
2) Find the names and cities of residence of all employees who work for State Bank.
3) Find all employee in the database who do not work for State Bank.
4) Find all employee in the database who earn more than every employee of UCO bank.

Solution :

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

```
SELECT employee_name
FROM Works
WHERE company_name='State Bank';
```

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

```
SELECT employee_name,city
FROM Employee, Works
WHERE company_name='State Bank' AND
Works.employee_name=Employee.employee_name;
```

3) Find all employee in the database who do not work for State Bank.

```
SELECT employee_name
FROM Works
WHERE company_name< > 'State Bank';
```

4) Find all employee in the database who earn more than every employee of UCO bank.

```
SELECT employee_name from Works
WHERE salary>(SELECT MAX(salary) FROM Works
WHERE company_name='UCO bank');
```

Example 4.2.11 *Consider following schema and write SQL for given statements.*
Student(Rollno,Name,Age,Sex,City).
Student_marks(Rollno,Sub1,Sub2,Sub3,Total,Average)
Write query to
i) Calculate and store total and average marks from sub1, sub2 and sub3.
ii)Display name of students who got more than 60 marks in subject Sub1.
iii)Display name of students with their total and average marks.
iv)Display name of students who got equal marks in subject sub2.

Solution :

i) Calculate and store total and average marks from sub1, sub2 and sub3.

```
UPDATE Student_marks
SET Total=sub1+sub2+sub3,
Average= (sub1+sub2+sub3)/3;
```


ii) Display name of students who got more than 60 marks in subject Sub1.

```
SELECT Name
FROM Student, Student_marks
WHERE Student.Rollno=Student_marks.Rollno AND Student_marks.sub1>60
```

iii) Display name of students with their total and average marks.

```
SELECT Name, Total, Average
FROM Student, Student_marks
WHERE Student.Rollno=Student_marks.Rollno
```

iv) Display name of students who got equal marks in subject sub2.

```
SELECT Name
FROM Student S, Student_marks SM1, Student_marks SM2
WHERE S.Rollno=SM1.Rollno AND SM1.sub2=SM2.sub2;
```

Example 4.2.12 *We have following relations.*
EMP (empno, ename, jobtitle, manager no, hiredate, sal, comm, dept no)
DEPT (dept no, dname, loc)
i) The employees who are getting salary greater than 3000 for those persons belongings to the department 20
ii) Employees who are not getting any commission.
iii) Find how many job titles are available in employee table.
iv) Display total salary spent for each job category.
v) Display number of employees working in each department and their department name.
vi) List ename whose manager is NULL.
vii) List all employee names and their salaries, whose salary lies between 1500/- and 3500/- both inclusive.

Solution :

i) The employees who are getting salary greater than 3000 for those persons belongings to the department 20

```
SELECT ename
FROM EMP
WHERE EMP.sal>3000 AND EMP.dept_no=20
```

ii) Employees who are not getting any commission.

```
SELECT ename
FROM EMP
WHERE EMP.comm IS NULL;
```

iii) Find how many job titles are available in employee table.

```
SELECT count(jobtitle)
FROM EMP
```

iv) Display total salary spent for each job category.

```
SELECT ename,SUM(sal)
FROM EMP
GROUP BY jobtitle;
```

v) Display number of employees working in each department and their department name.

```
SELECT COUNT(EMP.eno), DEPT.dname
FROM EMP, DEPT
WHERE EMP.dept_no=DEPT.dept_no
GROUP BY DEPT.dept_no;
```

vi) List ename whose manager is NULL.

```
SELECT ename
FROM EMP
WHERE manager_no IS NULL;
```

vii) List all employee names and their salaries, whose salary lies between 1500/- and 3500/- both inclusive.

```
SELECT ename, sal
FROM EMP
WHERE sal>=1500 AND sal<=3500;
```

Example 4.2.13 *We have following relations.*

EMP (empno, ename, jobtitle, manager no, hiredate, sal, comm, dept no)

DEPT (dept no, dname, loc)

Answer the following queries in SQL

i) Find the employees working in the department 10, 20, 30 only.

ii) Find employees whose names start with letter A or letter a.

iii) Find employees along with their department name.

iv) Find employees whose manager is KING.

v) Find the employees who are working in smith's department.

vi) Find the employees who get salary more than Allen's salary.

vii) Display employees who are getting maximum salary in each department.

Solution :

i) Find the employees working in the department 10, 20, 30 only.

```
SELECT empno
FROM EMP
WHERE dept_no BETWEEN 10 AND 30
```

ii) Find employees whose names start with letter A or letter a.

```
SELECT ename
FROM EMP
WHERE ename='A%' OR ename='a%';
```


iii) Find employees along with their department name.

```
SELECT EMP.ename,DEPT.dname
FROM EMP, DEPT
WHERE EMP.dept_no=DEPT.dept_no ;
```

iv) Find employees whose manager is KING.

```
SELECT ename
FROM EMP
WHERE managerno = (SELECT  empno FROM EMP WHERE ename='KING');
```

v) Find the employees who are working in smith's department.

```
SELECT ename
FROM EMP,DEPT
WHERE EMP.dept_no=DEPT.dept_no AND ename='Smith';
```

vi) Find the employees who get salary more than Allen's salary.

```
SELECT ename
FROM EMP
WHERE sal> (SELECT sal FROM EMP WHERE ename='Allen');
```

vii) Display employees who are getting maximum salary in each department.

```
SELECT ename, MAX(sal)
FROM EMP
GROUP BY dept_no;
```

Example 4.2.14 *Write queries for the following table.*

T1 (Empno, Ename, Salary, Designation),

T2 (Empno, Deptno.)

i) Display all rows for salary greater than 5000

ii) Display the deptno for the ename='shyam'.

iii) Add a new column deptname in table T2.

iv) Change the designation of ename='ram' from 'clerk' to 'senior clerk'.

v) Find the total salary of all the rows.

vi) Display Empno, Ename, Deptno and Deptname.

vii) Drop the table T1.

Solution :

i) Display all rows for salary greater than 5000

```
SELECT *
FROM T1
WHERE Salary>5000
```

ii) Display the deptno for the ename='shyam'.

```
SELECT deptno
FROM T1,T2
WHERE T1.Empno=T2.Empno AND T1.Ename='shyam';
```

iii) Add a new column deptname in table T2.

```
ALTER TABLE T2
ADD (deptname VARCHAR(20));
```

iv) Change the designation of ename='ram' from 'clerk' to 'senior clerk'.

```
UPDATE T1
SET T1.designation='Senior Clerk'
WHERE T1.ename='ram';
```

v) Find the total salary of all the rows.

```
SELECT SUM(Salary)
FROM T1;
```

vi) Display Empno, Ename, Deptno and Deptname.

```
SELECT E.Empno, E.Ename, D.Deptno, D.Deptname
FROM T1 E, T2 D
WHERE E.Empno=D.Empno;
```

vii) Drop the table T1.

```
DROP Table T1;
```

Example 4.2.15 *Solve following queries with following table, where underlined attribute is primary key.*

primery key.

Person(SS#, name, address)

Car(license, year, model)

Accident(date, driver,damage-amount)

Owns(SS##,license)

Log(licence, date, driver)

i) Find the name of person whose license number is '12345'.

ii) Display name of driver with number of accidents done by that driver.

iii) Add a new accident by 'Ravi' for 'BMW' car on 01/01/2013 for damage amount of 1.5 lakh rupees.

Solution :

i) Find the name of person whose license number is '12345'.

```
SELECT name
FROM Person, Owns,Car
WHERE Person.SS#=Owns.SS#  AND
```



```
Car.license=Owns.license AND
Car.license='12345';
```

ii) Display name of driver with number of accidents done by that driver.

```
SELECT driver,COUNT(*)
FROM Accident
GROUP BY driver
```

iii) Add a new accident by 'Ravi' for 'BMW' car on 01/01/2013 for damage amount of 1.5 lakh rupees.

```
INSERT INTO Person (SS#, name, address) VALUES(111,'Ravi','Mumbai');
INSERT INTO Car (license,year,model) VALUES('L111','2008,'BMW');
INSERT INTO Owns(SS#, license) VALUES(111, 'L111');
INSERT INTO Accident('01/01/2013','Ravi',150000);
INSERT INTO Log('L111', '01/01/2013','Ravi');
```

Example 4.2.16 *For Supplier - Parts database*

Supplier(S#, sname, status, city)

Parts(P#, pname,color,weight,city)

SP(S#, P#, quantity)

Answer the following queries in SQL.

i) Display the name of supplier who lives in 'Ahemdabad'.

ii) Display the parts name which is not supplied yet.

iii) Find all suppliers whose status is either 20 or 30.

Solution :

i) Display the name of supplier who lives in 'Ahemdabad'.

```
SELECT sname
FROM Supplier
WHERE city='Ahemdabad';
```

ii) Display the parts name which is not supplied yet.

```
SELECT pname
FROM Parts, SP
WHRE SP.P# <> Parts.P#
```

iii) Find all suppliers whose status is either 20 or 30.

```
SELECT sname
FROM Supplier
WHERE status = 20 or status = 30;
```

Example 4.2.17 *For Supplier - Parts database*

Supplier(S#, sname, status, city)

Parts(P#, pname,color,weight,city)

SP(S#, P#, quantity)
Answer the following queries in SQL.
i) Find the name of parts having 'Red' colour.
ii) Delete parts whose weight is more than 100 gram.
iii) Count how many times each supplier has supplied part 'P2'.
iv) How much times shipment is for more than 100 quantities ?

Solution :

i) Find the name of parts having 'Red' colour.

```
SELECT pname
FROM Parts
WHERE color='Red';
```

ii) Delete parts whose weight is more than 100 gram.

```
DELETE FROM Parts
WHERE weight>100
```

iii) Count how many times each supplier has supplied part 'P2'.

```
SELECT S#, COUNT(*)
FROM SP
WHERE P#='P2'
```

iv) How much times shipment is for more than 100 quantities?

```
SELECT S#, COUNT(*)
FROM SP
WHERE quantity>100;
```

Example 4.2.18 *Consider following schema and write SQL for given statements.*
student(RollNo, Name, Age, Sex, City)
Student_marks(RollNo, Sub1, Sub2, Sub3, Total, Average)
Write query to
i) Display name and city of students whose total marks are greater than 225.
ii) Display name of students who got more than 60 marks in each subject.
iii) Display name of city from where more than 10 students come from.
iv) Display a unique pair of male and female students.

Solution :

i) Display name and city of students whose total marks are greater than 225.

```
SELECT name, city
FROM student,student_marks
WHERE student.RollNo = student_marks.RollNo AND student_marks.Total>225
```


ii) Display name of students who got more than 60 marks in each subject.

```
SELECT name
FROM student,student_marks
WHERE student.RollNo = student_marks.RollNo AND
student_marks.Sub1>60 OR student_marks.Sub2>60 OR student_marks.Sub3>60;
```

iii) Display name of city from where more than 10 students come from.

```
SELECT city
FROM student
WHERE count(RollNo)>10
```

iv) Display a unique pair of male and female students.

```
SELECT S1.name
FROM Student S1, Student S2
WHERE S1.Name=S2.Name AND S1.sex='M' S2.sex='F'
```

Example 4.2.19 *C Write queries for the following tables :*
T1 (Empno, Ename, Salary, Designation)
T2 (Empno, Deptno.)
1) Display all the details of the employee whose salary is lesser than 10 K.
2) Display the Deptno in which employee Seeta is working.
3) Add a new column Deptname in table T2.
4) Change the designation of Geeta from 'Manager' to 'Senior Manager'.
5) Find the total salary of all the employees.
6) Display Empno, Ename, Deptno and Deptname
7) Drop the table T1.

Solution :

1) Display all the details of the employee whose salary is lesser than 10 K.

```
SELECT *
FROM T1
WHERE Salary<10000
```

2) Display the Deptno in which employee Seeta is working.

```
SELECT Deptno
FROM T2, T1
WHERE T1.Empno=T2.Empno AND T1.Ename='Seeta';
```

3) Add a new column Deptname in table T2.

```
ALTER TABLE T2
ADD (Deptname VARCHAR(20));
```

4) Change the designation of Geeta from 'Manager' to 'Senior Manager'.

```
UPDATE T1
SET Designation ='Senior Manager'
WHERE Ename='Geeta';
```

5) Find the total salary of all the employees.

```
SELECT SUM(Salary)
FROM T1
```

6) Display Empno, Ename, Deptno and Deptname

```
SELECT T1.Empno,T1.Ename, T2.Deptno, T2.Deptname
FROM T1, T2
```

7) Drop the table T1.

```
DROP TABLE T1
```

Example 4.2.20 *We have following relations :*

EMP(empno, ename, jobtitle, managerno, hiredate, sal, comm, deptno)

DEPT(deptno, dname, loc)

Answer the following queries in SQL.

i) Find the Employees working in the department 10, 20, 30 only.

ii) Find Employees whose names start with letter A or letter a.

iii) Find Employees along with their department name.

iv) Insert data in EMP table.

v) Find the Employees who are working in Smith's department

vi) Update Department name of Department No = 10

vii) Display employees who are getting maximum salary in each department

Solution :

i) Find the Employees working in the department 10, 20, 30 only.

```
SELECT ename
FROM EMP
WHERE deptno BETWEEN 10 AND 30
```

ii) Find Employees whose names start with letter A or letter a.

```
SELECT ename
FROM EMP
WHERE ename='A%' OR ename='a%';
```

iii) Find Employees along with their department name.

```
SELECT EMP.ename,DEPT.dname
FROM EMP, DEPT
WHERE EMP.deptno=DEPT.deptno ;
```


iv) Insert data in EMP table.

```
INSERT INTO EMP( empno, ename, jobtitle, managerno, hiredate, sal, comm, deptno)
VALUES ('E111','AAA','Manager',M123,01-01-2010,20000,2000,'D111');
```

v) Find the Employees who are working in Smith's department

```
SELECT ename
FROM EMP,DEPT
WHERE EMP.deptno=DEPT.deptno AND ename='Smith';
```

vi) Update Department name of Department No = 10

```
UPDATE DEPT
SET dname='Accounts'
WHERE deptno=10;
```

vii) Display employees who are getting maximum salary in each department

```
SELECT ename, MAX(sal)
FROM EMP
GROUP BY deptno;
```

Example 4.2.21 *Consider following Hotel database, primary keys are underlined :*

hotel(hotel_no,name,type,price)

room(room-no,hotel-no,type,price)

booking(hotel-no,guest-no,date-from,date-to,room-no)

guest(guest-no,name,address)

Give an expression in SQL for each of the following queries

(1) List the names and addresses of all guests in London, alphabetically ordered by name.

(2) List out hotel name and total number of rooms available

(3) List the details of all the rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.

(4) List all guests currently staying a the Grosvenor Hotel.

(5) List the rooms that are currently unoccupied at the Grosvenor Hotel.

(6) List the number of rooms in each hotel in London.

(7) List out all guests who have booked room for three or more days.

Solution :

(1) List the names and addresses of all guests in London, alphabetically ordered by name.

```
SELECT name, address
FROM guest
WHERE address LIKE '%London%'
ORDER BY name;
```

(2) List out hotel name and total number of rooms available

```
SELECT name, COUNT(room-no)
FROM hotel, room
WHERE hotel.hotel_no=room.hotel_no
GROUP by hotel_no;
```

(3) List the details of all the rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.

```
SELECT r.* FROM Room r LEFT JOIN
 (SELECT g.guestName, h.hotelNo, b.roomNo
       FROM guest g, booking b, hotel h
 WHERE g.guest_no = b.guest_no AND
b.hotel_no = h.hotel_no AND
 h.name= 'Grosvenor Hotel' AND
 date_from <= CURRENT_DATE AND
 date_to >= CURRENT_DATE)
AS XXX  ON r.hotel_no = XXX.hotel_no AND r.room_no = XXX.room_no;
```

(4) List all guests currently staying at the Grosvenor Hotel.

```
SELECT * FROM guest
 WHERE guest_no =
 (SELECT guest_no FROM booking
 WHERE
 date-from <= CURRENT_DATE AND date-to >= CURRENT_DATE AND
 hotel_no =
 (SELECT hotel_no FROM hotel
 WHERE name = 'Grosvenor Hotel'));
```

(5) List the rooms that are currently unoccupied at the Grosvenor Hotel.

```
SELECT (r.hotel_no, r.room_no, r.type, r.price)
FROM room r, hotel h
WHERE r.hotel_no = h.hotel_no AND
h.name = 'Grosvenor Hotel' AND
NOT EXIST
 (SELECT *
  FROM booking b, hotel h
  WHERE (date_from <= 'CURRENT_DATE'
   AND date_to >= 'CURRENT_DATE')
   AND r.hotel_no=b.hotel_no
   AND r.room_no=b.room_no
   AND r.hotel_no=h.hotel_no
    AND name = 'Grosvenor Hotel');
```

(6) List the number of rooms in each hotel in London.

```
SELECT hotel_no, COUNT(room_no) AS count
FROM room r, hotel h
 WHERE r.hotel_no = h.hotel_no AND city = 'London'
 GROUP BY hotel_no;
```


(7) List out all guests who have booked room for three or more days.

```
SELECT guest-no, name
FROM guest g ,booking b
WHERE b.date-to Minus b.date-from >=3
```

Example 4.2.22 *Consider following schema and write SQL for given statements*

employee(employee-name,street,city)

works(employee-name, company-name, salary)

company(company-name, city)

manages(employee-name,manager-name)

(1) Find the names of all employees who work for first bank corporation.

(2) Give all employees of first bank corporation a 10-percent raise.

(3) Find the names and cities of residence of all employees who work for first bank corporation.

(4) Find the names and street addresses, cities of residence of all employees who work for First Bank Corporation and earn more than $10,000

(5) Find all employees in the database who live in the same cities as the companies for which they work.

(6) Find all employees in the database who do not work for First Bank Corporation.

(7) Find the company and number of employees in company that has more than 30 employees

Solution :

(1) Find the names of all employees who work for First Bank Corporation.

```
SELECT employee_name
FROM works
WHERE company_name='First Bank Corporation';
```

(2) Give all employees of First Bank Corporation a 10-percent raise.

```
UPDATE works
SET salary=salary*1.1
WHERE company_name='First Bank Corporation';
```

(3) Find the names and cities of residence of all employees who work for First Bank Corporation.

```
SELECT employee_name,city
FROM employee
WHERE employee_name IN
(SELECT employee_name
FROM works
WHERE company-name='First Bank Corporation');
```

(4) Find the names and Street addresses, cities of residence of all employees who work for First Bank Corporation and earn more than $10,000

```
SELECT employee_name,city
FROM employee
WHERE employee_name IN
(SELECT employee_name
FROM works
WHERE company-name='First Bank Corporation' AND salary>10000);
OR
SELECT E.employee_name, E.street, E.city
FROM employee as E, works as W
WHERE
E.employee_name=W.employee_name AND
W.company_name='First Bank Corporation' AND W.salary>10000
```

(5) Find all employees in the database who live in the same cities as the companies for which they work.

```
SELECT E.employee_name
FROM employee as E, works as W, company as C
where E.employee_name=W.employee_name AND
E.city=C.city AND
W.company_name=C.company_name;
```

(6) Find all employees in the database who do not work for First Bank Corporation.

```
SELECT employee_name
FROM works
WHERE company_name <>'First Bank Corporation';
```

(7) Find the company and number of employees in company that has more than 30 employees

```
SELECT company-name,COUNT(employee-name)
FROM employee E, company C,works W
WHERE E.employee-name=W.employee-name AND
W.company-name=C.company-name
HAVING COUNT(employee-name)>30;
```

Example 4.2.23 *Consider following relations and write SQL queries for given statements*

Assume suitable constraints

Instructor(ID, Name, Dept_name, Salary)

Teaches(ID,Course_id, Sec_id, Semester(even/odd), Year)

(1) Write SQL query to create Instructor table

(2) Find the average salary of the instructor in computer department.

(3) Find the number of instructors in each department who teach a course in even semester of 2016

(4) Find the names of instructor with salary amounts between 30000 and 50000

Solution :

(1) Write SQL query to create Instructor table

```
CREATE TABLE Instructor
(ID CHAR,
Name VARCHAR(20),
Dept_name VARCHAR(15),
Salary numeric(8,2)
);
```

(2) Find the average salary of the instructor in computer department.

```
SELECT AVG(Salary)
FROM Instructor
WHERE Dept_name='Computer';
```

(3) Find the number of instructors in each department who teach a course in even semester of 2016

```
SELECT COUNT(DISTINCT ID)
FROM Teaches
WHERE Semester='even' AND Year=2016
```

(4) Find the names of instructor with salary amounts between 30000 and 50000

```
SELECT Name
FROM Instructor
WHERE Salary >=30000 AND Salary <=50000
```

Example 4.2.24 *Consider following schema and write SQL for given statements*

Client_master(clientno,name,address,city,pincode,state,baldue)

Product_master(productno,name,profitpercent,unitmeasure,sellprice,costprice)

Sales_master(Salesmanno,name,address,city,pincode,state,salary,tgtotget,remarks)

(1) Find out the names of all clients

(2) List all the clients who are located in Mumbai

(3) Delete all salesmen from salesman_master whose salaries are equal to ₹3500

(4) Destroy the table client_master along with data

(5) List the names of all clients having 'a' as the second letter in their names.

(6) Count the number of products having cost price is less than or equal to 500.

(7) Calculate the average, minimum and maximum Sell price of product

Solution :

```
(1) SELECT name FROM Client_master;
(2) SELECT * FROM Client_master
    WHERE city='Mumbai'
(3) DELETE FROM Salesman_master
    WHERE salary=3500;
```

(4) DROP TABLE Client_master;
(5) SELECT name
FROM Client_master
WHERE name like '_a%';
(6) SELECT count(productno)
FROM product_master
WHERE costprice <= 500
(7) SELECT AVG(sellprice), MIN(sellprice), MAX(sellprice)
FROM product_master;

Example 4.2.25 *Assume the following table.*

Degree (degcode, name, subject)

Candidate (seatno, degcode, name, semester, month, year, result)

Marks (seatno, degcode, semester, month, year, papcode, marks)

[degcode – degree code, name – name of the degree (Eg. MSc.), subject – subject of the course (Eg. Physis), papcode – paper code (Eg. A1)]

Solve the following queries using SQL:

Write a SELECT statement to display,

(i) all the degree codes which are there in the candidate table but not present in degree table in the order of degcode.

(ii) the name of all the candidates who have got less than 40 marks in exactly 2 subjects.

(iii) the name, subject and number of candidates for all degrees in which there are less than 5 candidates.

(iv) the names of all the candidate who have got highest total marks in MSc. Maths.

Solution :

(i) SELECT C.degcode

```
FROM Candidate C,
WHERE NOT EXISTS
        (SELECT  D.degcode
        FROM Degree D
        WHERE D.degcode=C.degcode)
ORDER by C.degcode
```

(ii) SELECT C.name

```
FROM Candidate C, Degree D, Marks M
WHERE
C.seatno=M.seatno AND C.degcode=D.degcode AND C.degcode=M.degcode AND M.marks<40
GROUP BY C.seatno
HAVING count(D.subject)=2;
```

(iii) SELECT D.name,D.subject,count(*)

```
FROM degree D, Candidate C
WHERE D.degcode=C.degcode
HAVING( SELECT count(*) FROM Candidate <5);
```


(iv) SELECT C.name

```
FROM Candidate C, Degree D, Marks M
WHERE
D.degname='MSc' AND D.subject='Maths' AND C.degcode=D.degcode AND C.seatno=M.seatno
AND
M.marks= (SELECT max(M.marks) FROM Marks M)
```

Example 4.2.26 *Consider a student registration database comprising of the below given table schema.*

Student File

Student Number	Student Name	Address	Telephone

Course File

Course Number	Description	Hours	Professor Number

Professor File

Professor Number	Name	Office

Registration File

Student Number	Course Number	Date

Consider a suitable sample of tuples / records for the above mentioned tables and write DML statements (SQL) to answer for the queries listed below.

i) Which courses does a specific professor teach ?

ii) What courses are taught by two specific professors ?

iii) Who teaches a specific course and where is his/her office ?

iv) For a specific student number, in which courses is the student registered and what is his/her name ?

v) Who are the professors for a specific student ?

vi) Who are the students registered in a specific course ?

Solution :

(i)

```
SELECT P.name,C.description
FROM Professor P, Course C
WHERE P.ProfessorNumber=C.ProfessorNumber
HAVING count(DISTINCT P.name)=2
```

(ii)

```
SELECT P.name,C.description
FROM Professor P, Course C
WHERE P.ProfessorNumber=C.ProfessorNumber
```

(iii)

```
SELECT P.name,P.office, C.description
FROM Professor P, Course C
WHERE P.ProfessorNumber=C.ProfessorNumber
```

(iv)

```
SELECT S.StudentNumber,S.StudentNumber,C.Description
FROM Student S, Course C, Registration R
WHERE S.StudentNumber=R.StudentNumber AND C.CourseNumber=R.CourseNumber
```

(v)

```
SELECT S.StudentName, P.Name
FROM Student S, Course C, Professor P, Registration R
WHERE C.ProfessorNumber=P.ProfessorNumber
        AND C.CourseNumber=R.CourseNumber
        AND S.StudentNumber=R.StudentNumber
        GROUP BY P.ProfessorNumber
```

(vi)

```
SELECT S.StudentName, C.Description
FROM Student S, Course C, Registration R
WHERE S.StudentNumber=R.StudentNumber
AND R.CourseNumber=C.CourseNumber
GROUP BY C.CourseNumber
```

Example 4.2.27 *Consider following RESORT database*

RESORT(resortno, resortname, resorttype, resortaddr,resortcity,numsuite)

SUITE (suiteno, resortno, suiteprice)

RESERVATION (reservationno, resorttype, resortaddr, resortcity , numsuite)

VISITOR(visitno, firstname, lastname, visitoraddr)

i) Write the SQL to list full details of all the resorts on Los Angeles.

ii) Write the SQL to list full details of all resorts having number of suites more than 30.

iii) Write the SQL to list visitors in ascending order by firstname.

VTU : July-18, Marks 6

Solution : (i) SELECT * FROM RESORT WHERE resortcity ='Los Angeles';

(ii) SELECT * FROM RESORT WHERE numsuite>30;

(iii) SELECT * FROM VISITOR ORDER BY firstname;

Example 4.2.28 *Consider the following database and answer the queries-*

WORKS(Pname,Cname,Salary)

LIVES(Pname, Street,City)

LOCATED_IN(Cname,City)

Write the following queries in SQL :

(i) List the names of the people who work for the company 'Wipro' along with cities they live in.

(ii) Find the names of the persons who do not work for 'Infosys'

(iii) Find the people whose salaries are more than that of all of the 'oracle' employees.

(iv)Find the persons who works and lives in the same city.

VTU : Jan.-19, Marks 10

Solution :

(i) SELECT L.Pname, L.City
FROM Works W, Lives L
Where W.Cname = "Wipro" and W.Pname = L.Pname;

(ii) SELECT Pname
FROM WORKS
WHERE Cname ! = 'Infoysy';

(iii) SELECT Pname
FROM WORKS
WHERE Salary > ALL
(SELECT Salary
FROM WORKS
WHERE Cname = "Oracle");

(iv) SELECT W.Pname
FROM WORKS W, LIVES L, LOCATED_IN I
WHERE W.Cname = I.Cname AND W.Pname = L.Pname
AND I.City = L.City

Example 4.2.29 *Write the SQL queries for the following relational schema :*

Sailors(Sid, Sname, Rating, Age)

Boats(Bid, Bname,color)

Reserve(Sid, Bid, Day)

i) Retrieve the Sailor's name who have reserved red and green boat

ii) Retrieve the no of boats which are not reserved.

iii) Retrieve the Sailors name who have reserved boat number 103

iv) Retrieve the Sailors name who have reserved all boats.

VTU : July-19, Marks 8

Solution :

```
(i) SELECT S.sname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid=R.sid
 AND R.bid=B.bid
 AND ( B.color='red' AND B.color='green')
(ii) SELECT Sname
FROM Sailors
WHERE Sid NOT IN
 (SELECT Sid
 FROM Reserve, Sailors
 WHERE Reserve.sid=Sailors.sid )
(iii) SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND bid=103
(iv) SELECT S.sname
FROM Sailors S
WHERE NOT EXISTS (( SELECT B.bid
 FROM Boats B )
 EXCEPT
 ( SELECT R.bid
 FROM Reserves R
 WHERE R.sid=S.sid ))
```

Example 4.2.30 *Write SQL queries for the following relational schema*

CUSTOMER(CID, CNAME,EMAIL,ADDR,PHONE)

ITEM(ITEM_NO,ITEM_NAME,PRICE, BRAND)

SALES(CID,ITEM_NO,#ITEMS, AMOUNT,SALE_DATE)

SUPPLIER(SID, SNAME, SPHONE, SADDR)

SUPPLY(SID, ITEM_NO, SUPPLY_DATE, QTY)

(i) List the items purchased by Customer 'Prasanth'

(ii) Retrieve items supplied by all suppliers starting from 1st Jan 2019 to 30th Jan 2019.

(iii) Get the details of customers whose total purchase of items worth more than 5000 rupees.

(iv) List total sales amount, total items, average sale amount of all items.

(v) Display customers who have not purchased any items

VTU : Jan.-20, Marks 8

Solution :

```
(i) SELECT ITEM_NAME
FROM ITEM I,CUSTOMER C and SALES S
WHERE C.CID= S.CID AND I.ITEM_NO = S.ITEM_NO AND C.CNAME='Prasanth'
(ii) SELECT ITEM_NAME
FROM ITEM, SUPPLIER, SUPPLY
WHERE ITEM.ITEM_NO=SUPPLY.ITEM_NO AND
AND SUPPLY.SID=SUPPLIER.SID AND
```



```
SUPPLY.SUPPLY_DATE '1 JAN 2019' AND '30 JAN 2019'
(iii) SELECT CID, CNAME, EMAIL,ADDR,PHONE
FROM CUSTOMER,SALES
WHERE CUSTOMER.CID=SALES.CID AND SALES.AMOUNT> =5000
(iv) SELECT
  SUM(AMOUNT),#item,AVG(AMOUNT)
FROM   SALES
WHERE SALES.ITEM_NO = ITEM.ITEM_NO
(v) SELECT CID, CNAME, EMAIL,ADDR,PHONE
FROM CUSTOMER
WHERE CID
NOT IN (
SELECT CID
FROM SALES
)
```

4.3 Specifying Constraints as Assertions and Action Triggers

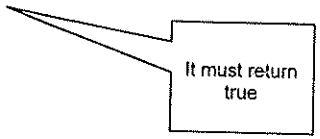
VTU : July-19, Jan.-20, Marks 8

In this section, we will discuss two important features of SQL and those are Assertions and triggers

4.3.1 Assertion

- An assertion is a predicate expressing a condition we wish the database to always satisfy.
- When created, the expression must be true.
- DBMS checks the assertion after any change that may violate the expression.
- **Syntax for creating an assertion**

CREATE ASSERTION <assertion-name>CHECK <predicate>



- **Example**

Consider following relation of customer

customer(customer_name, customer_street, customer_city)

If we want that the customer city should not be NULL then the Assertion can be written as follows

```
CREATE ASSERTION city_not_null Check
( NOT EXISTS
(Select  *
  From   customer
  Where  customer_city is null));
```

Example 4.3.1 *Write a SQL program to create an assertion to specify the constraint that the salary of an employee must not be greater than the salary of manager of the department that the employee works for in the COMPANY database.*

Solution :

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS
( SELECT *
FROM EMPLOYEE E, EMPLOYEE M,DEPARTMENT D
WHERE E.Salary>M.Salary
AND E.Dno=D.Dnumber
AND D.Mgr_ssn=M.Ssn ) );
```

4.3.2 Triggers

- A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database.
- It is specified by DBA.
- A trigger description contains **three parts :**
  - i) **Event :** A change to the database that activates the trigger.
  - ii) **Condition :** A query or test that is run when the trigger is activated. It is a true false kind of statement.
  - iii) **Action :** A procedure that is executed when the trigger is activated and its condition is true.
- Trigger is executed when the database is modified in a way that matches the event specification.
- An insert, delete, or update statement could activate a trigger.
- **Example :** The trigger called **init_Stud** initializes a counter variable before every execution of an INSERT statement that adds tuples to the Students relation. The trigger called **incr_Stud** increments the counter for each inserted tuple that satisfies the condition age < 18.

```
CREATE TRIGGER init_Stud ◄————Event
BEFORE INSERT ON Students
   DECLARE
        count INTEGER:
   BEGIN ◄—— Action
     count := 0;
   END
   CREATE TRIGGER incr_Stud AFTER INSERT ON Students ◄——Event
   WHEN (new.age < 18) ◄—— Condition is 'new' is just-inserted tuple
   FOR EACH ROW
   BEGIN ◄—— Action; a procedure
     count := count + 1;
   END
```


Types of Triggers

1) Row level trigger :

- Row level trigger is executed when each row of the table is inserted/ updated/ deleted.
- If it is a row level trigger, then we have to explicitly specify it while creating the trigger.
- Also, we have to specify the WHEN (condition) in the trigger.

2) Statement level trigger :

- This trigger will be executed only once for DML statement.
- This DML statement may insert / delete/ update one row or multiple rows or whole table.
- Irrespective of number of rows, this trigger will be fired for the statement.
- If we have not specified the type of trigger while creating, by default it would be a statement level trigger.

Example 4.3.2 *Write a trigger in SQL to call a stored procedure INFORM_SUPERVISOR(), whenever a new record is inserted or updated. Check whether an employee's salary is greater than the salary of his or her direct supervisor in the COMPANY database.*

VTU : Jan.- 20, Marks 7

Solution:

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
ON EMPLOYEE
FOR EACH ROW
WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
WHERE SSN = NEW.SUPERVISOR_SSN ) )
INFORM_SUPERVISOR(NEW.Supervisor_ssn,NEW.Ssn );
```

Difference between Assertion and Trigger

Sr. No.	Assertion	Trigger
1.	Assertions do not modify the data. They only check certain condition.	Triggers are powerful than assertions because they can check the condition as well as they can modify the data.
2.	Assertions are not linked to specific tables in the database and not linked to specific events.	Triggers are linked to specific tables and specific events.
3.	All assertions can be implemented as triggers.	All triggers cannot be implemented as assertions.
4.	Oracle does not support assertion.	Oracle support Triggers.

University Questions

1. *How are triggers and assertions defined in SQL ? Explain.* VTU - July 19 Marks 8
2. *What are assertions and triggers in SQL ? Write a SQL program to create an assertion to specify the constraint that the salary of an employee must not be greater than the salary of the department. The employee works for in the COMPANY database.* VTU - Jan 20 Marks 7

4.4 Views in SQL

VTU - July 18, 19, Jan 20 Marks 8

- Views in SQL are kind of virtual tables.
- A view also has rows and columns as they are in a real table in the database.
- We can create a view by selecting fields from one or more tables present in the database.
- A view can either have all the rows of a table or specific rows based on certain condition.

Creating View

We can create a view using CREATE VIEW statement. The syntax is

```
CREATE VIEW name_of_view AS
SELECT column1,column2,...
FROM table_name1,table_name2,...
WHERE condition;
```

Example

i) **Creating a view using single table :** Consider table Empolyee and create a view EmployeeDetails whose Salary is <10000

EmpID	EName	Salary
101	Archana	20000
102	Madhura	5000
103	Poonam	8000
104	Sharda	15000
105	Monika	7000

```
CREATE VIEW EmployeeDetails(EmpID, EName) AS
SELECT E.EmpID, E.EName
FROM Employee E
WHERE E.Salary > 10000
```


The **Output** will be -

EmpID	EName
102	Madhura
103	Poonam
105	Monika

ii) **Creating a view from multiple table :** In this example we will create a view from two tables Employee and Department.

Employee

EmpID	EName	Salary
101	Archana	20000
102	Madhura	5000
103	Poonam	8000
104	Sharda	15000
105	Monika	7000

Department

EmpID	DName
101	Accounts
104	Sales
105	Sales

Now we need to create a view named Employee_dept_Details in which the names and salary of employees belonging to Sales department is displayed. The SQL statement will then be –

```
CREATE VIEW Employee_dept_Details(EName,Salary) AS
    SELECT E.EName,E.Salary
    FROM Employee E, Department D
    WHERE E.EmpID=D.EmpID AND D.DName='Sales'
```

The output will be

EName	Salary
Sharda	15000
Monika	7000

View Update

- The SQL UPDATE VIEW command can be used to modify the data of a view.
- All views are not updatable. So, UPDATE command is not applicable to all views.

- An updatable view is one which allows performing a UPDATE command on itself without affecting any other table.
- **Syntax for updating view**

```
UPDATE < view_name >
SET<column1>=<value1>,<column2>=<value2>,....
WHERE <condition>;
```

- **Example**

```
UPDATE VIEW Employee_dept_Details
SET Salary=1000
WHERE EName='Monika';
This view can be viewed by using following query
SELECT * FROM Employee_dept_Details;
```

Rules for updating the views

The view can be updated in following conditions :

(1) The view can be defined based on one and only one table.

(2) The SELECT statement should not have DISTINCT keyword.

(3) The View should not have all NOT NULL values.

(4) The view should not be created from nested and complex queries.

(5) The SELECT statement which is used to create the view should not include GROUP BY clause or ORDER BY clause.

(6) The view should not have any field made out of aggregate functions.

(7) Any selected output fields of the view must not use constants, strings or value expressions.

Dropping View

For deleting a view, the DROP command is used.

Syntax

```
DROP VIEW view_name;
```

Example

```
DROP VIEW Employee_dept_Details;
```

Difference between View and Table

- Table is a database object that contains the data in row and column form. View is also database object and it is virtual table which is built on the top of the other tables.
- The table contains the data while view does not hold data itself.

- A table is designed with a limited number of columns and an unlimited number of rows while a view is designed as a virtual table that is extracted from a database.

Example 4.4.1 *Consider following company database :*

EMP(Name, SSn, Salary, Supersn, dno)

DEPT (dnum, dname, mgrssn)

DEPT_LOC (dnum, dlocation)

PROJECT(Pname, Pnumber, Plocation, dnum)

WORKS_ON(Essn, dept_name, sex)

Write SQL queries for the following:

i) Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.

ii) Retrieve the names of employees who make atleast 10000 more than the employee who is paid the least in the company.

iii) A view that has the employee name, supervisor name and employee salary for each employee who works in the Research' department.

iv) A view that has the project name, controlling department name, number of employees and total hours worked per week on the project for each project with more than one employee working on it.

VTU : July-18, Marks 8

Solution : (i)

```
SELECT Name FROM EMP WHERE dno =
        ( SELECT dno FROM EMP WHERE Salary =
            ( SELECT MAX(Salary) FROM EMP) )

(ii) SELECT Name FROM EMP WHERE Salary >= 10000 +
( SELECT MIN(Salary) FROM EMP)
(iii)
CREATE VIEW RESEARCH_EMPLOYEE_INFO (Lname, Fname,Supervisor,Salary) AS
SELECT E.Lname,E.Fname,S.Lname,E.Salary FROM
EMPLOYEE E, EMPLOYEE S, DEPARTMENT D
WHERE Dname ='Reasearch' AND
D.Dnumber=E.Dno AND E.super_ssn =S.ssn;
(iv) CREATE VIEW PROJECT_INFO AS
SELECT Pname,Dname,COUNT(WO.Essn),SUM(WO.Hours)
FROM PROJECT P, DEPARTMENT D, WORKS_ON WO
WHERE
P.Dnum = D.Dnumber AND
P.Pnumber = WO.Pno GROUP BY Pno
HAVING COUNT(WO.Essn)>1
```

University Questions

1. *How are views created and dropped ? Explain how the views are implemented and updated.* **VTU : July-19, Marks 8**
2. *How do you create a view in SQL ? Give examples. Can you update a view table? If yes, how ? If not, why not ? Discuss.* **VTU : Jan.-20, Marks 6**

4.5 Schema Change Statements in SQL

VTU : July-18, Jan.-19,20, Marks 6

Schema can be changed by adding or dropping tables, attributes and constraints. These commands are also known as **schema evolution commands.**

There are two commands that change the schema and those are DROP and ALTER.

Let us discuss these commands in detail

4.5.1 The DROP Command

- The DROP command is used to remove the object (table, domains and constraints) from the database. There are two options for the DROP command - CASCADE and RESTRICT.
- To use the RESTRICT option, the user must first individually drop each element in the schema, then drop the schema itself. That means, the schema is dropped only if it has no elements in it, otherwise the DROP command can not be executed.
- Otherwise to remove completely some database schema CASCADE option is chosen. For example – to remove the **Student_database**

  DROP SCHEMA Student_database **CASCADE**;
- If the table is to be deleted then the SQL command would be -

  DROP TABLE Student **CASCADE**;
- The DROP TABLE command not only deletes all the records in the table if successful, but also removes the table definition from the catalog. If it is desired to delete only the records but to leave the table definition for future use, then the DELETE command. For example -
- The following SQL statement deletes all rows in the "Students" table, without deleting the table :

  DELETE FROM Students;

4.5.2 The ALTER Command

There are SQL commands for alteration of table. That means we can add new column or delete some column from the table using these alteration commands.

Syntax for Adding columns

```
ALTER TABLE table_name
ADD column_name datatype;
```

Example

Consider following table

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. Road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute following command

```
ALTER TABLE Customers
ADD Email varchar(30);
```

Then the result will be as follows -

AadharNo	FirstName	MiddleName	LastName	Address	City	Email
111	AAA	BBB	CCC	M.G. road	Pune	NULL
222	DDD	EEE	FFF	Shivaji nagar	Pune	NULL
333	GGG	HHH	III	Chandani chowk	Delhi	NULL
444	JJJ	KKK	LLL	Viman nagar	Mumbai	NULL

Syntax for Deleting columns

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Example

Consider following table -

AadharNo	FirstName	MiddleName	LastName	Address	City
111	AAA	BBB	CCC	M.G. road	Pune
222	DDD	EEE	FFF	Shivaji nagar	Pune
333	GGG	HHH	III	Chandani chowk	Delhi
444	JJJ	KKK	LLL	Viman nagar	Mumbai

If we execute following command

```
ALTER TABLE Customers
DROP COLUMN Address;
```

Then the result will be as follows -

AadharNo	FirstName	MiddleName	LastName	City
111	AAA	BBB	CCC	Pune
222	DDD	EEE	FFF	Pune
333	GGG	HHH	III	Delhi
444	JJJ	KKK	LLL	Mumbai

University Questions

1. *Discuss how each of the following construct is used in SQL and discuss various options for each construct :*
   (i) Nested Queries (ii) Aggregate Functions (iii) Triggers (iv) Views and their updatability
   (v) Schema Change Statement (vi) Group By and Having Clause
   VTU : July-18, Marks 6
2. *Explain with example in SQL - Drop command in SQL.*
   VTU : Jan.-19, Marks 2
3. *With real world examples explain - schema change statement in SQL*
   VTU : Jan.-20, Marks 3

Part II : Application Development

4.6 Accessing Database from Applications VTU : July-18,19, Jan.-19,20, Marks 8

For accessing the database from the applications we need to write an application program in a programming language such as C or Java and then we need to embed the SQL statements within it. Such languages are called host languages. The embedded SQL statements then interact with the database and fetch the required data from the database.

There are three commonly used techniques for accessing the database from the application and those are -

(1) Embedded SQL (2) Cursors (3) Dynamic SQL.

Let us discuss them with the help of examples

4.6.1 Embedded SQL

- The programing module in which the SQL Statements are embedded is called Embedded SQL module.
- It is possible to embed SQL statements inside the programming language such as C, C++, PASCAL,Java and so on.
- It allows the application languages to communicate with DB and get requested result.

- The high level languages which supports embedding SQLs within it are also known as **host language**.
- An embedded SQL program must be processed by a special preprocessor prior to compilation. The preprocessor replaces embedded SQL requests with host-language declarations and procedure calls that allow runtime execution of the database accesses. Then, the resulting program is compiled by the host-language compiler. This is the main distinction between embedded SQL and JDBC or ODBC.

Declaring Variables and Exceptions

- We can declare the variables in the host language which are prefixed by the colon in SQL statements.
- These variables must be declared within a block which is starting by the command EXEC SQL BEGIN DECLARE SECTION and ends by the command EXEC SQL END DECLARE SECTION.
- Then the declaration of the variables is done using the syntax
  Data_type variable_name;
- For example –

```
EXEC SQL BEGIN DECLARE SECTION;
   int OrderID;       /* Employee ID (from user)       */
   int CustID;        /* Retrieved customer ID         */
   char SalesPerson[10]  /* Retrieved salesperson name  */
   char Status[6]     /* Retrieved order status        */
EXEC SQL END DECLARE SECTION;
```

Embedding SQL Query

- Similarly we can embed the SQL query using SELECT, INSERT, DELETE, or UPDATE statement in the host language. Again the query must begin with the command EXEC SQL. For example –

```
EXEC SQL SELECT CustID, SalesPerson, Status
   FROM Orders
    WHERE OrderID = :OrderID
    INTO :CustID, :SalesPerson, :Status;
```

Example of Embedded SQL

Following program prompts the user for an order number, retrieves the customer number, salesperson, and status of the order, and displays the retrieved information on the screen.

```
int main() {
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
```

```
    int OrderID;        /* Employee ID (from user)       */
    int CustID;         /* Retrieved customer ID         */
    char SalesPerson[10]  /* Retrieved salesperson name  */
    char Status[6]      /* Retrieved order status        */
  EXEC SQL END DECLARE SECTION;

  /* Set up error processing */
  EXEC SQL WHENEVER SQLERROR GOTO query_error;
  EXEC SQL WHENEVER NOT FOUND GOTO bad_number;

  /* Prompt the user for order number */
  printf ("Enter order number: ");
  scanf_s("%d", &OrderID);

  /* Execute the SQL query */
  EXEC SQL SELECT CustID, SalesPerson, Status
    FROM Orders
    WHERE OrderID = :OrderID
    INTO :CustID, :SalesPerson, :Status;

  /* Display the results */
  printf ("Customer number:  %d\n", CustID);
  printf ("Salesperson: %s\n", SalesPerson);
  printf ("Status: %s\n", Status);
  exit();

query_error:
  printf ("SQL error: %ld\n", sqlca->sqlcode);
  exit();
bad_number:
  printf ("Invalid order number.\n");
  exit();
}
```

Features of Embedded SQL

(1) It is easy to use.

(2) It is ANSI/ISO standard programming language.

(3) It requires less coding

(4) The precompiler can optimize execution time by generating stored procedures for the Embedded SQL statements.

(5) It is identical over different host languages, hence writing applications using different programming languages is quite easy.

Example 4.6.1 *Consider the relation student(Reg.No., name, mark, and grade). Write embedded dynamic SQL program in C language to retrieve all the students' records whose mark is more than 90.*

Solution :

```
int main()
{
/* Begin program */
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION
int Reg_No;
char name[10][10];
 float marks;
char grade;

EXEC SQL END DECLARE SECTION
EXEC SQL WHENEVER SQLERROR STOP
EXEC SQL SELECT Reg_No,name,marks,grade
 FROM Student
 WHERE marks>90
 INTO :Reg_No,:name,:marks,:grade;
  /* Display the results */
  printf ("Registration number:  %d\n", Reg_No);
  printf ("Name: %s\n", name);
  printf ("Marks: %f\n", marks);
  printf ("Grade: %c\n", grade);
  exit();
EXEC SQL DISCONNECT
/* End program */
}
```

Example 4.6.2 *Write a complete high level language program(in Java or C) to display the rows of customer table created in oracle having <custid,custname,balance> columns with embedded SQL.*

VTU : Jan.-20, Marks 8

Solution :

```
int main()
{
      /* Begin program */
      EXEC SQL INCLUDE SQLCA;
      EXEC SQL BEGIN DECLARE SECTION
      int custID;
      char name[10][10];
      float Balance
```

```
    EXEC SQL END DECLARE SECTION
    EXEC SQL WHENEVER SQLERROR STOP
    EXEC SQL SELECT custid,custname,balance
    FROM Customer
    INTO :custID,:name,:Balance;
    /* Display the results */
    printf ("Customer ID:  %d\n",custID);
    printf ("Customer Name: %s\n",name);
    printf ("Balance Amount: %f\n",Balance);
    exit();
    EXEC SQL DISCONNECT
    /* End program */
}
```

University Question

1. *With program segment, explain retrieving of tuples with embedded SQL in C.*

VTU : Jan.-19, Marks 6

4.6.2 Cursors

- When an SQL statement is processed, Oracle creates a memory area known as **context area**. A cursor is a **pointer to this context area**.
- It contains all information needed for processing the statement.
- In PL/SQL, the context area is controlled by Cursor.
- A cursor contains information on a SELECT statement and the rows of data accessed by it.
- The cursor is used to fetch and process the rows returned by SQL statement one at a time.
- There are two types of cursors -
  - Implicit cursor
  - Explicit cursor

(1) Implicit cursor

- Whenever Oracle executes an SQL statement such as SELECT INTO, INSERT, UPDATE, and DELETE, it creates an implicit cursor.
- The Implicit Cursor is the Default Cursor in PL/SQL block.

- Orcale provides some attributes known as Implicit cursor's attributes to check the status of DML operations. Some of them are as follows -
  - **%ROWCOUNT :** It Returns the number of rows influenced by an UPDATE, DELETE or an INSERT statement.
  - **%NOTFOUND :** It Returns TRUE if an UPDATE, DELETE or an INSERT statement influenced zero rows or a SELECT INTO statement returned no rows. Else, it Returns a FALSE.
  - **%FOUND :** It Returns TRUE if an UPDATE, DELETE or an INSERT statement influenced one or more rows or a SELECT INTO statement returned one or more rows. Else, it returns FALSE.
  - **%ISOPEN :** It Returns FALSE for Implicit cursors as Oracle closes the PL/SQL cursor by default after processing its associated PL/SQL statement.

Programming Example : Following is an example program that uses attributes of implicit cursor to demonstrate number of rows affected when SELECT query is executed for Student table

```
SET SERVEROUTPUT ON;
DECLARE
 s_name student.name%type;
BEGIN
 SELECT name INTO s_name
 FROM student
 WHERE marks>=70;
 DBMS_OUTPUT.PUT_LINE('Number of rows processed: '||sql%rowcount);
END;
/
```

Output

```
Number of rows processed:5
```

(2) Explicit cursor

- Explicit cursors are used when you are executing a SELECT statement query that will return more than one row.
- Cursors can process one record at a given point of time even though it stores more than one record.
- An explicit cursor is defined in the declaration section of the PL/SQL Block.
- Explicit cursors are used if you need to have better control over the Context Area via Cursor.

Syntax for Declaration of Explicit Cursor

```
CURSOR cursor_name
IS
SELECT statement
```

For example

```
CURSOR MyCursor
IS
SELECT * FROM Student;
```

The explicit cursor works in four stages –

1) **Declaration of cursor :** The declaration of cursor is in declaration section of PL/SQL block. The name of the cursor requires to be defined along with the SELECT Statement.

Syntax

```
CURSOR cursor-name IS
SELECT statement;
```

2) **Open the cursor :** Opening the Cursor also means allocating the memory for the Cursor in the Context Area which thereby makes it sufficient to fetch and store records in it.

Syntax

```
OPEN cursor-name;
```

3) **Fetch the cursor :** Fetching the Cursor involves retrieval of data using the Fetch statement. It is used to help the Cursor process and access records or rows at a time.

Syntax

```
FETCH cursor_name INTO variable_list;
```

4) **Close the cursor :** Once the work associated for a Cursor to be completed is accomplished, it is necessary to release the Allocated Memory of the Cursor to let other tasks occupy memory.

Syntax

```
Close cursor_name;
```

- **Example :** The cursor is created to display the names of students who have scored more than 70 marks. The name of the cursor is **My_Cursor**. The SELECT query associated with it is


```
    SELECT name FROM STUDENT WHERE Marks >70;
DECLARE
 s_name VARCHAR2(30);
 --Declare Cursor
 CURSOR My_Cursor IS
 SELECT name FROM Student
 WHERE Marks>70;
BEGIN
 OPEN My_Cursor;
 Loop
  FETCH My_Cursor INTO s_name;
  DBMS_OUTPUT.PUT_LINE(s_name);
  EXIT WHEN My_Cursor %NOTFOUND;
 END Loop;
 CLOSE My_Cursor;
END;
```

Example 4.6.3 *Write block of using explicit cursor that will display the customer name, the fixed deposit number and the fixed deposit amount of the first 5 customers holding the highest amount in fixed deposits.*

Use the following database :

cust_mstr(custno,name, occupation)

fd_dtls(fd_ser_no,fd_no,type,period,opndt,duedt,amt,dueamt)

acct_fd_cust_dtls(acct_fd_no,custno)

Solution :

```
DECLARE My_Cur IS
SELECT name, fd_no,amt
FROM cust_mstr C, fd_dtls F,acct_fd_cust_dtls A
WHERE C.custno=A.custNo AND F.fd_no=A.acct_fd_no
ORDER BY amt DESC;

s_name   VARCHAR2(30);
s_fd_no  fd_dtls.fd_no%TYPE;
s_amt    fd_dtls.amt%TYPE;
BEGIN
OPEN My_Cur;
          LOOP FETCH My_Cur INTO s_name,s_fd_no,s_amt;
                DBMS_OUTPUT.PUT_LINE(s_name ||'  '||s_fd_no|| ' '||s_amt);
EXIT WHEN (My_Cur%ROWCOUNT-1)=5 OR  My_Cur %NOTFOUND;
END LOOP;
      CLOSE My_Cur;
END;
```

Example 4.6.4 *Write a cursor to display the names and branch of all students from the STUDENT relation.*

Solution :

```
SET SERVEROUTPUT ON;
DECLARE
  s_name VARCHAR2(30);
  s_branch VARCHAR2(20);
  --Declare Cursor
  CURSOR My_Cursor IS
  SELECT name,Branch FROM Student;
BEGIN
  OPEN My_Cursor;
  Loop
    FETCH My_Cursor INTO s_name,s_branch;
    DBMS_OUTPUT.PUT_LINE(s_name||'    '||s_branch);
    EXIT WHEN My_Cursor %NOTFOUND;
  END Loop;
  CLOSE My_Cursor;
END;
```

4.6.3 Dynamic SQL

- Dynamic SQL is a programming technique which allows to build the SQL statements dynamically at runtime.
- Dynamic SQL statements are not embedded in the source program but stored as strings of characters that are manipulated during a program's runtime.
- These SQL statements are either entered by a programmer or automatically generated by the program.
- Dynamic SQL statements also may change from one execution to the next without manual intervention.
- Dynamic SQL facilitates automatic generation and manipulation of program modules for efficient automated repeating task preparation and performance.
- Dynamic SQL facilitates the development of powerful applications with the ability to create database objects for manipulation according to user input.
- The simplest way to execute a dynamic SQL statement is with an EXECUTE IMMEDIATE statement. This statement passes the SQL statement to the DBMS for compilation and execution.

Sr. No.	Embedded SQL	Dynamic SQL
1	SQL statements are compiled at compile time.	SQL statements are compiled at run time.
2	It is more efficient.	It is less efficient.
3	It is less flexible.	It is more flexible.
4	It is used in the situations where data is distributed uniformly	It is used in situations where data is distributed non uniformly.

University Question

1. *What is dynamic SQL and how it is different from embedded SQL.*

VTU : July-18, Marks 4

4.7 An Introduction to JDBC

VTU : Jan.-20, Marks 3

- JDBC stands for Java DataBase Connectivity.
- JDBC is nothing but an API (i.e. Application Programming Interface).
- It consists of various classes, interfaces, exceptions using which Java application can send SQL statements to a database. The SQL is a Structured Query Language used for accessing the database.
- JDBC is specially used for having connectivity with the RDBMS packages (such as Oracle or MYSQL) using corresponding JDBC driver.

4.7.1 JDBC Classes and Interfaces

The **JDBC API** is a set of classes, interfaces and exceptions used for establishing connection with data source. This JDBC API is defined in the **java.sql** and **javax.sql** packages. We use following core JDBC classes and interfaces that belong to **java.sql** package.

- **DriverManager -** When Java application needs connection to the database it invokes the **DriverManager** class. This class then loads JDBC **drivers** in the memory. The DriverManager also attempts to open a connection with the desired database. For JSP-MYSQL connectivity we can get DriverManager class as -

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

1) **Connection -** This is an interface which represents connectivity with the data source. The **connection** is used for creating the **Statement** instance.

```
Connection conn=
DriverManager.getConnection("jdbc:mysql://localhost:3306/databaseName","userName","password");
```

2) **Statement** - This interface is used for representing the SQL statements. Some examples of SQL statements are
   - SELECT * FROM students_table;
   - UPDATE students_table set name='Neel' where roll_no='1';
   - DELETE from students_table WHERE roll_no='10';

The code for statement is as follows -

```
Statement stmt=conn.createStatement();
stmt.executeQuery("select * from mytab");
```

There are two specialised Statement types: **PreparedStatement** and **CallableStatement**. The **PreparedStatement** represents the precompiled SQL statements.

For instance :

```
SELECT * from students_table WHERE name=?
```

The placeholder represented by ? is used in this type of statement. There are special type setter methods that assign the values to the placeholders before the SQL statement is executed.

CallableStatement is represents the stored procedures. These are similar to **PreparedStatement**. In this type of Statements we can assign the methods for the types of output arguments.

3) **ResultSet** - This interface is used to represent the database result set. After using SELECT SQL statement, the information obtained from the database can be displayed using **ResultSet**.

4) **SQLException** - For handling SQL exceptions this interface is used.

4.7.1.1 Statement Interface

(1) PreparedStatement :

- The **java.sql.PreparedStatement** interface object represents a precompiled SQL statement.
- This interface is used to efficiently execute SQL statements multiple times. That is when we want to insert a record in a table by putting different values at runtime.
- This statement is derived from the Statement class.
- The PreparedStatement interface can be created by calling PrepareStatement() method.
- The prepareStatement() is available in **java.sql.Connection** interface.
- The prepareStatement() method takes SQL statement in java format.


```
prepareStatement("insert into student values(?,?)").
```

where each ? represents the column index number in the table. If table **student** has **rollnumber** and **name** columns, then ? refers to **rollnumber**, 2nd ? refers to **name**.

Programming Example : Java Program to insert student record in the database using prepared statement.

```
import java.sql.*;
public class JDBCDemo5
{
 public static void main(String [ ] args)
 {
   Connection con = null;
   try
   {
        int rollnum=2;
        String name="Parth";
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver") ;
        con = DriverManager.getConnection("jdbc:odbc:My_database","","");
        PreparedStatement ps=con.prepareStatement("INSERT INTO My_table VALUES(?,?)");
        ps.setInt(1,rollnum);
        ps.setString(2,name);
        int result=ps.executeUpdate();
        if(result!=0)
                System.out.println("Values inserted in the table");
        else
                System.out.println("Values are not inserted in the table");
        String sql="SELECT * FROM My_table";
        ps=con.prepareStatement(sql);
        ResultSet rs=ps.executeQuery();
        System.out.println("Displaying the contents of the table...");
        while(rs.next())
        {
                int RollNo  = rs.getInt("Roll");
                String Name = rs.getString("StudName");
                //Display values
                System.out.print("Roll Number: " + RollNo);
                System.out.println(", Student Name: " + Name);
        }
        rs.close();
        ps.close();
     con.close();
 }
 catch (ClassNotFoundException e)
```

```
{
   System.err.println("Exception: "+e.getMessage());
  }
  catch (SQLException e)
  {
   System.err.println("Exception: "+e.getMessage());
  }

 }
}
```

(2) Callable Statement

- The callable statement is used when we want to access the **database stored procedures**.
- The stored procedure is basically a block of code which is identified by unique name.

Creating Procedure

The procedure can be created using following syntax.

```
DELIMITER //
CREATE PROCEDURE procedureName(paramters_list)
BEGIN
QL Statements to be executed
D //
```

Calling Procedure

When calling the stored procedure, the **CallableStatement** object is used. For this object three types of parameters are used.

Parameter	Description
IN	A parameter whose value is unknown when the SQL statement is created. Then the values can be associated with IN parameters with the setXXX() methods.
OUT	A parameter whose value is supplied by the SQL statement it returns. These values are obtained in OUT parameters with the getXXX() methods.
INOUT	A parameter that supplies input as well as accepts output parameter requires a call to the appropriate setXXX method.

- You can create an instance of a **CallableStatement** by calling the **prepareCall()** method on a connection object.

Programming Example

Step 1 : Creating Procedure listStudents()

```
DELIMITER //
      CREATE PROCEDURE listStudents()
      BEGIN
      SELECT * FROM my_table;
      END //
```

Step 2 : Creating JDBC code for calling the procedure

```
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
public class JDBCDemo6 {
      public static void main(String[] args)
      {
            Connection con=null;
            try
            {
//Getting the driver for MyQL
                  Class.forName("com.mysql.jdbc.Driver");
                  con = DriverManager.getConnection
                  ("jdbc:mysql://localhost/my_database","root","");
//Creating instance of CallableStatement using prepare call
            CallableStatement cs=con.prepareCall("CALL listStudents()");
            ResultSet rs=cs.executeQuery();
            while(rs.next()){
                  System.out.println(rs.getInt(1)+"\t"+rs.getString(2));
                  }
            }catch(Exception e){
                  e.printStackTrace();
            }
            finally {
                  try {
                        con.close();
                  }catch(SQLException e){
                        e.printStackTrace();
                  }
            }
      }

}
```

4.7.1.2 Role of ResultSet Interface

- The ResultSet interface is an important interface which is used to access the database table with general width and unknown length.
- The table rows are retrieved in sequence using **ResultSet** object. Within a row its column values can be accessed in any order.
- A ResultSet maintains a cursor pointing to its current row of data. Initially the cursor is positioned before the first row. The 'next' method moves the cursor to the next row.
- The ResultSet object can be created using **executeQuery**() method. For example

```
Statement statement = connection.createStatement();
ResultSet result = statement.executeQuery ("select * from my_table");
```

- There are various methods using which the data can be retrieved using the ResultSet object. These methods can be used with either column Name or with column Index. Few commonly used methods are

Sr. No.	Methods	Description
1.	public int getInt(int index)	Returns the integer value in the current row specified by the index.
2.	public int getInt(String Name)	Returns the integer value in the current row specified by the column Name.
3.	public Date getDate(int index)	Returns the Date value in the current row specified by index.
4.	Public Date getDate(String Name)	Returns the Date value in the current row specified by Name of the column.

Similar to **getInt** there are methods such as **getString, getBoolean, getByte, getFloat, getDouble** and so on.

4.7.2 Working of JDBC

Following is a way by which JDBC works -

1) First of all Java application establishes connection with the data source.
2) Then Java application invokes classes and interfaces from JDBC driver for sending queries to the data source.
3) The JDBC driver connects to corresponding database and retrieves the result.
4) These results are based on SQL statements which are then returned to Java application.
5) Java application then uses the retrieved information for further processing.

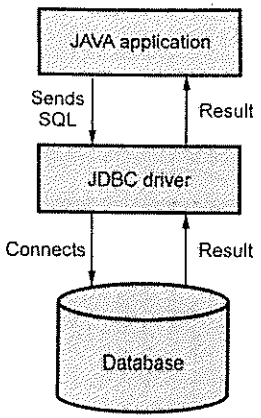


Fig. 4.7.1 Role of JDBC

Uses of JDBC

1. JDBC helps the client to store and retrieve the data to the databases.
2. JDBC allows the client to update the databases.

4.7.3 JDBC Driver Types

There are four JDBC Driver Types and those are -

Type 1 : JDBC-ODBCBridge

This driver translates all the JDBC calls into ODBC(Open Database Connectivity) calls and send them to ODBC driver. Thus JDBC access is via ODBC driver. The ODBC is a generic API. In this scenario the client database code must be present on the client machine.

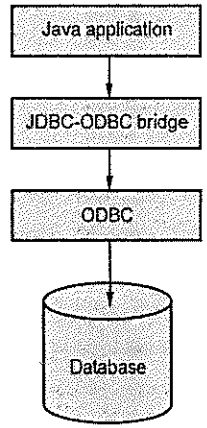


Fig. 4.7.2 JDBC-ODBC bridge

Type 2 : Native-API/Partly Java Driver

This driver translates all the JDBC calls into database-specific calls. This driver works specifically for particular database. For example MYSQL will have native MYSQL API. This type of driver directly communicates with the database server. Hence some binary code must be present on the client machine.

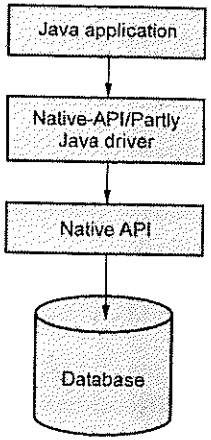


Fig. 4.7.3 Type - 2 driver

Type 3 : All JAVA/ Net Protocol Driver for Accessing Middleware Server

In this type of driver all the JDBC calls are passed through the network to the middle-ware server. The middleware server then translates the request to the database-specific native-connectivity interface and then the request is sent to the database server.

This driver is a server-based driver. This is also known as a re Java driver.

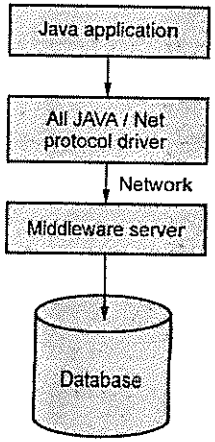


Fig. 4.7.4 Type - 3 driver

Type 4 : All JAVA / Native-Protocol Pure Driver

This type of driver converts the JDBC calls to network protocol used by the database directory so that the client application can directly communicate to the database server. This driver is also completely implemented in Java and hence it is referred as Pure Java driver.

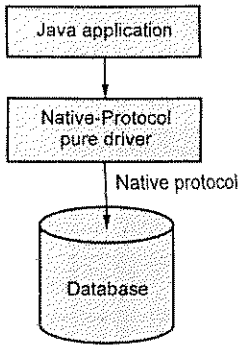


Fig. 4.7.5 Type - 4 driver

Example 4.7.1 *Assume that a database has a table Employee with two columns EmployeeID and Name. Assume that the administrator user id and password to access the database table are, scott and tiger. Write a JDBC program that can query and print all the entries in the table Employee.*

Solution : The JDBC code for displaying the employees in Employee table is as follows -

DisplayEmp.java

```
import java.io.*;
import java.util.*;
import javax.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class DisplayEmp extends HttpServlet
{
  public void service(HttpServletRequest request,HttpServletResponse response)
      throws IOException, ServletException
  {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Servlet Database Connectivity</title></head>");
    out.println("<body>");
    // connecting to database
    Connection con = null;
    Statement stmt = null;
    ResultSet rs=null;
    try
    {

      Class.forName("oracle.jdbc.OracleDriver");
```

```
    String dbURL = "jdbc:oracle:oci:@my_database";
                Properties properties = new Properties();
    properties.put("user", "scott");
    properties.put("password", "tiger");
    properties.put("defaultRowPrefetch", "20");
    con =DriverManager.getConnection (dbURL,properties);
    stmt = con.createStatement();
          rs=stmt.executeQuery("select * from Employee");
    out.print("<table border=1>");
    out.print("<tr><td>EmployeeID</td><td>Name</td></tr>");
    while(rs.next())
    {
      //Retrieve by column name
      int id  = rs.getInt("ID");
      String name = rs.getString("Name");
      out.print("<tr><td>"+id+"</td><td>"+name);
      out.print("</td></tr>");
    }
    out.println("</table></body></html>");
  }catch (SQLException e)
  {
    throw new ServletException("Servlet Could not display records.", e);
  }
  catch (ClassNotFoundException e)
  {
    throw new ServletException("JDBC Driver not found.", e);
  }
  finally
  {
    try
    {
      if(rs != null)
      {
        rs.close();
        rs = null;
      }
      if(stmt != null)
      {
        stmt.close();
        stmt = null;
      }
      if(con != null)
      {
        con.close();
        con = null;
      }
    }
```



```
      catch (SQLException e) {}
    }//end of finally
    out.close();
  }//end of service function
} //end of class
```

University Question

1. *With real world examples explain – JDBC.* **VTU : Jan.-20, Marks 3**

4.8 SQLJ

VTU : July -18, Marks 4

- SQLJ stands for SQL Java. It is very close to embedded SQL.
- In SQLJ the SQL syntax check is allowed at the compile time. Similarly in SQLJ the variables in the host language are always bound statically.
- All SQLJ statements have the prefix #sql. In SQLJ we retrieve the results of SQL queries with **iterator objects**. These are basically the cursors. An iterator is an instance of **iterator** class.
- The iterator goes through following steps –

Step 1 : Using the keyword iterator declare the iterator class. For example

```
#sql iterator Student(String name,int RollNo);
```

This statement creates a new Java Class that we can use to instantiate objects.

Step 2 : Instantiate an Iterator object from New Iterator class: We can instantiate our iterator in the statement. For example –

```
Student stud;
```

Step 3 : Initialize the iterator using SQL statement. For example

```
#sql stud = {
      SELECT Name, RollNo INTO :Name, :RollNo
      FROM Student WHERE subject = :subject
};
```

Step 4 : Iteratively, Read the Rows From the Iterator Object: This step is very similar to reading rows through a **ResultSet** object.

Step 5 : Finally close the **Iterator** object.

The above statements are used in the following SQLJ code. Here is a sample SQLJ program

```
String Name; int RollNo; String subject
#sql iterator Student (String Name, int RollNo);
Student stud;
```

```
// the application sets the subject
// execute the query and open the cursor
#sql stud = {
SELECT Name, RollNo INTO :name :RollNo
FROM Student WHERE subject = :subject
};
// retrieve results
while (stud.next()) {
        System.out.println(stud.Name() + ", " + stud.RollNo());
}
stud.close() ;
```

Advantages of SQLJ over JDBC

1. SQLJ programs require fewer lines of code than JDBC programs.
2. The code in SQLJ is shorter, and hence easier to debug.
3. SQLJ can perform syntactic and semantic checking on the code, using database connections at compile time.
4. SQLJ provides strong type-checking of query results and other return parameters, while JDBC values are passed to and from SQL without having been checked at compile time.
5. SQLJ provides a simplified way of processing SQL statements.

University Question

1. *What is SQLJ ? And how it is different from embedded SQL ?*

VTU : July -18, Marks 4

4.9 Stored Procedures

VTU : July-19, Jan.-20, Marks 8

- Stored procedure is a type of subprogram in PL/SQL block. It is a group of statements that can be called by its name.
- This is a subprogram that does **not return a value directly**.
- A procedure is created with the CREATE OR REPLACE PROCEDURE statement

Syntax

```
CREATE or REPLACE Procedure Procedure_Name
[(Parameter_Name [IN | OUT | IN OUT ] Type [...])]
[IS | AS]
BEGIN
      Procedure_Body
END;
```


- Where
  - **Procedure_Name** is used to specify the name of the procedure.
  - **CREATE** keyword is used to develop a new procedure and [OR REPLACE] option allows us to modify an existing procedure.
  - The optional **parameter list** contains Name and Types of Parameters.
- There are three **ways to pass parameters in procedures** -
  - **IN** represents that argument value will be passed from outside the Procedure. It is a read-only parameter. Parameters are passed by reference. Inside the procedure or a sub-program, an IN Parameter acts as a constant. It cannot be assigned a value. It is the default mode of parameter passing.
  - An **OUT** parameter returns a value to the calling program. OUT represents that this parameter will be used to return a value outside of the procedure.
  - An **IN OUT** parameter passes an initial value to the sub-program and returns an updated value to the caller. We can read and write values using this parameter.
- The Executable part is included in the Procedure Body.
- We can create a procedure without passing the parameters or by passing the parameters.
- WE can execute the procedure using **Execute** keyword.
- A standalone procedure can be called by using the EXECUTE keyword or by calling or mentioning the procedure by its name from a PL/SQL block.

Syntax

Execute [Procedure-Name];

4.9.1 Procedures without Parameter

Let us write a simple procedure for displaying some welcome message.

Step 1 : Create a procedure as follows

```
CREATE OR REPLACE PROCEDURE MyMessage ← Creation of procedure
IS
BEGIN
  dbms_output.put_line('Welcome friend!!!');
END;
/

EXECUTE MyMessage;  ← This is for executing the procedure
```

Step 2 : The output will be as follows -

Welcome friend!!!

Another way of execution of procedure

```
CREATE OR REPLACE PROCEDURE MyMessage ← Creation of procedure
IS
BEGIN
  dbms_output.put_line('Welcome friend!!!');
END;
/
BEGIN ← This is for calling the procedure
  MyMessage;
END;
/
```

4.9.2 Procedures with Parameters

Following example show the procedure for addition of two numbers. The result is stored in third variable which is an output variable.

Step 1 : We will create a procedure as follows

```
CREATE OR REPLACE PROCEDURE AddProc(x IN NUMBER, y IN NUMBER, z OUT NUMBER)
IS
BEGIN
 z:=x+y;
END;
```

Two input and one output parameter

Step 2 : The call to the procedure can be made as follows -

```
DECLARE
 NUMBER;
 NUMBER;
 NUMBER;

EGIN
 := 10;
 := 20;
AddProc(a,b,c);
ibms_output.put_line('Addition of two numbers:'||c);
ND;
```

4.9.3 Stored Procedure for Handling Database

Using a stored procedure we can access the database table and can perform insertion, deletion and updation of data.

Following is a simple example of procedure using which we insert some record in Student table.

Step 1 : Write a procedure as follows

```
CREATE OR REPLACE PROCEDURE InsertStudent(roll IN NUMBER,
name IN VARCHAR2,
marks IN NUMBER,
branch IN VARCHAR2)
IS
BEGIN
INSERT INTO student VALUES(roll,name,marks,branch);
END;
/
```

Step 2 : Now call the procedure as follows –

```
BEGIN
 InsertStudent(7,'Akash',78,'CIVIL');
 DBMS_OUTPUT.PUT_LINE('One row inserted in Student Table!!!');
END;
/
```

University Questions

1. *Explain the following :*
   i) Embedded SQL ii) Database stored procedure. **VTU : July-19, Marks 8**
2. *With real world examples explain - stored procedure.* **VTU : Jan.-20, Marks 3**

Part III : Internet Applications

4.10 The Three-tier Application Architecture **VTU : July-18, 19, Jan.-19, Marks 8**

The internet applications are based on three functional components namely -

(1) Data management (2) Application logic and (3) Presentation.

Before introduction of three tier architecture, there were two applications architecture that were used extensively and those were - Single Tier architecture and client server architecture.

Let us discuss them and then we will discuss three-tier architecture.

4.10.1 Single Tier and Client Server Architecture

Single Tier Architecture

- In this architecture application logic, presentation and data management all are combined in a single tier. For running the application, there was a use of mainframes. The application was accessible by **dumb terminals** that could perform only data input and display. It is as shown by following figure -

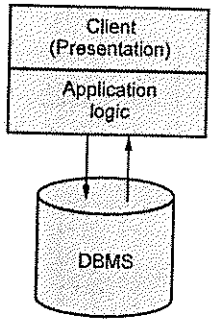


Fig. 4.10.1 Single tier architecture

- The advantage of this architecture is that it is easily maintained by central administrator.
- The main drawback of this architecture is that - it could support limited number of users.

Client Server Architecture

- The client server architecture is also called as **two tier architecture.**
- It consists of **Client computer** and **Server computer** which can communicate through well -defined protocol.

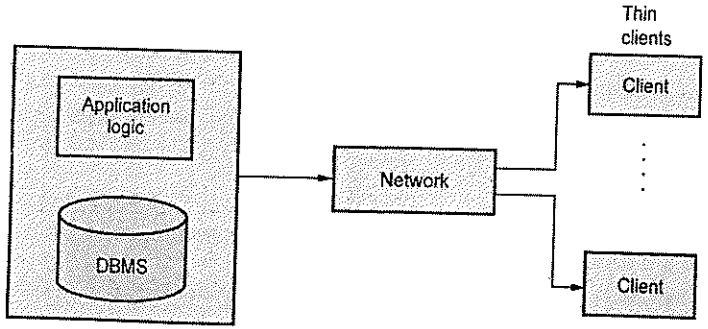


Fig. 4.10.2 Two server architecture

- In traditional client server architecture, the client computer implements simply the graphical user interface part. The Server computer implements application logic and data management part. In such architecture, the client is called as **thin client.**
- On the other hand there is some client server architecture, in which client implements graphical interface as well as business (application) logic part. Only data management part is taken care by server. Such client is called as **thick client.**
- The thick client model has several disadvantages as compared to thin client. Those are -
  - o There is no central place to update and maintain the business logic as application program runs on the client computer only.
  - o Client should run the business logic with reliability without affecting any security aspects.
  - o The thick client architecture can handle only limited number of users.

Three Tier Architecture

The three tier architecture is made up of three tiers as –

(1) Presentation Tier : The presentation tier is comprised of graphical user interface. The user always expects the GUI which is easy to input. He/She expects the results in some organized format. The use web-based interfaces are getting popular.

(2) Middle Tier : This is a tier in which the application or business logic executes. The complex business processes get executed at this tier. The business logic can be implemented in some suitable programming language like C++ or Java.

(3) Data Management Tier : This tier takes care of data management activities. The database management (DBMS) systems are located in this architecture.

The typical three tier architecture is represented by Fig. 4.10.3.

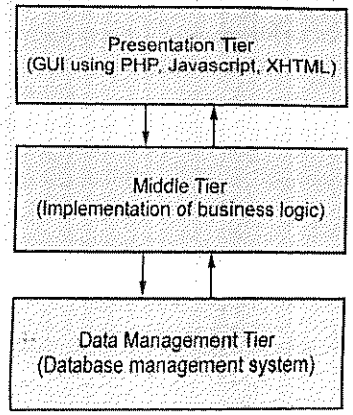


Fig. 4.10.3 Three tier architecture

Advantages of Three Tier Architecture

The Three tier architecture has following advantages –

(1) **Heterogeneous Systems :** Different platforms and different software components can be used at different tiers. Also it is easy to replace or modify some code present at any tier without affecting the other code.

(2) **Thin Clients :** Clients need enough computation power for presentation layer.

(3) **Integrated Data Access :** In many applications, data must be accessed from several sources. This is can be done transparently in three tier architecture by using the middle tier.

(4) **Scalability to Many Clients :** Multiple clients can access the system through middle tier.

(5) **Software Development Benefits :** By separating presentation, business logic and data management activities by means of separate tiers, it is easy to debug, maintain and modify the system as per the requirements. Interaction between tiers occur through well defined standardized APIs (Application Programming interfaces). Hence it is possible to create reusable components using three tier architecture.

University Questions

1. *Draw and explain 3-tier Architecture and technology relevant to each tier. Write advantages of 3-tier architecture.* **VTU : July-18, Marks 4**
2. *Explain the Single-tier and Client Server architecture, with a neat diagram.* **VTU : Jan.-19, July-19, Marks 8**

4.11 The Presentation Layer

The presentation layer is used by the client to issue the request and get the display response that the middle tier generates.

The graphical user interface is an important element of presentation layer. It can be created using various technologies such as HTML, JavaScript, Stylesheets and so on.

Let us discuss these technologies briefly –

4.11.1 HTML Forms

- HTML stands for Hyper Text Markup Language. This markup language is used to create web pages. The web pages may consist of HTML forms through which user can input data.
- Thus HTML forms are common way of communicating data from **client** tier to middle tier.
- **Syntax of HTML Form**

```
<FORM ACTION ="action_page" METHOD ="method name" NAME = "name of the form">
...
</FORM>
```

Where

Action : It specifies the name of the file to which the form contents must be submitted. This file can be JSP page or a servlet or any other type of file.

Method : There are two commonly used methods - GET and POST method.

Name : The name specifies the name of the form.

- There are various components of the form such as textbox, textarea, checkbox, radio buttons and so on.
- **Example of HTML Form**
- **Text** is typically required to place one line text. For example if you want to enter some name then it is always preferred to have Text field on the form.
- The text field can be set using

```
<input type="text" size="30" name ="username" value=" ">
```

The input type is text and the value of this text field is " " That means the blank text field is displayed initially and we can enter the text of our choice into it. There is size parameter which allows us to enter some size of the text field.

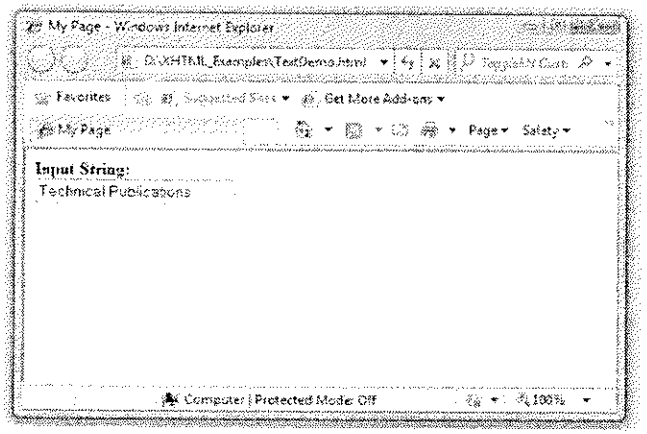
- Some other parameters or attributes can be
  - **maxlength** that allows us to enter the text of some maximum length.
  - **name** indicates name of the text field.
  - **align** denotes the alignment of the text in the text field. The alignment can be left, right, bottom and top.

Example Code

HTML Document [TextDemo.html]

```
<!DOCTYPE html>
<html>
 <head>
      <title>My Page</title>
 </head>
 <body>
 <form>
      <b>Input String:</b><br/><input type="text" size="25" value="">
 </form>
 </body>
</html>
```

Output



4.11.2 JavaScript

- JavaScript is a scripting language at the client tier with which we can add programs to webpages that run directly at the client.
- Following are some typical uses of JavaScript -
  - **Browser Support :** For running the JavaScript in the browser there is no need to use some plug-in. Almost all the popular browsers support Java Scripting.
  - **Form Validation :** JavaScript is useful web technology for Form validation. In this technique each field of the form is validated and by this way the user input can be validated.

  - **Browser Control :** This feature allows to open the web page in customized windows. For instance we can prevent the pop-ups to get displayed on the web page using JavaScript.

- **The JavaScript can be directly embedded within the HTML document or it can be stored as external file.**

Directly embedded JavaScript

The syntax of directly embedding the JavaScript in the HTML is

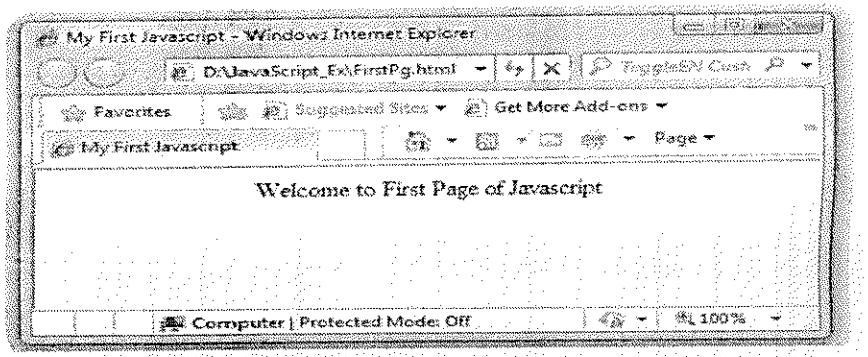
```
<script type="text/javascript">
                ...
                ...
                ...
</script>
```

Example 4.11.1 *Write a JavaScript to display Welcome message in JavaScript.*

Solution :

```
<!DOCTYPE html >
<html >
      <head>
                <title> My First Javascript </title>
      </head>
      <body>
                <center>
                <script type="text/javascript">
                /*This is the First JavaScript*/
                document.write(" Welcome to First Page of Javascript");
                </script>
                </center>
      </body>
</html>
```

Output



Script Explanation :

In above script

(1) we have embedded the javaScript within

```
<script type="text/javascript">
...
</script>
```

(2) a comment statement using /* and */. Note that this type of comment will be recognized only within the <script> tag. Because, JavaScript supports this kind of comment statement and not the XHTML document.

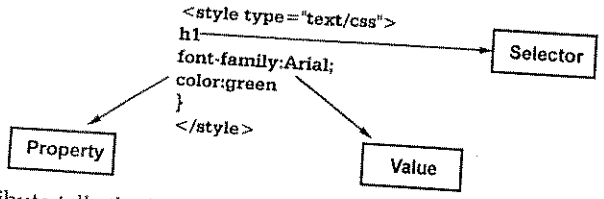
(3) Then we have written document.write statement, using which we can display the desired message on the web browser.

4.11.3 Stylesheet

- The Cascading Style Sheet (CSS) is a markup language used in the web document for presentation purpose.
- The **primary intension of CSS** was to separate out the web content from the web presentation.
- Various elements such as text, font and color are used in CSS for presentation purpose. Thus CSS specification can be applied to bring the styles in the web document.
- The style specification is specified differently for each different level. For instance :
- For inline cascading style sheets the **style** appears inside the tag for defining the value.

```
<p style="font-size: 30pt ;font-family:Script">
```

- For the document cascading style sheet the **style** specification appear as the content of a style element within the header of a document.



The type attribute tells the browser that what it is reading is text which is affected by ıe Cascading Style Sheet. The type specification is necessary because there is one more ɔecification used in JavaScript.

ɜatures of CSS

- By combining CSS with the html document, considerable amount of **flexibility** into the content submission can be achieved.

- Similarly, separating out the style from actual contents help in **managing** large-scale complex sites. Thus CSS facilitates publication of content in multiple presentation formats.
- If CSS is used, effectively then **global style sheet** can be applied to a web document. This helps in maintaining the **consistency in the web document**.
- If a **small change** needs to be done in the style of web content, then CSS makes it more convenient.

4.12 The Middle Tier

VTU : July-18, Marks 4

The middle tier usually contains the business logic. It is implemented in the programming languages like C, C++, Java, Perl and so on.

Various technologies associated with the middle tier **are**

4.12.1 CGI : The Common Gateway Interface

The main purpose of CGI is to add up the functionality to a web server.

The CGI is language-independent interface that allows a server to start an external process. These external processes further get information about the **request** using the command line arguments and its standard input stream. The **response** can then be given using standard output stream. Each request is answered in ***a separate process*** by ***a separate instance*** of the CGI program. The CGI programs can be written in Perl or C language.

How to write CGI scripts ?

- Before writing CGI scripts it is necessary to install web server on which the scripts can be executed. Apache is a open source web server and can be easily downloaded without any cost.
- The CGI script can be written in C or in PERL. It should be executed in cgi-bin directory of your web server.
- The extension to these CGI script files is **.cgi** or **.pm**

Here is a simple example of CGI program.

```
int main(void)
{
printf("Content-type: text/html\n\n");
printf("<H1>It's my First CGI Program!\n</H1>");
return 0;
}
```

The output on the web browser will be -

```
It's my first CGI program!
```

Working

When a browser of client sends a request for the data on the server then the server specially the web server locates the CGI program and passes this request to the CGI program. The CGI program decodes the information sent by the client and performs the necessary computations. The output of the CGI program is then sent back to the client's browser. This scenario is described by the following figure -

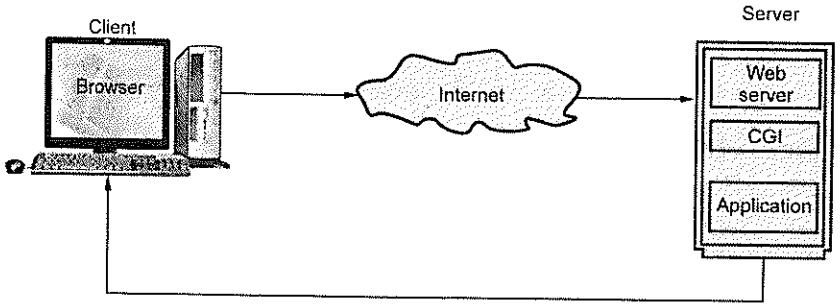


Fig. 4.12.1 Client-server communication with CGI

With these server side technologies it becomes easier to maintain the larger scale web applications with dynamic documents.

Disadvantages of CGI Script

(1) CGI programs are memory intensive. Each time the program needs to be loaded in the memory.

(2) CGI programs are complex to write.

(3) If proper care is not taken, then the CGI programs can compromise server security.

(4) It takes lot of processing time.

(5) It uses platform dependent language such as C, C++

4.12.2 Application Servers

- Application server is a server designed to run the applications.
- Each page request generated by user creates a new process. If this request is often gets fulfilled by the web server itself, then it causes the performance degrade. Hence to scale up the large number of requests simultaneously an application server is used.

- It is located in-between the primary web server and the back-end database.
- An application server maintains a pool of threads or processes and uses these to execute requests. Thus, it avoids the startup cost of creating a new process for each request.

General Architecture

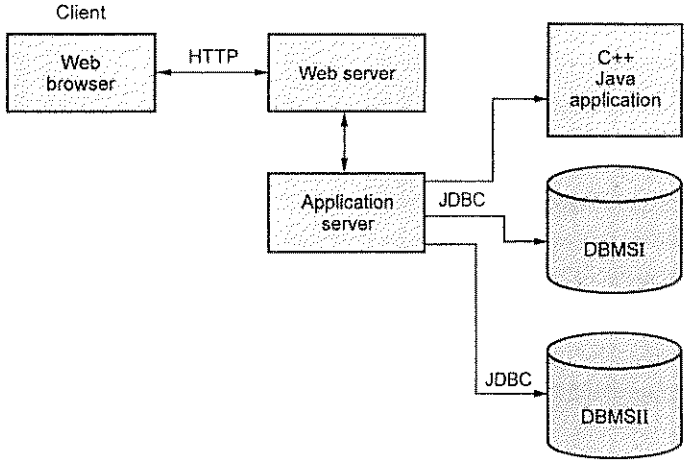


Fig. 4.12.2 Role of application server

- Client makes a request to the web server using HTTP protocol through a web browser (such as Google Chrome, Firefox).
- If the web page demanded by the client is a static web page such as simple HTML page, then web server immediately fulfills this request by sending the required web page to the client.
- But if the client demands the dynamic web page, then the web server sends this request to the application server.
- The application server contacts one or more data sources to acquire the desired data.
- The application server assembles all the necessary data and creates the dynamic web page as a response and sends it back it to the web server.
- Ultimately web server sends the response to the client.

Advantages of Application Server

(1) It provides the flexibility to middle tier architecture to accommodate any number of requests from user.

(2) It increases performance of execution of requests.

(3) It eliminates the process creation overhead of the middle tier.

(4) It provides the session management services. It can detect when the session starts and when it ends and can keep track of the sessions of individual users.

(5) Application servers also help to ensure secure database access by supporting general id user mechanism.

4.12.3 Servlets

- Servlets are basically the Java programs that run on server. These are the programs that are requested by the XHTML documents and are displayed in the browser window as a response to the request.
- The servlet class is instantiated when web server begins the execution.
- The execution of servlet is managed by servlet container, such as Tomcat.
- The servlet container is used in java for dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.
- Servlets are most commonly used with HTTP (i.e. Hyper Text Transfer Protocol) hence sometimes servlets are also called as "HTTP Servlet".
- The main purpose of servlets is to add up the functionality to a web server.

Working of How Servlet Works ?

- Before learning the actual servlet programming it is very important to understand how servlet works.

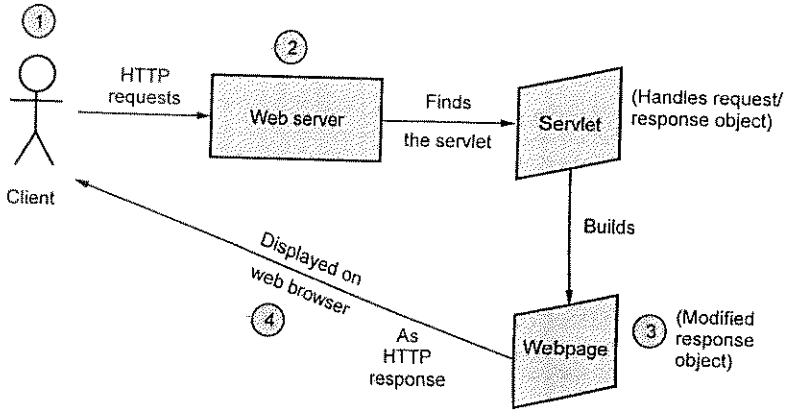


Fig. 4.12.3 How servlet works ?

1. When a client makes a request for some servlet, he/she actually uses the **Web browser** in which request is written as a URL.
2. The web browser then sends this request to **Web server**. The web server first finds the requested servlet.
3. The obtained servlet gathers the relevant information in order to satisfy the client's request and builds a **web page** accordingly.
4. This web page is then **displayed** to the client. Thus the request made by the client gets satisfied by the servlets.

Advantages

- The servlets are very **efficient** in their performance and get executed in the address space of the belonging web server.
- The servlets are **platform independent** and can be executed on different web servers.
- The servlets working is based on **Request-Response**. Any HTML form can take the user input and can forward this input to the servlet. The servlets are then responsible to communicate with the back-end database and manipulate the required business logic. These servlets embedded on the web servers using **Servlets API**.
- Servlets provide a way to generate the **dynamic document**. For instance : A servlet can display the information of current user logged in, his logging time, his last access, total number of access he made so far and so on.
- **Multiple users** can keep a co-ordination for some application among themselves using servlets.
- Using servlets **multiple requests** can be synchronized and then can be concurrently handled.

4.12.4 Java Server Pages

JSP stands for **Java Server Pages**.

It is an alternative way than the servlet to build the dynamic web pages. It is built on top of the Servlet.

Following are some advantages of Java Server pages -

1. JSP is useful for **server side programming**.
2. JSP can be used **along with servlets**. Hence **business logic** for any application can be developed using JSP.

3. **Dynamic contents** can be handled using JSP because JSP allows scripting and element based programming.
4. JSP allows creating and using our own **custom tag libraries**. Hence any application specific requirements can be satisfied using custom tag libraries. This helps the developer to develop any kind of application.
5. JSP is a **specification and not a product**. Hence developers can develop **variety of applications** and add up to performance and quality of software products. Due to this many companies are ready to invest in JSP technology.
6. JSP is an essential component of J2EE. Hence using JSP it is possible to develop simple as well as complex applications.

Difference between Servlet and JSP

Sr. No.	JSP	Servlet
1.	JSP is a scripting language that generates dynamic content.	Servlets are the Java programs that can be compiled to generate dynamic content.
2.	JSP runs slower than servlet.	Servlet runs faster than JSP.
3.	In Model View Controller JSP acts as view.	In Model View Controller servlet acts as controller.
4.	JSP can build custom tags.	There is no facility of creating custom tags.
5.	JSP can directly call Java beans.	Servlets have no facility of calling Java bean.
6.	It is easier to code in JSP.	The servlets are basically complex Java programs.

4.12.5 Maintaining State

- When you open some application, use it for some time and then close it. This entire scenario is named as session.
- **Session state** is a server-based state mechanism that lets web applications store and retrieve objects of any type for each **unique user session.** That is, each browser session has its own session state stored as a serialized file on the server, which is deserialized and loaded into memory as needed for each request. Refer Fig. 4.12.4.
- Because server storage is a finite resource, objects loaded into memory are released when the request completes, making room for other requests and their session objects. This means there can be more active sessions on disk than in memory at any one time.

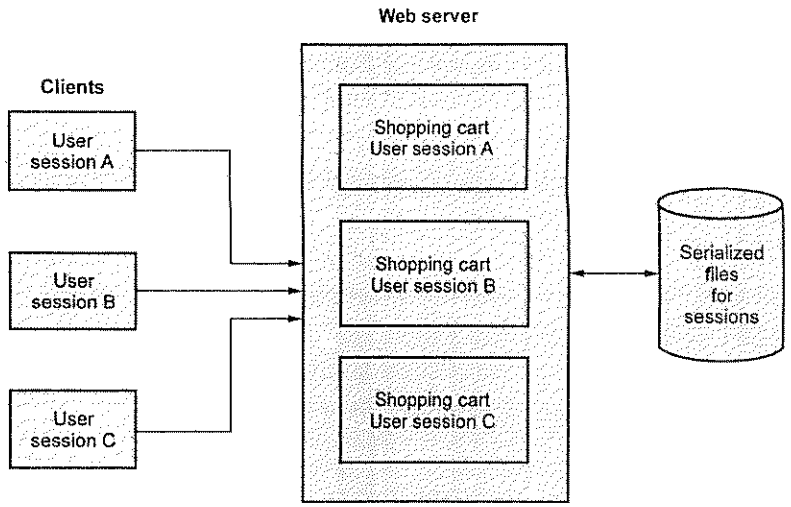


Fig. 4.12.4 Session state

- Sometimes the information about the session is required by the server. This information can be collected during the session. This process is called **session tracking**.
- For sending all state information to and fro between browser and server, usually an ID is used. This ID is basically a **session-ID**.
- Thus session-ID is passed between the browser and server while processing the information. This method of keeping track of all the information between server and browser using session-ID is called **session tracking**.
- In servlets, for creating the sessions **getSession()** method can be used. This method returns the object which stores the bindings with the names that use this object. And these bindings can be managed using **getAttribute()**, **setAttribute()**, **removeAttribute()** methods. Actually in session tracking two things are playing an important role -
  1) One is **HttpServletRequest** interface which supports **getSession()** method.
  2) The another class is **HttpSession** class which supports the binding managing functions such as **getAttribute()**, **setAttribute()**, **removeAttribute()** and **getAttributeNames()**.

4.12.6 Cookies

Definition : Cookies are some little information that can be left on your computer by the other computer when we access an internet.

- Generally this information is left on your computer by some advertising agencies on the internet. Using the information stored in the cookies these advertising agencies can keep track of your internet usage, liking of users.
- For the applications like on-line purchase systems once you enter your personal information such as your name or your e-mail ID then it can be remembered by these systems with the help of cookies.
- Sometimes cookies are very much dangerous because by using information from your local disk some malicious data may be passed to you. So it is upto you how to maintain your own privacy and security.
- The **cookie** class is used to create cookies in servlet.
- **Syntax :** The Syntax for constructor for cookies are
  - Cookie()
  - Cookie(String name, String value)
- Various methods used in Cookie are described in following table

Sr. No.	Method	Purpose
1.	public string getName()	It returns the name of the cookie.
2.	public String getValue()	It returns the value of the cookie.
3.	public string setName()	It sets or changes the name of the cookie.
4.	public String setValue()	It sets or changes the value of the cookie.
5.	public void addCookie(Cookie c)	The cookie is added in the response object of HttpServletResponse interface.
6.	public Cookie[] getCookies()	All the cookies can be returned using this method with the help of HttpServletRequest interface.

- In Servlet following steps are used to support for cookies.

tep 1 : Creation of Cookies :

The cookie can be created in Servlet and added to response object using **addCookie** ıethod

```
ɔokie cookie=new Cookie("user","XYZ");//creating cookie object
sponse.addCookie(cookie);//adding cookie in the response
```


Step 2 : Reading Cookies :

The value from the cookie can be obtained using the **getName()** and **getValue()** methods.

```
Cookie[ ] my_cookies=req.getCookies();
int n=my_cookies.length;
for(int i=0;i<n;i++)
{
String name=my_cookies[i].getName();
String value=my_cookies[i].getValue();
}
```

University Question

1. *What is CGI ? Why was CGI introduced ? What are the disadvantages of an architecture using CGI script ?* **VTU : July-18, Marks 4**

❑❑❑

Notes

MODULE - 4

5 Normalization

Syllabus

Normalization : Database Design Theory : *Introduction to Normalization using Functional and Multivalued Dependencies : Informal Design Guidelines for Relation Schema, Functional Dependencies, Normal Forms, based on Primary Keys, Second and Third Normal Forms, Boyce/Codd Normal Form, Multivalued Dependencies and Fourth Normal Form, Join dependencies and Fifth Normal Form,* ***Normalization Algorithms :*** *Inference Rules, Equivalence, and Minimal Cover, Properties of Relational Decompositions, Algorithms for Relational Database Schema Design, Nulls, Dangling Tuples and Alternate Relational Designs, Further discussion of Multivalued dependencies and 4NF, Other Dependencies and Normal Forms.*

Contents

Part I : Database Design Theory and Introduction to Normalization

5.1 Informal Design Guidelines for Relation Schema

VTU : July-18, 19, Jan.-19, 20, Marks 4

There are four informal measures for relational schema and those are -

(1) Semantics of the attributes

(2) Reducing the redundant value in tuple

(3) Reducing NULL values in tuple

(4) Disallowing generation of spurious tuples

Let us discuss them in detail -

5.1.1 Semantics of the Attributes

The term semantics of the attributes refers to meaning of the attributes. It tells how to interpret the values stored.

If the semantic is easier then the relational schema would be better.

Guideline 1 : "Design the relation schema so that it is easy to explain the meaning. Do not combine attributes from multiple entry types into single relation."

Note that the relational schema should always correspond to one entity type or one relationship type for clear meaning. For example -

Following is a bad design as it combines two entities in one schema

EmpNo	EmpName	DeptNo	DeptName	ProjectName

5.1.2 Reducing Redundant Values in Tuple

Definition of redundancy : Redundancy is a condition created in database in which same piece of data is held at two different places.

Redundancy is at the root of several problems associated with relational schemas.

Problems caused by redundancy : Following problems can be caused by redundancy -

i) **Redundant storage :** Some information is stored repeatedly.

ii) **Update anomalies :** If one copy of such repeated data is updated then inconsistency is created unless all other copies are similarly updated.

iii) **Insertion anomalies :** Due to insertion of new record repeated information get added to the relation schema.

iv) **Deletion anomalies :** Due to deletion of particular record some other important information associated with the deleted record get deleted and thus we may lose some other important information from the schema.

Example : Following example illustrates the above discussed anomalies or redundancy problems consider following schema in which all possible information about employee is stored.

EmpID	EName	Salary	DeptID	DeptName	DeptLoc
1	AAA	10000	101	XYZ	Pune
2	BBB	20000	101	XYZ	Pune
3	CCC	30000	101	XYZ	Pune
4	DDD	40000	102	PQR	Mumbai

Redundancy!!!

1) **Redundant storage :** Note that the information about **DeptID, DeptName** and **DeptLoc** is repeated.

2) **Update anomalies :** In above table if we change **DeptLoc** of Pune to Chennai, then it will result **inconsistency** as for DeptID 101 the DeptLoc is Pune. Or otherwise, we need to **update multiple copies of DeptLoc** from Pune to Chennai. Hence this is an update anomaly.

3) **Insertion anomalies :** For above table if we want to add new tuple say (5, EEE, 50000) for **DeptID 101** then it will cause repeated information of (101, XYZ, Pune) will occur.

4) **Deletion anomalies :** For above table, if we delete a record for **EmpID 4**, then automatically information about the **DeptID 102, DeptName PQR** and **DeptLoc Mumbai** will get deleted and one may not be aware about **DeptID 102**. This causes deletion anomaly.

Guideline 2 :

- Design relation schemas so that no insertion, deletion or modification anomalies are present in the relations.
- If any anomalies are present, note them clearly and make sure that the application programs that update the database will operate correctly.

5.1.3 Reducing NULL Values in Tuple

Following are the reasons why NULL values appear -

1. Attribute does not apply to the tuple.
2. The attribute value for this tuple is unknown.
3. The value is known but not recorded.

Guideline 3 :

- As far as possible, avoid placing attributes in the base relation whose values frequently be null.
- If nulls are unavoidable they should be applied only to exceptional cases and not to majority of tuples.

5.1.4 Disallowing Generation of Spurious Tuples

Splitting of tables based on non primary key values results in spurious tuples or incorrect information.

Bad designs for a relational database may result in erroneous results for certain JOIN operations.

Guideline 4 :

- Design relation schemas so that they can be joined with equality conditions on primary key/foreign keys, which guarantees no spurious tuples to be generated.
- Do not have relations that contain matching attributes other than PK/FK combinations.

Example of spurious tuple

Consider poorly designed relation Student_Info

RollNo	Subject_Code	Name	SubjectName	Location
1	S101	AAA	Maths	Pune
1	S102	AAA	English	Mumbai
2	S101	BBB	Maths	Pune
2	S102	BBB	English	Mumbai
3	S101	CCC	Maths	Pune

The poorly designed table will split on the basis of **Location** which is a non primary key attribute. The resultant relation would be -

Student_Loc

Name	Location
AAA	Pune
AAA	Mumbai
BBB	Pune
BBB	Mumbai
CCC	Pune

Student_Subject

RollNo	Subject_Code	SubjectName	Location
1	S101	Maths	Pune
1	S102	English	Mumbai
2	S101	Maths	Pune
2	S102	English	Mumbai
3	S101	Maths	Pune

Now if Student_Loc and Student_Subject tables are joined on the basis of **Location** then we will not get the original table.

University Questions

1. *Discuss insertion, deletion and modification anomalies. Why are they considered bad ? Illustrate with examples.* **VTU : July-18, 19, Marks 4**
2. *Explain any two informal quality measures employed for a relation schema design.* **VTU : Jan.-19, Marks 4**
3. *What are the problems caused by insertion, updation and deletion anomalies ? Discuss with an example.* **VTU : Jan.-20, Marks 6**

5.2 Functional Dependencies

VTU : Jan.-19, 20, Marks 6

Definition : A functional dependency A->B in a relation holds if two tuples having same value of attribute A also have the same value for attribute B. It is denoted by A->B where A is called determinant and B is called dependent.

For example - Consider Student table as follows -

Roll	Name	City
1	AAA	Mumbai
2	BBB	Pune
3	CCC	Gandhinagar

Here

Roll ->Name **hold**

But

Name->City **does not hold**

- In above table, student roll number is unique hence each student's name and city can be uniquely identified using his roll number.

- But using name we cannot uniquely identify his/her city because there can be same names of the students. Similarly using city name we can not identify the student uniquely. As in the same city may belong to multiple students.

Another example -

Consider a relation in which the roll of the student and his/her name is stored as ollows :

R	N
1	AAA
2	BBB
3	CCC
4	DDD
5	EEE

Fig. 5.2.1 : Table which holds functional dependency i.e. R->B

Here, R->N is true. That means the functional dependency holds true here. Because for very assigned RollNuumber of student there will be unique name. For instance : The ame of the Student whose RollNo is 1 is AAA. But if we get two different names for the ame roll number then that means the table does not hold the functional dependency. ollowing is such table -

R	N
1	AAA
2	BBB
3	CCC
1	XXX
2	YYY

Fig. 5.2.2 : Table which does not hold functional dependency

In above table for RollNumber 1 we are getting two different names - "AAA" and XXX". Hence here it does not hold the functional dependency.

Trivial FD : The functional dependency A->B is trivial if B is a subset of A.

For example {A,B}->A

Non Trivial FD : The functional dependency A->B is non trivial if B is not a subset of A.

For example {A,B}->C

Example 5.2.1 *For the given below relation R(A,B,C,D,E) and its instance, check whether FDs given hold or not. Give reasons.*

i) A->B ii) B->C iii) D->E iv) CD->E

A	B	C	D	E
a1	b1	c1	d1	e1
a1	b2	c1	d1	e1
a2	b2	c1	d2	e3
a2	b3	c3	d2	e2

VTU : Jan.-20, Marks 4

Solution : Association among attributes is known as Functional Dependencies (FD). AFD X->Y require that the value of X uniquely determines the value of Y where X and Y are set of attributes.

For example,

Roll_No -> Name : the value of Roll_No uniquely determines the Name.

Now from, the given relation and its instance -

i) The FD A->B does not hold because – a1 has two different values b1 and b2. Similarly a2 has two different values and those are b2 and b3.

ii) The FD B->C holds true.

iii) D->E does not hold true because d2 gives two different values e3 and e2.

iv) CD->E hold true as (c1,d1) gives e1 , (c1,d2) gives e3 and (c3,d2) gives e2. All are uniquely identified.

5.2.1 Inference Rules

The closure set is a set of all functional dependencies implied by a given set F. It is denoted by F^+

The closure set of functional dependency can be computed using basic three rules which are also called as **Armstrong's Axioms.**

These are as follows -

i) **Reflexivity :** If $X \supseteq Y$, then $X \rightarrow Y$

ii) **Augmentation :** If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z

iii) **Transitivity :** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

In addition to above axioms some additional rules for computing closure set of functional dependency are as follows -

- **Union :** If $X \rightarrow Y$ and $X \rightarrow Z$ then $X \rightarrow YZ$
- **Decomposition :** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

Let us understand how to apply Armstrong's axioms for finding the closure of set of functional dependencies -

Example 5.2.2 *Given FD's for relation R{A,B,C,D,E,F}, Find closure of FD set by applying Armstrong's Axioms.*
A->B, A->C, CD->E, CD->F, B->E

Solution :

Step 1 : A -> gives A attribute itself by reflexivity. It is called trivial production.

A -> B and A -> C, Hence by union rule A -> BC

A -> B and B -> E, Hence by transitivity rule A -> E

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

Step 2 : B -> gives B itself

And B -> E

$\therefore (B)^+ = \{B,E\}$

Step 3 : CD -> gives CD itself. It is trivial.

CD -> E, CD -> F, Hence by union rule CD -> EF

$\therefore (CD)^+ = \{C,D,E,F\}$

So by omitting trivial productions, we get

$\therefore F^+ = \{A \rightarrow BC, A \rightarrow E, B \rightarrow E, CD \rightarrow EF\}$

Example 5.2.3 *Compute the closure of the following set F of functional dependencies for relational schema R = (A,B,C,D,E) A -> BC, CD -> E, B -> D, E -> A*

Solution : The closure of F is denoted by F^+ and it can be computed in following steps

Step 1 : As A -> BC is given we get

A -> B and A -> C By decomposition rule

Step 2 : As

A -> B (Refer step 1)

B -> D (given)

A -> D (transitivity rule)

Step 3 :

A -> CD because A -> C and A -> D (From step 1 and Step 2 applying union rule)

Step 4 :

CD -> E (given)

A -> E (transitive rule as A->CD, CD->E hence A->E)

Step 5 :

Since $A \rightarrow A$, we have (reflexive)

$\mathbf{A \rightarrow ABCDE}$ from the above steps (union)

Step 6 :

Since $E \rightarrow A$, $\mathbf{E \rightarrow ABCDE}$ (transitive)

Step 7 :

Since $CD \rightarrow E$, $\mathbf{CD \rightarrow ABCDE}$ (transitive)

Step 8 :

Since $B \rightarrow D$ and $BC \rightarrow CD$, $\mathbf{BC \rightarrow ABCDE}$ (augmentative, transitive)

Step 9 :

Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$

Thus any functional dependency with A, E, BC, or CD on the left hand side of the arrow is in F^+

Example 5.2.4 *Give Armstrong's axioms and using it find the closure of following FD set.*
A->B, AB->C, D->AC, D->E

Solution :

Step 1 : Consider the rules D -> AC and D -> E

$\therefore$ **D -> ACE** **(Union rule)**

Step 2 : Consider AB -> C then we get

∴ A -> C and B -> C **(decomposition rule)**

Thus F^+ ={A -> C, B -> C, D -> ACE}

Example 5.2.5 *R = {A,B,C,D,E,F} and FDs are A -> BC E -> CF B -> E CD -> EF compute closure of $\{A,B\}^+$*

Solution :

Step 1 : A -> BC

∴ A -> B and A -> C (decomposition)

So we add A, B, C in closure set

Step 2 : E -> CF

∴ E -> C and E -> F (decomposition)

Step 3 : B -> E ∴ B -> C, F

So we add E and F in closure set

Hence

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

Example 5.2.6 *Consider schema EMPLOYEE(E-ID, E-NAME, E-CITY, E-STATE) and FD = {E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, E-CITY-> E-STATE}*

(1) Find attribute of closure for$(E\text{-}ID)^+$

(2) Find$(E\text{-}NAME)^+$

olution :

1) Finding $(E\text{-}ID)^+$ means finding the closure. In this process we try to find out, all the attributes that can be derived from E-ID.

   As E-ID->E-NAME, E-ID->E-CITY, E-ID->E-STATE, We add E-NAME, E-ID, E-CITY, E-STATE to $(E\text{-}ID)^+$

   As E-ID->E-CITY, E-CITY->E-STATE, we add E-STATE to $(E\text{-}ID)^+$

   ∴ $(E\text{-}ID)^+$ = {E-ID, E-NAME, E-CITY, E-STATE)

2) E-NAME derives no rule. Hence (E-NAME)+ = {E-NAME}

2.2 Keys and Functional Dependencies

- For a given relation R = {A1, A2, A3, ..., An} K is a key of R then if closure $(K)^+$ = {A1, A2, ..., An} and no subset of K i.e. X such that $(X)^+$ = A1, A2, ..., An}

- In other words there are two conditions -
  1) The $(K)^+$ contains all the attributes of relations R -
  2) All subset X of K, $(X)^+$ never contains all the attributes of R i.e. $(X)^+ \neq$ {A1, A2, ..., An}
- If only one subset of R satisfy above condition then it is known as **primary key.**
- If more than one subset of R satisfies above condition then all these subsets are recognized as **candidate keys.** In that case one of the candidate key is also considered as **primary key.**
- A superset of candidate key K is known as **superkey.**

Example 5.2.7 *Give R = {A,B,C,G,H,I}. The following set F of functional dependencies holds A -> B, A -> C, CG -> H, CG -> I , B -> H*

Computer AG^+ . Is AG candidate key ?

Solution :

Step 1 : A -> B, A -> C, Hence add A, B, C to the set of AG^+.

Step 2 : A -> B B -> H, hence add H to the set of AG^+

Step 3 : CG -> I, hence add I to the set of AG^+. Also add G to the set.

Thus $(AG)^+$ = {A,B,C,G,H,I} = Relation R

Hence $(AG)^+$ is a candidate key.

Example 5.2.8 *Compute the closure of R(A,B,C,D,E) with the following set of functional dependencies A -> BC, CD -> E, B -> D, E -> A*

List the candidate keys of R

Solution :

Step 1 : A -> BC hence add A,B,C to $(A)^+$

A -> BC can be decomposed into A -> B and A -> C. Also B -> D. Thus A -> D is also true by transitivity rule.

Hence add D to $(A)^+$

A -> C, A -> D ∴ By union rule A -> CD.

As CD -> E add E to $(A)^+$

∴ $(A)^+$ = {A,B,C,D,E}

Step 2 : Consider $(B)^+ = \{B,D\} \neq R$ hence it is not a candidate key

Step 3 : Consider $(BC)^+ = \{B,C,D,E,A\} = \{A,B,C,D,E\} = R$. Hence it is a candidate key

Step 4 : Consider CD -> E, E -> A, hence $(CD)^+ = \{A,B,C,D,E\}$.
Hence it is a candidate key

Step 5 : Consider E -> A, A -> BC, B -> D, CD -> E.
Hence $(E)^+ = \{A,B,C,D,E\}$ is a candidate key.

Thus we get the candidate keys as {A,BC, CD, E}

Example 5.2.9 *Consider schema R = (A,B,C,G,H,I) and the set F of functional dependencies {A -> B, A -> C, CG -> H, CG -> I, B -> H}. Use $(F)^+$ Prove $(AG)^+$ -> I*

Solution :

Step 1 : As A -> B
B -> H
∴ **A -> H** **(transitivity rule)**

Step 2 : CG -> H
CG -> I
∴ **CG -> HI** **(Union rule)**

Step 3 : A -> C
CG -> I
AG -> I (Pseudo transitive rule)

Thus AG -> I is proved

$(F)^+ = \{A \to B, A \to C, CG \to H, CG \to I, B \to H, A \to H, CG \to HI, AG \to I\}$

University Question

1. *What do you mean by closure of attribute ? Write an algorithm to find closure of attribute.* **VTU : Jan.-19, Marks 6**

5.3 Equivalence and Minimal Cover

VTU : July-18, 19, Jan.-19, 20, Marks 8

- A set of functional dependencies E is said to covered by F if every FD in E is also in F^+, i.e. every dependency in F can be inferred from E and vice versa.
- Two sets of FDs E and F are equivalent if $E^+ = F^+$.

- If E and F are equivalent only if both $E^+ = F^+$
  1) E covers F
  2) F covers E holds

Example 5.3.1 *Here are two sets of FDs for R(A,B,C,D,E). Are they equivalent ?*

1) A -> B
AB -> C
D -> AC
D -> E

2) A -> BC
D -> AE

Solution : Two sets of FDs are said to be equivalent if

Rule 1 : If FD2 ⊃ FD1. That means all FDs of FD1 can be derived from all the FDs of FD2.

Rule 2 : If FD1 ⊃ FD2. That means all FDs of FD2 can be derived from all the FDs of FD1.

Rule 3 : If both rule 1 and rule 2 are true then FD1 = FD2.

For given two sets

Step 1 : We will first check if all the FDs of FD1 are present in FD2

A -> B is present in FD1.	A -> BC is in FD2, that also means A -> B and A -> C by decomposition rule
Similarly AB -> C is in FD1 ∴ $(A)^+ = \{A,B,C\}$	Hence $(A)^+ = \{A,B,C\}$
∴ $(AB)^+ = \{A,B,C\}$	As D -> AE then by decomposition rule,
D -> AC	D -> A, D -> E
i.e D -> A, D -> C by decomposition rule.	As A -> B, then by transitivity rule D -> A, A -> B, D -> B
The D -> A, A -> B and C	∴ $(D)^+ = \{A,B,C,D,E\}$
D -> A, D -> B, D -> C by transitivity rule D -> E is given ∴ $(D)^+ = \{A,B,C,D,E\}$	

Step 2 : We will first check if all the FDs of FD2 are present in FD1

$\therefore (A)^+ = \{A,B,C\}$	$(A)^+ = \{A,B,C\}$
$\therefore (AC)^+ = \{A,B,C\}$	$\therefore (D)^+ = \{A,B,C,D,E\}$
$\therefore (D)^+ = \{A,B,C,D,E\}$	

Thus from Step 1 and Step 2, FD2 ⊃ FD1 and FD1 ⊃ FD2. Hence both the sets are equivalent.

Example 5.3.2 *Consider two sets of functional dependency.*
F = {A -> C, AC -> D, E -> AD, E -> H} and G = {A -> CD, E -> AH}. Are they equivalent ?
VTU : July-18, 19, Marks 8

Solution : We will find the FDs of both F and G and then check for F and G ⊃ FD2 and G ⊃ F.

Step 1 : For F

Using G functional dependencies -

$(A)^+ = \{A,C,D\}$ ← As A -> CD is in G

$(AC)^+ = \{A,C,D\}$ ← As A -> CD is in G

$(E)^+ = \{A,C,D,E,H\}$ ← As E -> AH, A -> CD is in G

Using F functional dependencies -

$(A)^+ = \{A,C,D\}$ ← As A -> C and AC -> D is in F

$(AC)^+ = \{A,C,D\}$ ← AC -> D is in F

$(E)^+ = \{A,C,D,E,H\}$ ← As E -> AD, E -> H and A -> C is in F

Step 2 : For G

Using F functional dependencies -

$(A)^+ = \{A,C,D\}$ ← As A -> C and AC -> D is in F

$(E)^+ = \{A,C,D,E,H\}$ ← As E -> AD, E -> H is in F

Using G functional dependencies -

$(A)^+ = \{A,C,D\}$ ← As A -> CD is in G

$(E)^+ = \{A,C,D,E,H\}$ ← As E -> AH and A -> CD is in G

Step 3 : From both these steps

G⊇F and F⊇G

Hence F and G are equivalent.

Example 5.3.3 *Given below are two sets of FDs for a relation R(A,B,C,D,E), Are they equivalent ?*
i) A->B, AB->C, D->AC, D->E ii) A->BC, D->AE
VTU : Jan.-19, Marks 6

Solution : We will assume these relations as F and G. That means -

F : A -> B, AB -> C, D -> AC, D -> E

G : A -> BC, D -> AE

Now we will find the F^+ and G^+ as follows

Step 1 : For F

Using G functional dependencies -

$(A)^+ = \{A,B,C\}$

$(AB)^+ = \{A,B,C\}$

$(D)^+ = \{D,A,C,E,B\}$

Using F functional dependencies -

$(A)^+ = \{A,B,C\}$

$(AB)^+ = \{A,B,C\}$

$(D)^+ = \{D,A,C,E,B\}$

Step 2 : For G

Using F functional dependencies –

$(A)^+ = \{A,B,C\}$

$(D)^+ = \{D,A,C,E,B\}$

Using G functional dependencies -

$(A)^+ = \{A,B,C\}$

$(D)^+ = \{D,A,C,E,B\}$

Step 3 : From both these steps

G⊇F and F⊇G

Hence F and G are equivalent.

Minimal cover

Formal definition : A minimal cover for a set F of FDs is a set G of FDs such that :

1) Every dependency in G is of the form X -> A, where A is a single attribute.
2) **The closure F^+ is equal to the closure G^+.**

3) If we obtain a set H of dependencies from G by deleting one or more dependencies or by deleting attributes from a dependency in G, then $F^+ \neq H^+$.

Concept of extraneous attributes

Definition : An attribute of a functional dependency is said to be extraneous if we can remove it without changing the closure of the set of functional dependencies. The formal definition of extraneous attributes is as follows :

Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F

- Attribute A is extraneous in α if $A \in \alpha$ and F logically implies
  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$
- Attribute A is extraneous in β if $A \in \beta$ and the set of functional dependencies
  $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha \rightarrow (\beta - A)\}$ logically implies F.

Algorithm for computing canonical cover for set of functional dependencies F

$F_c = F$

repeat

Use the union rule to replace any dependencies in F_c of the form

$\alpha 1 \rightarrow \beta 1$ and $\alpha 1 \rightarrow \beta 2$ and $\alpha 1 \rightarrow \beta 1\ \beta 2$

Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β.

/* The test for extraneous attributes is done using F_c, not F */

If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$ in F_c.

until (F_c does not change)

Example 5.3.4 *Consider the following functional dependencies over the attribute set R(ABCDE) for finding minimal cover FD = {A -> C, AC -> D, B -> ADE}.*

Solution :

Step 1 : Split the FD such that R.H.S contain single attribute. Hence we get

A -> C

AC -> D

B -> A

B -> D

B -> E

Step 2 : Find the **redundant entries** and **delete** them. This can be done as follows -

- **For A -> C :** We find $(A)^+$ by assuming that we delete A -> C temporarily. We get $(A)^+ = \{A\}$. Thus from A it is not possible to obtain C by deleting A -> C. This means we can not delete A -> C.
- **For AC -> D :** We find $(AC)^+$ by assuming that we delete AC -> D temporarily. We get $(AC)^+ = \{AC\}$. Thus by such deletion it is not possible to obtain D. This means we can not delete AC -> D.
- **For B -> A :** We find $(B)^+$ by assuming that we delete B -> A temporarily. We get $(B)^+ = \{BDE\}$. Thus by such deletion it is not possible to obtain A. This means we can not delete B -> A.
- **For B -> D :** We find $(B)^+$ by assuming that we delete B -> D temporarily. We get $(B)^+ = \{BEACD\}$. This shows clearly that even if we delete B -> D we can obtain D. This means we can delete B -> A. Thus it is redundant.
- **For B -> E :** We find $(B)^+$ by assuming that we delete B -> E temporarily. We get $(B)^+ = \{BDAC\}$. Thus by such deletion it is not possible to obtain E. This means we can not delete B->E.

To summarize we get now

A -> C

AC -> D

B -> A

B -> E

Thus R.H.S gets simplified.

Step 3 : Now we will simplify L.H.S.

Consider AC -> D. Here we can split A and C. For that we find closure set of A and C.

$(A)^+ = (AC)$

$(C)^+ = (C)$

Thus C can be obtained from both A as well as C. That also means we need not have to have AC on L.H.S. Instead, only A can be allowed and C can be eliminated. Thus after simplification we get

A ->D

To summarize we get now

A ->C

A ->D

B ->A

B ->E

Thus L.H.S gets simplified.

Step 3 : The simplified L.H.S. and R.H.S can be combined together to form

A ->CD

B ->AE

This is a **minimal cover** or **canonical cover** of functional dependencies.

Example 5.3.5 *A relation R(A,C,D,E,H) satisfies the following FDs A -> C, AC -> D, E -> AD, E -> H. Find the canonical cover for this set of FD's.*

VTU : July-18, Marks 6

Solution : For obtaining canonical cover we have to find the redundant entries from both LHS and RHS and eliminate them.

Step 1 : Suppose we minimize LHS first, then go through each production rule one by one considering LHS.

A -> C , Keep it as it is.

AC -> D, Here A -> C and A -> D, So we remove A -> C , hence **A -> D** is kept by eliminating C from LHS.

E -> AD, keep it as it is as E is a single attribute at LHS.

E -> H, keep it as it is

Step 2 : Now we will minimize RHS.

A -> C , keep it as it is

A -> D, keep it as it is

E -> AD. That means E -> A and E -> D.

As A -> D is also present in the FD, so we get E -> A and A -> D. Thus E -> D is transitive . Hence neglect it. So we keep **E -> A** only

E -> H, Keep it as it is.

Step 3 : From steps 1 and 2, we get minimal cover of FD as

A -> C

A -> D

E -> A

E -> H

Hence the canonical for is

A -> CD

E -> AH

Example 5.3.6 *What is functional dependency ? Find the minimal cover using the minimal cover algorithm for the following functional dependency*

F = {AB -> D,B -> C, AE -> B, A -> D, D -> EF}

VTU : July-19, Marks 8

Solution :

Step 1 : We will make right hand sides atomic

AB -> D

B -> C

AE -> B

A -> D

D -> E

D -> F

Step 2 : Now we will remove redundant FDs using RHS

- For AB -> D. Now compute (AB)+ without considering the AB -> D i.e.{G-(AB->D)}

We get (AB)+ = {ABCDEF}. That means we can remove AB -> D as it is redundant entry.

Hence grammar is

B -> C

AE -> B

A -> D

D -> E

D -> F

- For B -> C compute (B)+ by considering {G - (B -> C)}

(B)+ = {AEBDF}. As C is not present in this set. That means B -> C is not redundant. So we can not remove it.

Hence grammar is

B -> C

AE -> B

A -> D

D -> E

D -> F

- For AE -> B, we will compute (AE)+ under (G – (AE -> B))

(AE)+ = {AEDF} as B is not present in (AE)+. So we cannot remove AE -> B from grammar. Hence grammar will be

B -> C

AE -> B

A -> D

D -> E

D -> F

For A -> D, we will compute(A)+ under (G - (A -> D))

(A)+ = {AEBC}. As D is not present in (A)+, hence we can not eliminate A -> D. The grammar is

B -> C

AE -> B

A -> D

D -> E

D -> F

- For D -> E, compute (D)+ under (G - (D -> E))

(D)+ = {DF}. As E is not present in (D)+, We cannot remove D -> E

- For D -> F, compute (D)+ under (G-(D -> F))

(D)+ = {DE}. As F is not present in (D)+, We cannot remove D -> F. Finally the grammar is

B -> C

AE -> B

A -> D

D -> E

D -> F

Step 3 : Remove redundant entries based on RHS

B -> C, A -> D, D -> E and D -> F as LHS is atomic.

Now we consider AE -> B

For A : compute E+ with respect to (G - (AE→B) ∪ (E→B))

E+ using {B→C, E→B, A→D, D→E, D→F} = EBC

E+ doesn't contain A, so A not redundant in AE→B

For E : compute A+ with respect to (G - (AE→B) ∪ (A→B))

A+ using {B→C, A→B, A→D, D→E, D→F} = ABDEFC

A+ contains E, so E is redundant in AE→B

Hence we consider AE -> B as A -> B.

Finally minimal closure is {B -> C,A -> B, A -> D, D -> E, D -> F}

Example 5.3.7 *Using the minimal cover algorithm, find the minimal cover for the following FDs :*

F = {AB -> C, A -> D, BD -> C, D -> BG, AE -> F}.

VTU : Jan.-20, Marks 10

Solution : We will make right hand side atomic

AB -> C

A -> D

BD -> C

D -> B

D -> G

AE -> F

Step 2 : Now we will remove redundant FDs using RHS

- For AB -> C we compute (AB)+ without considering the rule AB -> C i.e. (G - (AB -> C)). We get (AB)+ = {ABDBGC} i.e C is present by other way also in the set. Hence we can **remove AB -> C** as it is redundant entry.
- For A -> D, we compute (A)+ using (G - (A -> D)). We get (A)+ = {A}. We can not get D. So it is not a redundant entry and we can not remove it.
- For BD -> C, We compute (BD)+ using (G - (BD -> C)). We get (BD)+ ={BDG}. This is also not a redundant entry and we can not remove it.
- For D -> B. Let us compute (D)+ using (G - (D -> B)). We get (D)+={DG}. This again indicates that we can not get B without the rule D -> B. Hence it is not a redundant entry and we can not remove it.
- Similarly, we can conclude For D -> G and AE -> F as not redundant entries.
- Finally, the grammar will be

A -> D

BD -> C

D -> B

D -> G

AE -> F

Step 2 : Now we will remove redundant entries based on LHS.

The A -> D, D -> B, D -> G remain as it is in minimal cover as LHS is atomic.

Now consider BD -> C, We can replace this by B -> C and eliminate D, as D -> B is present.

Similarly consider AE -> F. But as we cannot replace it either by A->F or E->F. So it is not redundant.

The minimal cover is {A->D, D->B. D->G, B->C, AE->F}.

5.4 Properties of Relational Decompositions

- Decomposition is the process of breaking down one table into multiple tables.
- **Formal definition of decomposition is -**
- A decomposition of relation schema R consists of replacing the relation schema by two relation schema that each contain a subset of attributes of R and together include all attributes of R by storing projections of the instance.
- For example - Consider the following table

Employee_Department table as follows -

Eid	Ename	Age	City	Salary	Deptid	DeptName
E001	ABC	29	Pune	20000	D001	Finance
E002	PQR	30	Pune	30000	D002	Production
E003	LMN	25	Mumbai	5000	D003	Sales
E004	XYZ	24	Mumbai	4000	D004	Marketing
E005	STU	32	Hyderabad	25000	D005	Human Resource

We can decompose the above relation Schema into two relation schemas as **Employee (Eid, Ename, Age, City, Salary)** and **Department (Deptid, Eid, DeptName)** as follows -

Employee Table

Eid	Ename	Age	City	Salary
E001	ABC	29	Pune	20000
E002	PQR	30	Pune	30000
E003	LMN	25	Mumbai	5000
E004	XYZ	24	Mumbai	4000
E005	STU	32	Hyderabad	25000

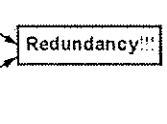
Department Table

Deptid	Eid	DeptName
D001	E001	Finance
D002	E002	Production
D003	E003	Sales
D004	E004	Marketing
D005	E005	Human Resource

- The decomposition is used for eliminating redundancy.
- **For example :** Consider following relation **Schema R** in which we assume that the grade determines the salary, the redundancy is caused

Schema R

Name	eid	deptname	Grade	Salary
AAA	121	Accounts	2	8000
AAA	132	Sales	3	7000
BBB	101	Marketing	4	7000
CCC	106	Purchase	2	8000



- Hence, the above table can be decomposed into two Schema S and T as follow :

Schema S

Name	eid	deptname	Grade
AAA	121	Accounts	2
AAA	132	Sales	3
BBB	101	Marketing	4
CCC	106	Purchase	2

Schema T

Grade	Salary
2	8000
3	7000
4	7000
2	8000

Problems related to decomposition :

Following are the potential problems to consider :

1) Some queries become more **expensive.**
2) Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation!
3) Checking some dependencies may require joining the instances of the decomposed relations.
4) There may be loss of information during decomposition.

Properties associated with decomposition

There are two properties associated with decomposition and those are -

1) **Loss-less join or non loss decomposition :** When all information found in the original database is preserved after decomposition, we call it as loss less or non loss decomposition.
2) **Dependency preservation :** This is a property in which the constraints on the original table can be maintained by simply enforcing some constraints on each of the smaller relations.

5.5 Loss-Less Join

The lossless join can be defined using following three conditions :

i) **Union** of attributes of R1 and R2 must be equal to attribute of R. Each attribute of R must be either in R1 or in R2.

Att(R1) ∪ Att(R2) = Att(R)

ii) **Intersection** of attributes of R1 and R2 must not be NULL.

Att(R1) ∩ Att(R2) ≠ Φ

iii) **Common attribute** must be a key for at least one relation (R1 or R2)

Att(R1) ∩ Att(R2) -> Att(R1)

or **Att**(R1) ∩ Att(R2) -> Att(R2)

Example 5.5.1 *Consider the following relation R(A, B, C, D)and FDs A->BC, is the decomposition of R into R1(A, B, C), R2(A, D). Check if the decomposition is lossless join or not.*

Solution :

Step 1 : Here Att(R1) ∪ Att(R2) = Att(R) i.e R1(A,B,C) ∪ R2(A,D) = (A,B,C,D) i.e R.

Thus first condition gets satisfied.

Step 2 : Here R1 ∩ R2 = {A}. Thus Att(R1) ∩ Att(R2) ≠ ∅. Here the second condition gets satisfied.

Step 3 : Att(R1) ∩ Att(R2) -> {A}. Now (A)+ = {A,B,C} ∈ attributes of R1. Thus the third condition gets satisfied.

This shows that the given decomposition is a **lossless join.**

Example 5.5.2 *Consider the following relation R(A, B, C, D, E, F) and FDs A->BC, C->A, D->E, F->A, E->D is the decomposition of R into R1(A, C, D), R2(B, C, D) and R3 (E, F, D). Check for lossless.*

Solution :

Step 1 : R1 ∪ R2 ∪ R3 = R. Here the first condition for checking lossless join is satisfied as (A,C,D) ∪ (B,C,D) ∪ (E,F,D) = {A,B,C,D,E,F} which is nothing but R.

Step 2 : Consider R1∩ R2 = {CD} and R2∩R3 = {D}. Hence second condition of intersection not being Φ gets satisfied.

Step 3 : Now, consider R1(A, C, D) and R2(B, C, D). We find R1∩R2 = {CD}

$(CD)^+$ = {ABCDE} ∈ attributes of R1 i.e.{A, C, D}. Hence condition 3 for checking lossless join for R1 and R2 gets satisfied.

Step 4 : Now, consider R2(B, C, D) and R3(E, F, D) . We find R2∩R3={D}.

$(D)^+$ = {D, E} which is neither complete set of attributes of R2 or R3.

[Note that F is missing for being attribute of R3].

Hence it is not **lossless join decomposition.** Or in other words we can say it is a **lossy decomposition.**

Example 5.5.3 *Suppose that we decompose schema R = (A,B,C,D,E) into (A,B,C) (C,D,E) Show that it is not a lossless decomposition.*

Solution :

Step 1 : Here we need to assume some data for the attributes A, B, C, D, and E. Using this data we can represent the relation as follows -

Relation R

A	B	C	D	E
a	1	x	p	q
b	2	x	r	s

Relation R1 = (A,B,C)

A	B	C
a	1	x
b	2	x

Relation R2 = (C,D,E)

C	D	E
x	p	q
x	r	s

Step 2 : Now we will join these tables using natural join, i.e. the join based on common attribute C. We get R1 ⋈ R2 as

A	B	C	D	E
a	1	x	p	q
a	1	x	r	s
b	2	x	p	q
b	2	x	r	s

Here we get more rows or tuples than original relation R

Clearly R1 ⋈ R2 ⊄ R. Hence it is not lossless decomposition.

5.6 Dependency Preservation

- **Definition :** A decomposition D = {R1, R2, R3....Rn} of R is dependency preserving for a set F of functional dependency if - (F1 ∪ F2 ∪ ... ∪ Fm) = F.
- If decomposition is not dependency-preserving, some dependency is lost in the decomposition.

Example 5.6.1 *Consider the relation R (A, B, C) for functional dependency set {A -> B and B → C} which is decomposed into two relations R1 = (A, C) and R2 = (B, C). Then check if this decomposition dependency preserving or not.*

Solution : This can be solved in following steps :

Step 1 : For checking whether the decomposition is dependency preserving or not we need to check following condition

$$F^+ = (F1 \cup F2)^+$$

Step 2 : We have with us the F^+ ={ A->B and B->C }

Step 3 : Let us find $(F1)^+$ for relation R1 and $(F2)^+$ for relation R2

R1(A,C)	R2(B,C)
A->A Trivial	B->B Trivial
C->C Trivial	C->C Trivial
A->C ∵ In (F)+A->B->C and it is Nontrivial	B->C ∵ In $(F)^+$ B->C and it is Non-Trivial
AC->AC Trivial	BC->BC Trivial
A->B but is not useful as B is not part of R1 set	We can not obtain C->B
We can not obtain C->A	

Step 4 : We will eliminate all the trivial relations and useless relations. Hence we can obtain R1 and R2 as

R1(A,C)	R2(B,C)
A->C Nontrivial	B->C Non-Trivial

$(F1 \cup F2)^+ = \{A->C, B->C\} \neq \{A->B, B->C\}$ i.e. $(F)^+$

Thus the condition specified in step 1 i.e. F+ = (F1 ∪ F2)+ **is not true**. Hence it is **not dependency preserving decomposition.**

Example 5.6.2 *Let relation R(A, B, C, D) be a relational schema with following functional dependencies {A->B, B->C, C->D and D->B}. The decomposition of R into (A, B), (B, C) and (B, D). Check whether this decomposition is dependency preserving or not.*

Solution :

Step 1 : Let $(F)^+ = \{A->B, B->C, C->D, D->B\}$.

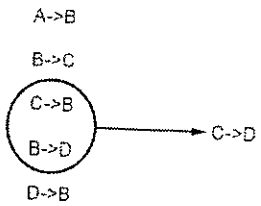
Step 2 : We will find $(F1)^+$, $(F2)^+$, $(F3)^+$ for relations R1(A,B) , R2(B,C) and R3(B,D) as follows -

R1(A,B)	R2(B,C)	R3(B,D)
A->A Trivial	B->B Trivial	B->B Trivial
B->B Trivial	C->C Trivial	D->D Trivial
A->B ∵ $(F)^+$ and it's non Trivial	B->C ∵ $(F)^+$ and it's non Trivial	B-> D ∵ $(F)^+$ as and B->C->D and it's non Trivial
B->A can not be obtained	C->B ∵ In $(F)^+$ and C->D->C and it is Nontrivial	D->B ∵ $(F)^+$ and it's non Trivial
AB->AB	BC->BC Trivial	BD->BD Trivial

Step 3 : We will eliminate all the trivial relations and useless relations. Hence we can obtain R1 ∪ R2 ∪ R3 as

R1(A,B)	R2(B,C)	R2(B,D)
A->B	B->C	B-> D
	C->B	D->B

Step 4 : As from above FD's we get



Step 5 : This proves that $F^+ = (F1 \cup F2 \cup F3)^+$. Hence given **decomposition is dependency preserving.**

5.7 Normal Forms

- Normalization is the process of reorganizing data in a database so that it meets **two basic requirements :**

1) There is **no redundancy** of data (all data is stored in only one place) and
2) **Data dependencies** are logical (all related data items are stored together)

Need for normalization

1) **It eliminates redundant data.**
2) **It reduces chances of data error.**
3) The normalization is important because it allows database to take up **less disk space.**
4) It also help in increasing the **performance.**
5) It improves the data integrity and consistency.

5.8 First Normal Form

VTU : Jan.-19, Marks 4

The table is said to be in 1NF if it follows following rules -

i) It should only have single (atomic) valued attributes/columns.
ii) Values stored in a column should be of the same domain.
iii) All the columns in a table should have unique names.
iv) And the order in which data is stored, does not matter.

Consider following student table

Student

sid	sname	Phone
1	AAA	11111 22222
2	BBB	33333
3	CCC	44444 55555

As there are multiple values of phone number for sid 1 and 3, the above table is not in 1NF. We can make it in 1NF. The conversion is as follows -

sid	sname	Phone
1	AAA	11111
1	AAA	22222
2	BBB	33333
3	CCC	44444
3	CCC	55555

University Question

1. *Suggest and explain three different techniques to achieve 1NF. using suitable example.*

VTU : Jan.-19, Marks 4

5.9 Second Normal Form

Before understanding the second normal form let us first discuss the concept of partial functional dependency and prime and non prime attributes.

Concept of partial functional dependency

Partial dependency means that a nonprime attribute is functionally dependent on part of a candidate key.

For example : Consider a relation R(A,B,C,D) with functional dependency {AB->CD, A->C}

Here (AB) is a candidate key because

$$(AB)^+ = \{ABCD\} = \{R\}$$

Hence {A,B} are prime attributes and {C,D} are non prime attribute. In A->C, the non prime attribute C is dependent upon A which is actually a part of candidate key AB. Hence due to A->C we get partial functional dependency.

Prime and non prime attributes

- **Prime attribute :** An attribute, which is a part of the candidate-key, is known as a prime attribute.
- **Non-prime attribute :** An attribute, which is not a part of the prime-key, is said to be a non-prime attribute.
- **Example : Consider a Relation** R = {A,B,C,D} and candidate key as AB, the Prime attributes : A, B.

Non Prime attributes : C, D

The second normal form

For a table to be in the Second Normal Form, following conditions must be followed

i) It should be in the First Normal form.

ii) It should not have partial functional dependency.

For example : Consider following table in which every information about a the Student is maintained in a table such as student id(sid), student name(sname), course id(cid) and course name(cname).

Student_Course

sid	sname	cid	cname
1	AAA	101	C
2	BBB	102	C++
3	CCC	101	C
4	DDD	103	Java

This table is not in 2NF. For converting above table to 2NF we must follow the following steps -

Step 1 : The above table is in 1NF.

Step 2 : Here **sname** and **sid** are associated similarly **cid** and **cname** are associated with each other. Now if we delete a record with **sid = 2,** then automatically the course C++ will also get deleted. Thus,

sid->sname or **cid->cname** is a partial functional dependency, because **{sid,cid}** should be essentially a candidate key for above table. Hence to bring the above table to 2NF we must decompose it as follows :

Student

sid	sname	cid
1	AAA	101
2	BBB	102
3	CCC	101
4	DDD	103

Here candidate key is (sid, cid) and (sid, cid)->sname

Course

cid	cname
101	C
102	C++
101	C
103	Java

Here candidate key is cid
here cid- > cname

Thus now table is in 2NF as there is no partial functional dependency.

Example 5.9.1 *Study the relation given below and state what level of normalization can be achieved and normalize it upto that level.*

Order no.	*Order date*	*Item lines*		
		Item code	*Quantity*	*Price/Unit*
1456	*26-12-1999*	*3687*	*52*	*50.4*
		4627	*38*	*60*
		3214	*20*	*20.00*
1886	*04-03-1999*	*4629*	*45*	*20.25*
		4627	*30*	*60.20*
1788	*04-04-1999*	*4627*	*40*	*60.20*

Solution :

Reason for the given relation being unnormalized

1. Observe order for many items.
2. Item lines has many attributes-called composite attributes.
3. Each tuple has variable length.
4. Difficult to store due to non-uniformity.
5. Given item code difficult to find qty-ordered and hence called Unnormalized relation.

For conversion to first normal form -

- Identify the composite attributes, convert the composite attributes to individual attributes.
- Duplicate the common attributes as many times as lines in composite attribute.
- Every attribute now describes single property and not multiple properties, some data will be duplicated.

- Now this is called First normal form (1NF) also called flat file.

Order no.	Order date	Item lines		
		Item code	Quantity	Price/Unit
1456	26-12-1999	3687	52	50.4
1456	26-12-1999	4627	38	60
1456	26-12-1999	3214	20	20.00
1886	04-03-1999	4629	45	20.25
1886	04-03-1999	4627	30	60.20
1788	04-04-1999	4627	40	60.20

Fig. 5.9.1 Table in first normal form

- The above table has insertion, deletion and update anomalies. For instance - if we delete order no. 1886, then the item code 4629 gets lost. Similarly if we update 4627, then all instances of 4627 need to be changed.
- We need to convert 2NF if it is in 1NF. The non-key attributes are functionally dependent on key attribute and if there is a composite key then no non-key attribute is functionally depend on one part of the key.
- The table can be converted to 2NF as follows -

Orders

OrderNo	OrderDate
1456	26-12-1999
1886	04-03-1999
1788	04-04-1999

Order details

OrderNo	ItemCode	Qty
1456	3687	52
1886	4629	45
1788	4627	40

Prices

ItemCode	Price/Unit
3687	50.4
4627	60
3214	20
4629	20.25

5.10 Third Normal Form

VTU : July-18, 19, Jan.-19, 20, Marks 10

Before understanding the third normal form let us first discuss the concept of transitive dependency, super key and candidate key.

Concept of transitive dependency

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For example -

X -> Z is a transitive dependency if the following functional dependencies hold true :

X->Y

Y->Z

Concept of super key and candidate key

Superkey : A super key is a set or one of more columns (attributes) to uniquely identify rows in a table.

Candidate key : The minimal set of attribute which can uniquely identify a tuple is known as candidate key. For example consider following table

RegID	RollNo	Sname
101	1	AAA
102	2	BBB
103	3	CCC
104	4	DDD

Superkeys

- {RegID}
- {RegID, RollNo}
- {RegID, Sname}

- {RollNo, Sname}
- {RegID, RollNo, Sname}

Candidate keys

- {RegID}
- {RollNo}

Third normal form

A table is said to be in the third normal form when,

i) It is in the second normal form.(i.e. it does not have partial functional dependency).

ii) It doesn't have transitive dependency.

Or in other words

In other words 3NF can be defined as : A table is in 3NF if it is in 2NF and for each functional dependency

X-> Y

at least one of the following conditions hold :

i) X is a super key of table.

ii) Y is a prime attribute of table.

For example : Consider following table **Student_details** as follows -

sid	sname	zipcode	cityname	state
1	AAA	11111	Pune	Maharashtra
2	BBB	22222	Surat	Gujarat
3	CCC	33333	Chennai	Tamilnadu
4	DDD	44444	Jaipur	Rajastan
5	EEE	55555	Mumbai	Maharashtra

Here

Super keys : {sid},{sid,sname},{sid,sname,zipcode}, {sid,zipcode,cityname}... and so on.

Candidate keys : {sid}

Non-prime attributes : {sname,zipcode,cityname,state}

The dependencies can be denoted as

sid->sname
sid->zipcode
zipcode->cityname
cityname->state

The above denotes the transitive dependency. Hence above table is not in 3NF. We can convert it into 3NF as follows :

Student

sid	sname	zipcode
1	AAA	11111
2	BBB	22222
3	CCC	33333
4	DDD	44444
5	EEE	55555

Zip

zipcode	cityname	state
11111	Pune	Maharashtra
22222	Surat	Gujarat
33333	Chennai	Tamilnadu
44444	Jaipur	Rajasthan
55555	Mumbai	Maharashtra

Example 5.10.1 *Consider the relation R = {A, B, C, D, E, F, G, H, I, J} and the set of functional dependencies F = {{A, B} → C, A→ {D, E}, B→ F, F→{G, H}, D→{I, J} }*

1. What is the key for R ? Demonstrate it using the inference rules.

2. Decompose R into 2NF, then 3NF relations.

VTU : July-18, Marks 6

Solution : Let,

A →DE (given)

∴ A →D, A → E (decomposition rule)

As D →I J, A → I J

Using union rule we get

A →DEIJ

As A →A

we get A →ADEIJ

Using augmentation rule we compute AB

AB →ABDEIJ

But AB →C (given)

∴ AB →ABCDEIJ

B →F (given) F → GH ∴ B → GH (transitivity)

∴ AB →AGH is also true

Similarly AB →AF ∵ B → F (given)

Thus now using union rule

AB →ABCDEFGHIJ

∴ AB is a key

The table can be converted to 2NF as

R_1 = (A, B, C)

R_2 = (A, D, E, I, J)

R_3 = (B, F, G, H)

The above 2NF relations can be converted to 3NF as follows

R_1 = (A, B, C)

R_2 = (A, D, E)

R_3 = (D, I, J)

R_4 = (B, E)

R_5 = (E, G, H).

Example 5.10.2 *A software contract and consultancy firm maintains details of all the various projects in which its employees are currently involved. These details comprise :*

- *Employee number*
- *Employee name*
- *Date of birth*
- *Department code*
- *Department name*
- *Project code*

- *Project description*
- *Project supervisor*

Assume the following :

- *Each employee number is unique.*
- *Each department has a single department code.*
- *Each project has a single code and supervisor.*
- *Each employee may work on one or more projects.*
- *Employee names need not necessarily be unique.*
- *Project code, project description and project supervisor are repeating fields.*

Normalise this data to third normal form.

Solution :

Un-Normalized Form

Employee Number, Employee Name_Date of Birth_Department Code_Department Name_Project Code_Project Description_Project Supervisor

1NF

Employee Number, Employee Name_Date of Birth
Department Code, Department Name
Employee Number, Project Code, Project Description_Project Supervisor

2NF

Employee Number, Employee Name_Date of Birth_Department Code_Department Name
Employee Number,Project Code,
Project Code, Project Description,Project Supervisor

3NF

Employee Number, Employee Name_Date of Birth_*Department Code
Department Code, Department Name
Employee Number, Project Code
Project Code, Project Description, Project Supervisor

Example 5.10.3 *What is normalization ?Normalize below given relation upto 3NF STUDENT.*

StudID	*StudName*	*City*	*Pincode*	*ProjectID*	*ProjectName*	*Course*	*Content*
S101	*Ajay*	*Surat*	*326201*	*P101*	*Health*	*Programming*	*C++, Java, C*
S102	*Vijay*	*Pune*	*325456*	*P102*	*Social*	*WEB*	*HTML, PHP, ASP*

Solution : For converting the given schema to first normal form, we will arrange it in such a way that have each tuple contains single record. For that purpose we need to split the schema into two tables namely **Student** and **Projects.**

1NF

Student

StudID	StudName	Pincode	City
S101	Ajay	326201	Surat
S102	Vijay	325456	Pune

Projects

StudID	ProjectID	ProjName	Course	Content
S101	P101	Health	Programming	C++
S101	P101	Health	Programming	Java
S101	P101	Health	Programming	C
S102	P102	Social	WEB	HTML
S102	P102	Social	WEB	PHP
S102	P102	Social	WEB	ASP

2NF

For a table to be in 2NF, there should not be any partial dependency.

Student

StudID	StudName	Pincode	City
S101	Ajay	326201	Surat
S102	Vijay	325456	Pune

Project

StudID	ProjectID	ProjName	CourseID
S101	P101	Health	C101
S101	P101	Health	C102
S101	P101	Health	C103
S102	P102	Social	C104
S102	P102	Social	C105
S102	P102	Social	C106

CourseDetails

CourseID	Course	Content
C101	Programming	C++
C102	Programming	Java
C103	Programming	C
C104	WEB	HTML
C105	WEB	PHP
C106	WEB	ASP

3NF

There was a transitive dependency in 2NF tables because city is associated with student ID and city depends upon zip code. Hence the transitive dependency is removed to covert table into 3NF. The required 3NF schema is as below -

Student

StudID	StudName	Pincode
S101	Ajay	326201
S102	Vijay	325456

Student_Address

Pincode	City
326201	Surat
325456	Pune

Project

StudID	ProjectID	ProjName	CourseID
S101	P101	Health	C101
S101	P101	Health	C102
S101	P101	Health	C103
S102	P102	Social	C104
S102	P102	Social	C105
S102	P102	Social	C106

CourseDetails

CourseID	Course	Content
C101	Programming	C++
C102	Programming	Java
C103	Programming	C
C104	WEB	HTML
C105	WEB	PHP
C106	WEB	ASP

Example 5.10.4 *What is the need for normalization ? Consider the relation : Emp-proj = {ssn, Pnumber, Hours, Ename, Pname, Plocation}*

Assume {ssn,Pnumber} as primary key.

The dependencies are:

{ssn,Pnumber}->Hours

Ssn-> Ename

Pnumber ->{Pname,Plocation}

Normalize the above relation to 3NF.

VTU : July-19, Marks 8

Solution : Need for normalization - Refer section 5.7.

Consider the given dependencies

(1) {ssn,Pnumber}->Hours

(2) ssn-> Ename

(3) Pnumber ->{Pname,Plocation}

The dependencies 2 and 3 represents the partial dependency. Hence we convert the relation into second normal form by splitting the given Emp-proj into three relations

Emp = {ssn, Ename}

Proj = {Pnumber, Pname, Plocation}

Works = {ssn, Pnumber, Hours}

Example 5.10.5 *Consider the following relation for CARSALE(CAR-NO,Date-Sold,Salesman-no, Commission, Discount).*

Assume a car can be sold by multiple salesman and hence primary is (CAR-NO,Salesman-no)

Additional dependencies are

Date_Sold ->Discount

Salesman_no ->Commission

i) Is this relation in 1NF, 2NF, 3NF ? Why and Why not ?

ii) How would you normalize this completely ?

VTU : Jan.-19, Marks 6

Solution :

i) Let us check the database schema against each normal form.

First normal form : As the relation have no multivalued attributes or nested relations, the given relation is in 1st normal form.

Second normal form : This relation is not in second normal form because the attribute **commission** is dependent on part of primary key **Salesman-no.**

Third normal form : This relation is not in third normal form because firstly it is not in 2nd normal form and there should be transitive dependency of a nonkey attribute on primary key.

(CAR-No, Salesman-no) ->Date_sold->Discount

(ii) To normalize this relation we will decompose it into

R1={CAR-No, Salesman-no, Date_sold}

R2={Date_sold, Discount}

R3={Salesman-no, commission}

The functional dependency is as follows -

F1={(CAR-No, Salesman-no)→Date_sold}

F2={Date_sold→Discount}

F3={Salesman-no→commission}

Example 5.10.6 *Normalize the below relation upto 3NF*

Module	*Dept*	*Lecturer*	*Text*
M1	*D1*	*L1*	*T1*
M1	*D1*	*L1*	*T2*
M2	*D1*	*L1*	*T1*
M2	*D1*	*L1*	*T3*
M3	*D1*	*L2*	*T4*
M4	*D2*	*L3*	*T1*
M4	*D2*	*3*	*T5*
M5	*D2*	*L4*	*T6*

VTU : Jan.-20, Marks 10

Solution : The given relation is already in 1st normal form. But it has Insert, delete and update anomalies. Because -

1) Insert anomalies : We can not add a module(M) with no texts(T).

2) Delete anomalies : If we remove M3, we remove L2 as well.

3) Update anomalies : To change lecturer for M1, we have to change two rows.

Hence we will convert it to second normal form.

Step 1 : We can define the functional dependency FD as

{Module, Text}->{Lecturer, Dept}

But

{Module}->{Lecturer, Dept}

That means Lecturer and Dept are partially dependent on the primary key. Hence for conversion of first normal form to second normal form we will decompose the give table into two tables as

Table 2a

Module	Dept	Lecturer
M1	D1	L1
M2	D1	L1
M3	D1	L2
M4	D2	L3
M5	D2	L4

Table 2b

Module	Text
M1	T1
M1	T2
M2	T1
M2	T3
M3	T4
M4	T1
M4	T5
M5	T6

The relation is now in second normal form.

Step 3 : The table 2a has Insert, Delete and Update anomalies. Because -

1) INSERT anomalies : We can't add lecturers who teach no modules.

2) UPDATE anomalies : To change the department for L1 we must alter two rows.

3) DELETE anomalies : If we delete M3 we delete L2 as well.

Hence, to eliminate these anomalies, we decompose table 2a into two tables and convert it to third normal form.

Step 4 : Hence we get

Table 3a

Lecturer	Dept.
L1	D1
L2	D1
L3	D2
L4	D2

Table 3b

Module	Lecturer
M1	L1
M2	L1
M3	L2
M4	L3
M5	L4

Step 5 : Thus now the complete relation is decomposed into three tables and it is in third normal form. It is summarized as below

Table 3a

Lecturer	Dept.
L1	D1
L2	D1
L3	D2
L4	D2

Table 3b

Module	Lecturer
M1	L1
M2	L1
M3	L2
M4	L3
M5	L4

Table 2b

Module	Text
M1	T1
M1	T2
M2	T1
M2	T3
M3	T4
M4	T1
M4	T5
M5	T6

University Question

1. *Which normal form is based on the concept of transitive functional dependency ? Explain the same with an example.* **VTU : July-19, Marks 8**

5.11 Boyce/Codd Normal Form(BCNF)

Boyce and Codd Normal Form is a **higher version** of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF.

A 3NF table which **does not have multiple overlapping** candidate keys is said to be in BCNF.

Or in other words,

For a table to be in BCNF, following conditions must be satisfied :

i) R must be in 3rd Normal Form

ii) For each functional dependency ($X \rightarrow Y$), X should be a super key. In simple words if Y is a prime attribute then X can not be non prime attribute.

For example - Consider following table that represents that a student enrollment for the course -

Enrollment table

sid	Course	Teacher
1	C	Ankita
1	Java	Poonam
2	C	Ankita
3	C++	Supriya
4	C	Archana

From above table following observations can be made :

- One student can enroll for multiple courses. For example student with sid = 1 can enroll for C as well as Java.
- For each course, a teacher is assigned to the student.
- There can be multiple teachers teaching one course for example course C can be taught by both the teachers namely - Ankita and Archana.
- The candidate key for above table can be (sid, course), because using these two columns we can find,

- The above table holds following dependencies
  - (sid, course)->Teacher
  - Teacher->course
- The above table is not in BCNF because of the dependency teacher->course. Note that the teacher is not a superkey or in other words, teacher is a non prime attribute and course is a prime attribute and non-prime attribute derives the prime attribute.
- To convert the above table to BCNF we must decompose above table into student and course tables

Student

sid	Teacher
1	Ankita
1	Poonam
2	Ankita
3	Supriya
4	Archana

Course

Teacher	Course
Ankita	C
Poonam	Java
Ankita	C
Supriya	C++
Archana	C

Now the table is in BCNF

Example 5.11.1 *Consider a relation(A,B,C,D) having following FDs.{AB->C, AB->D, C->A, B->D}. Find out the normal form of R.*

Solution :

Step 1 : We will first find out the candidate key from the given FD.

$$(AB)^+ = \{ABCD\} = R$$

$(BC)^+ = \{ABCD\} = R$

$(AC)^+ = \{AC\} \neq R$

There is no involvement of D on LHS of the FD rules. Hence D can not be part of any candidate key. Thus we obtain two candidate keys $(AB)^+$ and $(BC)^+$. Hence

prime attributes = {A,B,C}

Non prime attributes = {D}

Step 2 : Now, we will start checking from reverse manner, that means from BCNF, then 3NF, then 2NF.

Step 3 : For R being in BCNF for X->Y the X should be candidate key or super key.

From above FDs consider C->D in which C is not a candidate key or super key. Hence given relation is not in BCNF.

Step 4 : For R being in 3NF for X->Y either i) the X should be candidate key or super key or

ii) Y should be prime attribute. (For prime and non prime attributes refer step 1)

- For AB->C or AB->D the AB is a candidate key. Condition for 3NF is satisfied.
- Consider C->A. In this FD the C is not candidate key but A is a prime attribute. Condition for 3NF is satisfied.
- Now consider B->D. In this FD, the B is not candidate key, similarly D is not a prime attribute. Hence condition for 3NF fails over here.

Hence given relation is not in 3NF.

Step 5 : For R being in 2NF following condition **should not occur.**

Let X->Y, if X is a proper subset of candidate key and Y is a non prime attribute. This is a case of partial functional dependency.

For relation to be in 2NF there should not be any partial functional dependency.

- For AB->C or AB->D the AB is a **complete candidate key.** Condition for 2NF is satisfied.
- Consider C->A. In this FD the C is not candidate key. Condition for 2NF is satisfied.
- Now consider B->D. In this FD, the B is a part of candidate key(AB or BC), similarly D is not a prime attribute. That means partial functional dependency occurs here.

Hence condition for 2NF fails over here.

Hence given relation is not in 2NF.

Therefore we can conclude that the given relation **R is in 1NF.**

Example 5.11.2 *Consider a relation R(ABC) with following FD A->B, B->C and C->A. What is the normal form of R ?*

Solution :

Step 1 : We will find the candidate key

$(A)^+ = \{ABC\} = R$

$(B)^+ = \{ABC\} = R$

$(C)^+ = \{ABC\} = R$

Hence A, B and C all are candidate keys

Prime attributes = {A,B,C}

Non prime attribute{}

Step 2 : For R being in BCNF for X->Y the X should be candidate key or super key.

From above FDs

- Consider A->B in which A is a candidate key or super key. Condition for BCNF is satisfied.
- Consider B->C in which B is a candidate key or super key. Condition for BCNF is satisfied.
- Consider C->A in which C is a candidate key or super key. Condition for BCNF is satisfied.

This shows that the given relation **R is in BCNF.**

Example 5.11.3 *Consider table R(A,B,C,D,E) with FDs as A->B, BC->E, and ED->A.The table is in which normal form ? Justify your answer.*

Solution :

Step 1 : We will first find out the candidate keys for given relation R

$(ACD)^+ = \{A,B,C,D,E\}$

$(BCD)^+ = \{A,B,C,D,E\}$

$(CDE)^+ = \{A,B,C,D,E\}$

Step 2 : Let A->B, the ACD is candidate key and A is a partial key, B is a prime attribute(i.e. it is also part of candidate key). Hence A->B is not a partial functional dependency.

Similarly in BC->E and ED->A,

E and A are prime-attributes and hence both are not partial functional dependencies.

Hence R is in 2NF.

Step 3 : According to 3NF, every non-prime attribute must be dependent on the candidate key.

In the given functional dependencies, all dependent attributes are prime-attributes. Hence the relation R is in 3NF.

Step 4 : For R being in BCNF for X->Y the X should be candidate key or super key.

The table is not in BCNF, none of A, BC and ED contain a key.

Example 5.11.4 *A college maintains details of its lecturers' subject area skills. These details comprise :*

- *Lecturer number*
- *Lecturer name*
- *Lecturer grade*
- *Department code*
- *Department name*
- *Subject code*
- *Subject name*
- *Subject level*

Assume that each lecturer may teach many subjects but may not belong to more than one department.

Subject code, subject name and subject level are repeating fields.

Normalise this data to third normal form.

Solution :

Unnormalized form

Lecturer Number, Lecturer Name_Lecturer Grade_Department Code,Department Name_Subject Code, Subject Name_Subject Level

1NF

LecturerNumber	Lecturer Name	Lecturer Grade	Department Code	Department Name

Lecturer Number	Subject Code	Subject Name	Subject Level

2NF

Lecturer Number	_Lecturer Name	_Lecturer Grade	_Department Code	_Department Name

Lecturer Number	Subject Code

Subject Code	_Subject Name	_Subject Level

3NF

Lecturer Number	Lecturer Name	Lecturer Grade

*Department Code

Department Code	_Department Name

Lecturer Number	Subject Code

Subject Code	Subject Name	Subject Level

Example 5.11.5 *Prove that any relational schema with two attributes is in BCNF.*

Solution : Here, we will consider R={A,B} i.e. a relational schema with two attributes. Now various possible FDs are A->B, B->A.

From the above FDs

- Consider A->B in which A is a candidate key or super key. Condition for BCNF is satisfied.
- Consider B->A in which B is a candidate key or super key. Condition for BCNF is satisfied.
- Consider both A->B and B->A with both A and B is candidate key or super key. Condition for BCNF is satisfied.

- No FD holds in relation R. In this {A,B} is candidate key or super key. Still condition for BCNF is satisfied.

This shows that any relation R **is in BCNF** with two attributes.

Example 5.11.6 *Prove the statement "Every relation which is in BCNF is in 3NF but the converse is not true."*

Solution : For a relations to be in 3NF

A table is said to be in the Third Normal Form when,

i) It is in the Second Normal form. (i.e. it does not have partial functional dependency)

ii) It doesn't have transitive dependency.

Or in other words

In other words 3NF can be defined as : A table is in 3NF if it is in 2NF and for each functional dependency X-> Y

at least one of the following conditions hold :

iii) X is a super key of table

iv) Y is a prime attribute of table

For a relation to be in BCNF

1) It should be in 3NF

2) A 3NF table which **does not have multiple overlapping** candidate keys is said to be in BCNF.

For proving that the table can be in 3NF but not in BCNF consider Following relation

R(Student, Subject, Teacher) . Consider following are FDs

(Subject, Student)-> Teacher

Because subject and student combination gives unique teacher.

Teacher -> Subject

Because each teacher teaches only Subject.

(Teacher, Student)->Subject

- So, this relation is in 3NF as every non-key attribute is non-transitively fully functional dependent on the primary key.
- But it is not in BCNF. Because this is a case of **overlapping of candidate keys** because there are two composite candidate keys :
  - (Subject, Student)
  - (Teacher, Student)

And student is a common attribute in both the candidate keys.

So we need to normalize the above table to BCNF. For that purpose we must set Teacher to be a candidate key

The decomposition of above takes place as follows

R1(Student, Teacher)

R2(Teacher, Subject)

Now table is in 3NF, as well as in BCNF.

This show that the relation Every relation which is in BCNF is in 3NF but the converse is not true.

5.12 Multivalued Dependencies and Fourth Normal Form

VTU : July-18, Jan.-19, Marks 6

Concept of multivalued dependencies

- A table is said to have multi-valued dependency, if the following conditions are true,

1) For a dependency A ▯ B, if for a single value of A, **multiple values** of B exists, then the table may have multi-values dependency.

2) Also, a table should have **at-least 3 columns** for it to have a multi-valued dependency.

3) And, for a relation R(A,B,C), if there is a multi-valued **dependency between,** A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

- In simple terms, if there are two columns A and B - and for column A if there are multiple values of column B then we say that MVD exists between A and B
- The multivalued dependency is denoted by ↠
- If there exists a multivalued dependency then the table is **not in 4^{th} normal form.**
- **For example :** Consider following table for information about student

Student

sid	Course	Skill
1	C	English
	C++	German
2	Java	English
		French

Here sid = 1 leads to multiple values for courses and skill. Following table shows this

sid	Course	Skill
1	C	English
1	C++	German
1	C	German
1	C++	English
2	Java	English
2	Java	French

Here **sid** and **course** are dependent but the **Course** and **Skill** are independent. The multivalued dependency is denoted as :

sid ↠ Course

sid ↠ Skill

Fourth normal form

Definition : For a table to satisfy the fourth normal form, it should satisfy the following two conditions :

1) It should be in the Boyce-Codd Normal Form(BCNF).

2) And, the table should not have any multi-valued dependency.

For example : Consider following student relation which is not in 4NF as it contains multivalued dependency.

Student table

sid	Course	Skill
1	C	English
1	C++	German
1	C	German
1	C++	English
2	Java	English
2	Java	French

Now to convert the above table to 4NF we must decompose the table into following two tables.

Student_Course Table

Key : (sid,Course)

sid	Course
1	C
1	C++
2	Java

Student_Skill Table

Key : (sid,Skill)

sid	Skill
1	English
1	German
2	English
2	French

Thus the tables are now in 4NF.

University Question

1. *Define multi-valued dependency. Explain fourth normal form with an example*
   VTU : July-18, Marks 4, Jan.-19, Marks 6

5.13 Join Dependencies and Fifth Normal Form

VTU : Jan.-20, Marks 10

- Join decomposition is a further generalization of multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a Join Dependency (JD) exists.
- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.

- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Concept of fifth normal form

The database is said to be in 5NF if -

i) It is in 4^{th} normal form

ii) If we can decompose table further to eliminate redundancy and anomalies and when we rejoin the table we should not be losing the original data or get a new record **(join Dependency Principle).**

The fifth normal form is also called as **project join normal form**

For example - Consider following table

Seller	Company	Product
Rupali	Godrej	Cinthol
Sharda	Dabur	Honey
Sharda	Dabur	Hairoil
Sharda	Dabur	Rosewater
Sunil	Amul	Icecream
Sunil	Britania	Biscuits

Here we assume the keys as{Seller, Company, Product}

The above table has multivalued dependency as

Seller→→{Company, Product}. Hence table is not in 4^{th} normal form. To make the above table in 4^{th} normal form we decompose above table into two tables as

Seller_Company	
Seller	**Company**
Rupali	Godrej
Sharda	Dabur
Sunil	Amul
Sunil	Britania

Seller_Product	
Seller	**Product**
Rupali	Cinthol
Sharda	Honey
Sharda	Hairoil
Sharda	Rosewater
Sunil	Icecream
Sunil	Biscuits

The above table is in 4^{th} normal form as there is no multivalued dependency. But it is not in 5^{th} normal form because if we join the above two table we may get

Seller	Company	Product
Rupali	Godrej	Cinthol
Sharda	Dabur	Honey
Sharda	Dabur	Hairoil
Sharda	Dabur	Rosewater
Sunil	Amul	Icecream
Sunil	Amul	Biscuits
Sunil	Britania	Icecream
Sunil	Britania	Biscuits

Newly added records which are not present in original table

To avoid the above problem we can decompose the tables into three tables as Seller_Company, Seller_Product, and Company Product table

Seller_Company

Seller	Company
Rupali	Godrej
Sharda	Dabur
Sunil	Amul
Sunil	Britania

Seller_Product

Seller	Product
Rupali	Cinthol
Sharda	Honey
Sharda	Hairoil
Sharda	Rosewater
Sunil	Icecream
Sunil	Biscuit

Company_Product

Company	Product
Godrej	Cinthol
Dabur	Honey
Dabur	Hairoil
Dabur	Rosewater
Amul	Icecream
Britania	Biscuit

Thus the table in in 5^{th} normal form.

University Question

1. *Define multivalued dependency and join dependency. Explain 4NF and 5NF with examples.* **VTU : Jan.-20, Marks 10**

Part II : Normalization Algorithms

5.14 Algorithms for Relational Database Schema Design

VTU - July-18, Marks 4

5.14.1 Dependency Preserving Decomposition into 3NF

Algorithm : Relational synthesis algorithm with dependency-preserving.

Input : A universal relation R and a set of functional dependencies F on the attributes of R.

Output : A dependency-preserving decomposition DECOMP = {R1, R2, . . . , Rn} of R that all Ri 's in Decomposition are in 3NF.

Step 1 : Find a minimal cover G for F;

Step 2 : For each left-hand-side X of a functional dependency that appears in G, create a relation schema in decomposition with attributes {X ∪ {A1} ∪ {A2} . . . ∪ {Ak}}, where X → A1, X → A2, . . . , X → Ak are the only dependencies in G with X as left-hand-side (X is the key of this relation);

Step 3 : Place any remaining attributes (that have not been placed in any relation) in a single relation schema to ensure the attribute preservation property.

For example

R = {A, B, C, D, E, H}, F = {AE → BC, B → AD, CD → E, E → CD, A → E}.

Find a dependency-preserving decomposition = {R1, R2, . . . , Rn} of R such that each Ri in decomposition set is in 3NF.

Step 1 : A minimal cover G = {A → B, A → E, B → A, CD → E, E → CD} of F is derived from algorithm

Step 2 : Decompose R to

R1 = {A, B, E} and F1 = {A → B, A → E}

R2 = {B, A} and F2 = {B → A}

R3 = {C, D, E} and F3 = {CD → E}

R4 = {E, C, D} and F4 = {E → CD}

Combine R3 and R4 into one relation schema

R5 = {C, D, E} and F5 = {CD → E, E → CD}.

Step 3 : There is one attribute H in R – (R1 ∪ R2 ∪ R5).

Create another relation schema to contain this attribute.

R6 = {H} and F6 = {}.

All relational schemas in the decomposition {R1, R2, R5, R6} are in 3NF.

Thus the dependency are preserved: $\{F1 \cup F2 \cup F5 \cup F6\}^+ = F^+$

5.14.2 Lossless (Nonadditive) Join Decomposition into BCNF Schemas

Algorithm : Relational decomposition into BCNF relations with lossless join preperty.

Input : A universal relation R and a set of functional dependencies F on the attributes of R.

Step 1 : Set decomposition = {R}

Step 2 : While there is a relation schema Q in Decomposition that is not in BCNF do

```
{
choose a relation schema Q in DECOMP that is not in BCNF;
find a functional dependency X → Y in Q that violates BCNF;
replace Q in DECOMP by two relation schemas (Q − Y ) and (X ∪ Y );
};
```

For example

R = {A, B, C}

F = {AB → C, C → B}

Step 1 : Let decomposition = {{A, B, C}};

Step 2 : {A, B, C} in decomposition that is not in BCNF;

Pick {A, B, C} in decomposition;

Pick C → B in {A, B, C} that violates BCNF;Replace {A, B, C} in decomposition by {A, C} and {B, C};

Step 2 : The decomposition {{A, C}, {B, C}} and both {A, C} and {B, C} are in BCNF;

Therefore, the decomposition {{A, C}, {B, C}} has the lossless join property.

5.14.3 Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas

Algorithm : Relational synthesis into 3NF with dependency preservation and Non-additive (lossless) join property.

Input : A universal relation R and a set of functional dependencies F on the attributes of R.

Output : A dependency-preserving and lossless-join decomposition {R1, R2, . . . , Rn} of R that all Ri's in Decomposition are in 3NF.

Step 1 : Find a minimal cover G for F

Step 2 : For each left-hand-side X of a functional dependency that appears in G create a relation schema in Decomposition with attributes {X ∪ {A1} ∪ {A2} . . . ∪ {Ak}}, where X → A1, X → A2, . . . , X → Ak are the only dependencies in G with X as left-hand-side (X is the key of this relation).

Step 3 : If none of the relation schemas in decomposition contains a key of R, then create one more relation schema in D that contains attributes that form a key of R.

For example :

R = {A, B, C, D, E, H},

F = {AE → BC, B → AD, CD → E, E → CD, A → E}.

Find a dependency-preserving and lossless-join decomposition DECOMP = {R1, R2, . . . , Rn} of R such that each Ri in decomposition is in 3NF.

Step 1 : A minimal cover G = {A → B, A → E, B → A, CD → E, E → CD} for F

Step 2 : Decompose R to

R1 = {A, B, E} and F1 = {A → B, A → E}

R2 = {B, A} and F2 = {B → A}

R3 = {C, D, E} and F3 = {CD → E}

R4 = {E, C, D} and F4 = {E → CD}

Combine R3 and R4 into one relation schema

R5 = {C, D, E} and F5 = {CD → E, E → CD}.

Step 3 : AH and BH are candidate keys of R, and neither of them appear in R1, R2, R5. Create another relation schema R6 = {A, H} and F = {}.

Then, all relational schemas in the decomposition DECOMP = {R1, R2, R5, R6} are in 3NF.

University Question

1. *Define non-additive property of decomposition and write an algorithm for testing for non-additive join property.* **VTU : July-18, Marks 4**

5.15 NULLs, Dangling Tuples and Alternate Relational Designs

NULLS

There are certain problems that are associated with the NULL values when designing the relational database schema. Out of these there are two commonly occurring problems with NULL values and those are -

1) If some tuple have NULL value then it may cause the problem when we join the individual relation in decomposition. For example - Consider two relations - Employee and department.

Suppose that there are two Employees namely XXX and YYY who are newly joined employees. If these employees are not assigned any department yet, and if retrieve the (EmpName,DeptName) values for all the employees using **NATURAL JOIN** on Employee and Department table then the tuples with EmpName = XXX and YYY will not appear. Here an **OUTER JOINT** could solve this problem - tuples with null values on join attributes still appear - but may give you more info than you want.

2) If null values are present in the tuple, then on applying the aggregate functions such as SUM, AVG can cause problems.

Dangling tuple

Concept of dangling tuple : Assume that some entity is represented in more than one relation if tuple exists in one but not another, it is called a dangling tuple.

For example - Consider A table as -

X	Y
101	111
102	222

Now consider table B as

Y	Z
111	100
111	200
333	300

The natural join creates the referencing relation C as follows :

X	Y	Z
101	111	100
101	111	200
102	222	NULL
NULL	333	300

Thus we get the dangling tuples indicating the NULL values. Because these are the tuples which are not present in referenced relation(i.e. parent relation either A or B) but present in the referencing relation C.

The major problem with dangling tuple is that it becomes difficult to specify all the functional dependencies for a database. Algorithms cannot be deterministic.

5.16 Other Dependencies and Normal Forms

1) Inclusion dependencies

The inclusion dependency is a statement in which some columns of a relation are contained in other columns.

An inclusion dependency R.X < S.Y between two sets of attributes (X of schema R and Y of schema S) specifies the constraint that, at any specific time where r is a relation state of R and s is a relation state of S, we must have $\pi X(r) \subseteq \pi Y(s)$. Here X and Y must have the same number of attributes and the domain of corresponding attribute must be compatible.

For Example : DEP.D_MGR_SSN < EMP.SSN i.e., Social Security Number of managers for a department is always a subset of Social Security Number of Employees of a department.

2) Template dependencies

- Templates can be used to provide a general framework for specifying dependencies.
- There are two kinds of templates which are used to specify dependencies : Tuple generating templates and constraint templates.
- The template consists of two things - a list of **hypothesis tuples** followed by a **template conclusion.**
- The **conclusion** is a set of tuples that must exist in the relation if the **hypothesis tuples** are there.

- For **constraint generating templates,** the template conclusion is a **condition** that must hold on **the hypothesis tuples.**
- For example -

Consider relation R={A,B,C,D}

Hypothesis :

a1	b1	c1	d1
a1	b1	c2	d2

Conclusion : c1 = c2 and d1 = d2

This represents the functional dependency AB->CD

3) Domain Key Normal Form (DKNF)

The basic idea behind the DKNF is to specify the normal form that takes into account all the possible dependencies and constraints.

In other words DKNF is a normal form used in database normalization which requires that the database contains no constraints other than domain constraints and key constraints.

i) Domain constraint - which specifies the possible values of some attribute.

E.g. The only colors of cars are blue, white, red, grey.

E.g. The age of a person is between 0 and 150.

ii) Key constraint - which specifies keys of some relation.

Example of domain key constraint : Consider SALES table as follows -

Customer_ID	Product	Price
1111	Soap	10
2222	Toothbrush	20
3333	Toothpaste	30

Suppose we impose the constraint as customer_ID must be an integer value which is > 1100. This is a domain constraint which can be easily applied.

Now consider another constraint as -

1. Customer_ID determines product
2. Product determines price.

This constraint is hard to apply as product is not a key. Hence to satisfy this constraint - we must divide the SALES table into two tables as follows - In one table Customer_ID is a key and in another table Product is a key.

Customer_ID	Product
1111	Soap
2222	Toothbrush
3333	Toothpaste

Product	Price
Soap	10
Toothbrush	20
Toothpaste	30

Here the above mentioned key constraints are satisfied. Thus the relational schema is in domain key normal form.

❑❑❑

MODULE - 5

6 Transaction Processing, Concurrency Control and Recovery Protocols

Syllabus

Transaction Processing : *Introduction to Transaction Processing, Transaction and system concepts, Desirable properties of Transactions, Characterizing schedules based on recoverability, Characterizing schedule based on Serializibility, Transaction support in SQL,* ***Concurrency Control in Databases :*** *Two Phase Locking Techniques for Concurrency Control, Concurrency Control Based on Timestamp Ordering, Multi-version Concurrency control techniques, Validation Concurrency control techniques, Granularity of data items and multiple granularity locking.* ***Introduction to Database Recovery Protocols :*** *Recovery Concepts, NO-UNDO/REDO recovery based on deferred update, Recovery techniques based on immediate update, shadow paging, Database backup and recovery from catastrophic failures.*

Contents

Part I : Transaction Processing

6.1 Introduction to Transaction Processing

VTU : Jan.-19, 20, July-19, Marks 12

6.1.1 Single User Vs Multiuser Systems

- A DBMS system is said to be single user system if at the most one user at a time can use the system.
- A DBMS system is said to be multiuser system if many users can use the system.
- The access to the database system in multiuser system is concurrently.
- Most of the DBMS systems are based on multi user systems.
- Database systems used in banks, insurance agencies, stock exchanges, supermarkets, and many other applications are multiuser systems.

6.1.2 Transaction, Database Items, Read and Write Operations and DBMS Buffers

- **Definition of Transaction :** A transaction can be defined as a **group of tasks** that form a single logical unit. For example - Suppose we want to withdraw ₹ 100 from an account then we will follow following operations :
  1) Check account balance
  2) If sufficient balance is present request for withdrawal.
  3) Get the money
  4) Calculate Balance = Balance – 100
  5) Update account with new balance.

  The above mentioned **four steps** denote **one transaction.**
- There are two modes of concurrency -
  - **Interleaved Processing :** Concurrent execution of processes is interleaved in a single CPU.
  - **Parallel Processing :** Processes are concurrently executed in multiple CPUs.
- Basic transaction processing theory assumes interleaved concurrency.
- A database is a collection of named data items.
- **Granularity or size of data items :** The size of data items can be a field, a record or a whole disk block.
- Basic operations on data item A are -

1. read_item(X)	2. write_item(X)

1. **read_item(X) :** This is a reading operation in which database item named X is read into a programming variable. We can name the program variable as X for simplification.

2. **write_item(X)** : This operation writes the value of program variable X into the database item which is also named as X.

- Basic unit of data transfer from the disk to the computer main memory is one block.
- **read_item(X) command includes the following steps :**

  Step 1 : Find the address of the disk block that contains item X.

  Step 2 : Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).

  Step 3 : Copy item X from the buffer to the program variable named X.
- **write_item(X) command includes the following steps :**

  Step 1 : Find the address of the disk block that contains item X.

  Step 2 : Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).

  Step 3 : Copy item X from the program variable named X into its correct location in the buffer.

  Step 4 : Store the updated block from the buffer back to disk.
- **Example of sample transaction performing read and write operations**

```
read_item(X);
X=X+M;
Write_item(X);
```

- **Transaction Notations :**

The transaction notations focuses on read and write operations. For example – Following are two transactions denoted by T1 and T2

```
T1:b1;r1(X);w1(X);r1(Y);W1(Y);e1;
T2:b2;r2(X);r2(Y);e2
```

The r and w represents the read and write operations. The b1 and b2 represents the beginning and e1 and e2 represents ending of transaction.

6.1.3 Why Concurrency Control is Needed ?

- Concurrent execution of transactions over shared database creates several data integrity and consistency problems - these are,

1) Lost update problem :

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.

For example - Consider following transactions,

1) Salary of employee is read during transaction T_1.
2) Salary of employee is read by another transaction T_2.
3) During transaction T_1, the salary is incremented by ₹ 200
4) During transaction T_2, the salary is incremented by ₹ 500

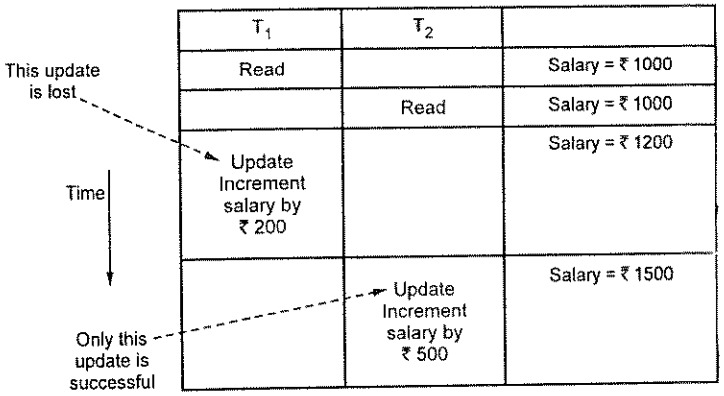


Fig. 6.1.1

The result of the above sequence is that the update made by transaction T_1 is completely lost. Therefore this problem is called as **lost update problem.**

2) Dirty read or uncommited read problem or temporary update :

The dirty read is a situation in which one transaction reads the data immediately after the write operation of previous transaction

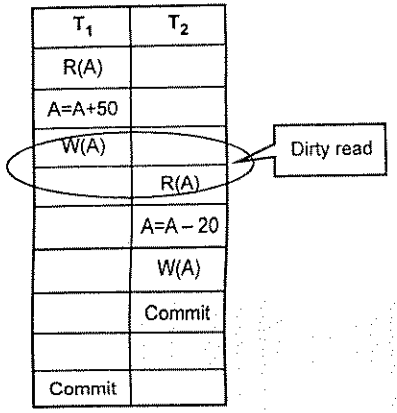
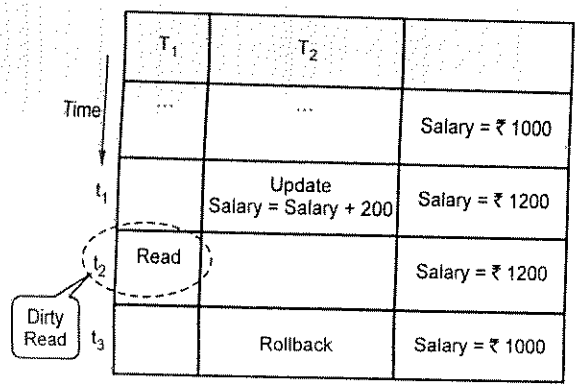


Fig. 6.1.2

For example - Consider following transactions,

Assume initially salary is = ₹ 1000.



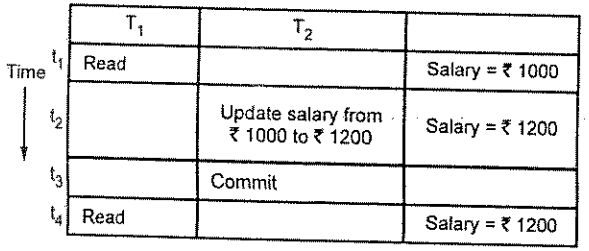
	T_1	T_2	
	...	...	Salary = ₹ 1000
t_1		Update Salary = Salary + 200	Salary = ₹ 1200
t_2	Read		Salary = ₹ 1200
t_3		Rollback	Salary = ₹ 1000

Fig. 6.1.3

1) At the time t_1, the transaction T_2 updates the salary to ₹ 1200.
2) This salary is read at time t_2 by transaction T_1. Obviously it is ₹ 1200.
3) But at the time t_3, the transaction T_2 performs rollback by undoing the changes made by T_1 and T_2 at time t_1 and t_2.
4) Thus the salary again becomes = ₹ 1000. This situation leads to **Dirty Read or Uncommited Read** because here the read made at time t_2 **(immediately after update of another transaction)** becomes a dirty read.

3) Non-repeatable read problem :

This problem is also known as **inconsistent analysis problem**. This problem occurs when a particular transaction sees two different values for the same row within its lifetime. For example -



	T_1	T_2	
t_1	Read		Salary = ₹ 1000
t_2		Update salary from ₹ 1000 to ₹ 1200	Salary = ₹ 1200
t_3		Commit	
t_4	Read		Salary = ₹ 1200

Fig. 6.1.4

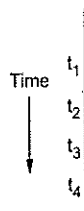
1) At time t1, the transaction T1 reads the salary as ₹ 1000.
2) At time t2 the transaction T2 reads the same salary as ₹ 1000 and updates it to ₹1200.

3) Then at time t3, the transaction T2 gets committed.
4) Now when the transaction T1 reads the same salary at time t4, it gets different value than what it had read at time t1. Now, transaction T1 cannot repeat its reading operation. Thus inconsistent values are obtained.

Hence the name of this problem is non-repeatable read or inconsistent analysis problem.

(4) Incorrect read problem :

This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently. For example -



	T_1	T_2	
t_1	Read		Salary = ₹ 1000
t_2		Read	Salary = ₹ 1000
t_3	Delete salary		No salary
t_4		Read	"Can not find salary"

Fig. 6.1.5

(1) At time t1, the transaction T1 reads the value of salary as ₹ 1000
(2) At time t2, the transaction T2 reads the value of the same salary as ₹ 1000
(3) At time t3, the transaction T1 deletes the variable salary.
(4) Now at time t4, when T2 again reads the salary it gets error. Now transaction T2 can not identify the reason why it is not getting the salary value which is read just few time back.

This problem occurs due to changes in the database and is called **phantom read problem.**

6.1.4 Why Recovery is Needed ?

The important reason why recovery is needed is because some transaction gets failed. Following are the reasons that indicate why recovery is needed -

1) **A computer failure :**
   - The computer failure means some hardware crash or some software error may occur.
   - During execution of transaction if such computer failure occurs then the contents of the computer's internal memory may be lost.

2) **A transaction or system error :**
   - Some operation in the transaction may cause it to fail, such as integer overflow or division by zero.
   - Transaction failure may also occur because of erroneous parameter values or because of a logical programming error.

3) **Local errors or exception conditions detected by the transaction**
   - Certain conditions occur that cancels the transactions. For example – in Banking application, if the insufficient balance is found then the transaction such as 'withdrawal of money' gets canceled.

4) **Concurrency control enforcement**
   - The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock.

5) **Disk failure**
   - Sometimes due to read/write head crash disk blocks may lose their data. This is a severe failure.

6) **Physical problems and catastrophes**
   - It includes various problems such as power failure, fire, overwriting disks or tapes by mistake, mounting wrong tapes by operator, theft and so on.

University Questions

1. *Describe the problems that occur when concurrent execution uncontrolled. Give examples.* **VTU : Jan.-19, Marks 6**
2. *Why concurrency control is needed ? Demonstrate with an example.* **VTU : July-19, Marks 12**
3. *What are anomalies occur due to interleave execution? Explain them with example.* **VTU : Jan.-19, Marks 6**
4. *Describe the database inconsistency problems : Lost update, dirty read and blind write.* **VTU : Jan.-20, Marks 6**

6.2 Transaction and System Concepts **VTU : Jan.-20, Marks 7**

6.2.1 Transaction States

Each transaction has following five states :

1) **Active :** This is the first state of transaction. For example : Insertion, deletion or updation of record is done here. But data is not saved to database.

2) **Partially committed :** When a transaction executes its final operation, it is said to be in a partially committed state.

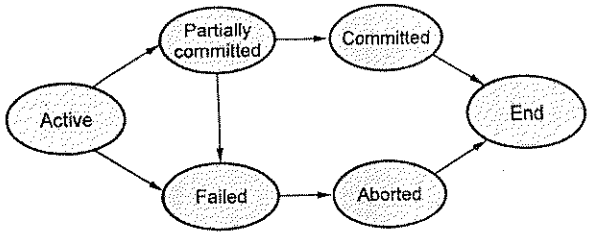


Fig. 6.2.1 Transaction states

3) **Failed :** A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

4) **Aborted :** If a transaction is failed to execute, then the database recovery system will make sure that the database is in its previous consistent state. If not, it brings the database to consistent state by aborting or rolling back the transaction.

5) **Committed :** If a transaction executes all its operations successfully, it is said to be committed. This is the last step of a transaction, if it executes without fail.

Example 6.2.1 *Define a transaction. Then discuss the following with relevant examples : 1. A read only transaction 2. A read write transaction 3. An aborted transaction*

Solution : 1) Read only transaction :

T1
Read(A)
Read(B)
Display(A – B)

2) A read write transaction :

T1
Read(A)
A=A+100
Write(A)

3) An aborted transaction

T1	T2	
Read(A)		Assume A=100
A=A+50		A=150
Write(A)		
	Read(A)	A=150
	A=A+100	A=250
RollBack		A=100 (restore back to original value which is before Transaction T1)
	Write(A)	

University Question

1. *With a neat diagram, explain the various states of a transaction execution.* **VTU : Jan.-20, Marks 7**

6.3 Desirable Properties of Transactions

VTU : July-18, 19, Jan.-19, Marks 4

In a database, each transaction should maintain ACID property to meet the consistency and integrity of the database.

1) Atomicity :

- This property states that each transaction must be considered as a **single unit** and **must be completed fully or not completed at all.**
- No transaction in the database is left half completed.
- Database should be in a state either before the transaction execution or after the transaction execution. It should not be in a state 'executing'.
- For example - In above mentioned withdrawal of money transaction all the five steps must be completed fully or none of the step is completed. Suppose if transaction gets failed after step 3, then the customer will get the money but the balance will not be updated accordingly. The state of database should be either at before ATM withdrawal (i.e customer without withdrawn money) or after ATM withdrawal (i.e. customer with money and account updated). This will make the system in consistent state.

2) Consistency :

- The database must remain in consistent state after performing any transaction.
- **For example :** In ATM withdrawal operation, the balance must be updated appropriately after performing transaction. Thus the database can be in consistent state.

3) Isolation :

- In a database system where **more than one transaction** are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
- No transaction will affect the existence of any other transaction.
- **For example :** If a bank manager is checking the account balance of particular customer, then manager should see the balance either before withdrawing the money or after withdrawing the money. This will make sure that each individual transaction is completed and any other dependent transaction will get the consistent data out of it. Any failure to any transaction will not affect other transaction in this case. Hence it makes all the transactions consistent.

4) Durability :

- The database should be **strong enough** to handle any **system failure.**
- If there is any set of insert/update, then it should be able to handle and commit to the database.
- If there is any failure, the database should be able to recover it to the consistent state.
- **For example :** In ATM withdrawal example, if the system failure happens after customer getting the money then the system should be strong enough to update database with his new balance, after system recovers. For that purpose the system has to keep the log of each transaction and its failure. So when the system recovers, it should be able to know when a system has failed and if there is any pending transaction, then it should be updated to database.

University Question

1. *Discuss ACID properties of a database transaction.* **VTU : July-18, 19, Jan.-19, Marks 4**

6.4 Characterizing Schedules based on Recoverability

The classification of the schedule based on recoverability is -

1. Recoverable Schedule 2. Cascadeless Schedule 3. Strict Schedule

1. **Recoverable Schedule :** This is a kind of schedule where no transaction needs to be rolled back.

Definition : A recoverable schedule is one where, for each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the commit operation of T_j.

For example : Consider following schedule, consider A = 100

T_1	T_2
R(A)	
A=A+50	
W(A)	
	R(A)
	A=A – 20
	W(A)
	Commit
some transaction…	
Commit	

Failure

- The above schedule is inconsistent if failure occurs after the commit of T_2.
- It is because T_2 is **dependable transaction** on T_1. A transaction is said to be dependable if it contains a **dirty read**.
- The dirty read is a situation in which one transaction reads the data immediately after the write operation of previous transaction.

T_1	T_2
R(A)	
A=A+50	
W(A)	
	R(A)
	A=A-20
	W(A)
	Commit
Commit	

Dirty read

- Now if the dependable transaction i.e. T2 is committed first and then failure occurs then if the transaction T1 makes any changes then those changes will not be known to the T2. This leads to non recoverable state of the schedule.
- To make the schedule recoverable we will apply the rule that - commit the independent transaction before any dependable transaction.
- In above example independent transaction is T1, hence we must commit it before the dependable transaction i.e. T2.
- The recoverable schedule will then be -

T1	T2
R(A)	
A=A+50	
W(A)	
	R(A)
	A=A-20
	W(A)
Commit	
	Commit

(2) Cascadeless Schedule

- **Definition :** If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written that data item is committed or aborted, then such a schedule is known as a **cascadeless schedule**.

- The cascadeless schedule allows only committed Read operation. For example :

T1	T2	T3
R(A)		
A=A+50		
W(A)		
Commit		
	R(A)	
	A=A–20	
	W(A)	
	Commit	
		R(A)
		W(A)

- In above schedule at any point if the failure occurs due to commit operation before every read operation of each transaction, the schedule becomes recoverable and atomicity can be maintained.

(3) Strict Schedule

- A schedule in which a transaction can neither read or write an item X until the last transaction that wrote X has committed.

For example

T1	T2
R(A)	
A=A+50	
W(A)	
Commit	
	R(A)
	A=A-20
	W(A)
	Commit

6.5 Characterizing Schedule based on Serializibility

VTU : Jan.-19, 20, July-18, Marks 7

6.5.1 Concept of Schedule

Schedule is an order of multiple transactions executing in concurrent environment. Following Fig. 6.5.1 represents the types of schedules.

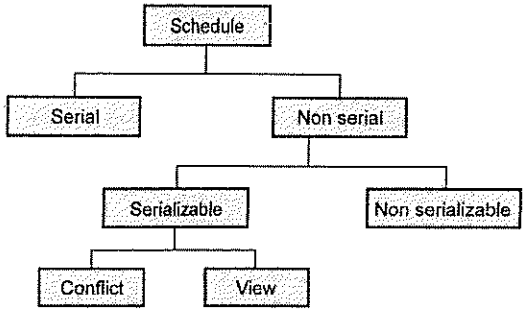


Fig. 6.5.1 : Types of schedule

Serial schedule : The schedule in which the transactions execute one after the other is called **serial schedule**. It is consistent in nature. For example : Consider following two transactions T1 and T2.

T1	T2
R(A)	
W(A)	
R(B)	
W(B)	
	R(A)
	W(A)
	R(B)
	W(B)

All the operations of transaction T1 on data items A and then B executes and then in transaction T2 all the operations on data items A and B execute. The **R stands for read** operation and **W stands for write** operation.

Non serial schedule : The schedule in which operations present within the transaction are intermixed. This may lead to conflicts in the result or inconsistency in the resultant data.

For example -

Consider following two transactions,

T1	T2
R(A)	
W(A)	
	R(A)
	W(B)
R(A)	
W(B)	
	R(B)
	W(B)

The above transaction is said to be non serial which result in inconsistency or conflicts in the data.

6.5.2 Serializability of Scheduling

- When multiple transactions run concurrently, then it may lead to inconsistency of data (i.e. change in the resultant value of data from different transactions).
- Serializability is a concept that helps to identify which non serial schedule and find the transaction equivalent to serial schedule.

For example :

T1	A	B	T2
Initial Value	100	100	
A=A – 10			
W(A)			
B=B+10			
W(B)			
	90	110	
			A=A–10
			W(A)
	80	110	

- In above transactions initially T1 will read the values from database as A=100,

B=100 and modify the values of A and B. But transaction T2 will read the modified value i.e. 90 and will modify it to 80 and perform write operation. Thus at the end of transaction T1 value of A will be 90 but at end of transaction T2 value of A will be 80. Thus conflicts or inconsistency occurs here. This sequence can be converted to a sequence which may give us consistent result. This process is called **serializability**.

Difference between serial schedule and serializable schedule

Sr. No.	Serial schedule	Serializable schedule
1	No concurrency is allowed in serial schedule.	Concurrency is allowed in serializable schedule.
2	In serial schedule, if there are two transactions executing at the same time and no interleaving of operations is permitted, then following can be the possibilities of execution - i) Execute all the operations of transactions T_1 in a sequence and then execute all the operations of transactions T_2 in a sequence. ii) Execute all the operations of transactions T_2 in a sequence and then execute all the operations of transactions T_1 in a sequence.	In serializable schedule, if there are two transactions executing at the same time and interleaving of operations is allowed there can be different possible orders of executing an individual operation of the transactions.

Example of serial schedule

T_1	T_2
Read(A)	
A=A–50	
Write(A)	
Read(B)	
B=B+100	
Write(B)	
	Read(A)
	A=A+10
	Write(A)

Example of serializable schedule

T_1	T_2
Read(A)	
A=A–50	
Write(A)	
	Read(B)
	B=B+100
	Write(B)
Read(B)	
Write(B)	

- **There are two types of serializabilities** : Conflict serializability and view serializability.

6.5.3 Conflict Serializability

Definition : Suppose T_1 and T_2 are two transactions and I_1 and I_2 are the instructions in T_1 and T_2 respectively. Then these two transactions are said to be conflict serializable, if both the instruction access the data item d, and at least one of the instruction is write operation.

What is conflict ? : In the definition **three conditions** are specified for a conflict in conflict serializability -

1) There should be **different transactions**
2) The **operations** must be performed on **same data** items
3) **One of the operation** must be the **Write (W)** operation.

- We can test a given schedule for conflict serializability by constructing a **precedence graph** for the schedule, and by searching for absence of cycles in the graph.
- Predence graph is a directed graph, consisting of G = (V,E) where V is set of vertices and E is set of edges. The set of vertices consists of all the transactions participating in the schedule. The set of edges consists of all edges $T_i \rightarrow T_j$ for which one of three conditions holds :
  1. T_i executes write(Q) before T_j executes read(Q).
  2. T_i executes read(Q) before T_j executes write(Q).
  3. T_i executes write(Q) before T_j executes write(Q).
- A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting**.

Testing for serializability

Following method is used for testing the serializability : To test the conflict serializability we can draw a graph G = (V, E) where V = vertices which represent the number of transactions.

E = edges for conflicting pairs.

Step 1 : Create a node for each transaction.

Step 2 : Find the conflicting pairs (RW, WR, WW) on the same variable (or data item) by different transactions.

Step 3 : Draw edge for the given schedule. Consider following cases

1. T_i executes write(Q) before T_j executes read(Q), then draw edge from T_i to T_j.

2. T_i executes read(Q) before T_j executes write(Q) , then draw edge from T_i to T_j
3. T_i executes write(Q) before T_j executes write(Q), , then draw edge from T_i to T_j

Step 4 : Now, if precedence graph is cyclic then it is a non conflict serializable schedule and if the precedence graph is acyclic then it is conflict serializable schedule.

Example 6.5.1 *Consider the following two transactions and schedule (time goes from top to bottom). Is this schedule conflict-serializable ? Explain why or why not.*

T_1	T_2
R(A)	
W(A)	
	R(A)
	R(B)
R(B)	
W(B)	

Solution :

Step 1 : To check whether the schedule is conflict serializable or not we will check from top to bottom. Thus we will start reading from top to bottom as,

T_1 : R(A) -> T_1 : W(A) ->T_2 : R(A) -> T_2 : R(B) ->T_1 : R(B) -> T_1 : W(B)

Step 2 : We will find **conflicting operations**. Two operations are called as **conflicting operations** if all the following conditions hold true for them -

i) Both the **operations belong to different transactions**.

ii) Both the operations are on **same data item**.

iii) **At least one** of the two operations is a **write operation**.

From above given example in the top to bottom scanning we find the conflict as T_1 : W(A) ->T_2 : R(A).

i) Here note that there are two different transactions T_1 and T_2,

ii) Both work on same data item i.e. A and

iii) One of the operation is write operation.

Step 3 : We will build a precedence graph by drawing one node from each transaction. In above given scenario as there are two transactions, there will be two nodes namely T_1 and T_2.

Fig. 6.5.2

Step 4 : Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from T_1 : W(A) to T_2 : R(A). Hence edge must be from T_1 and T_2.

Fig. 6.5.3

Step 5 : Repeat the step 4 while reading from top to bottom. Finally the **precedence graph** will be as follows

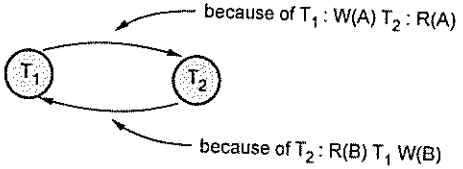


Fig. 6.5.4 : Precedence graph

Step 6 : Check if **any cycle** exists in the graph. Cycle is a path using which we can start from one node and reach to the same node. If the is cycle found then schedule is not conflict serializable. In the step 5 we get a **graph with cycle**, that means given schedule is **not conflict serializable.**

Example 6.5.2 *Check whether following schedule is conflict serializable or not. If it is not conflict serializable then find the serializability order.*

T1	T2	T3
R(A)		
	R(B)	
		R(B)
	W(B)	
W(A)		
		W(A)
	R(A)	
	W(A)	

Solution :

Step 1 : We will read from top to bottom, and build a precedence graph for conflicting entries :

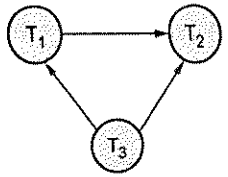


Fig. 6.5.5 Precedence graph

Step 2 : As there is **no cycle** in the precedence graph, the given sequence is **conflict serializable**. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step 3 : A **serializability order** of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting.**

Step 4 : Find the vertex which has no incoming edge which is T1. Finally find the vertex having no outgoing edge which is T2. So in between them is T3. Hence the order will be T1 – T3 – T2.

Example 6.5.3 *Check whether the below schedule is conflict serializable or not.*
{B2,r2(X),b1,r1(X),W1(X),r1(Y),W1(Y),W2(X),e1,C1,e2,C2}

VTU : Jan.-20, Marks 7

Solution : b2 and b1 represents **begin transaction 2** and **begin transaction 1**. Similarly, e1 and e2 represents **end transaction 1** and **end transaction 2.**

We will rewrite the schedule as follows -

T1	T2
	r2(X)
r1(X)	
W1(X)	
r1(Y)	
W1(Y)	
	W2(X)

Step 1 : We will find **conflicting operations**. Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the **operations belong to different transactions**.

ii) Both the operations are on **same data item**.

iii) **At least one** of the two operations is a **write operation**

The conflicting entries are as follows -

T_1	T_2
	$R_2(X)$
$R_1(X)$	
$W_1(X)$	
$R_1(Y)$	
$W_1(Y)$	
	$W_2(X)$

Step 2 : Now we build a precedence graph for conflicting entries.

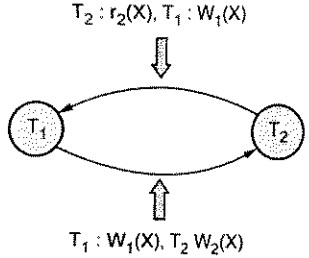


Fig. 6.5.6

As there are two transactions only two nodes are present in the graph.

Step 3 : We get a **graph with cycle**, that means given schedule is **not conflict serializable**.

Example 6.5.4 *Consider the three transactions T1, T2, and T3 and schedules S1 and S2 given below. Determine whether each schedule is serializable or not ? If a schedule is serializable write down the equivalent serial schedule(S).*

T1: R1(x) R1(z);W1(x);

T2: R2(x);R2(y);W2(z);W2(y)

T3:R3(x);R3(y);W3(y);

S1: R1(x);R2(z);R1(z);R3(x);R3(y);W1(x);W3(y);R2(y);W2(z);W2(y);

S2: R1(x);R2(z);R3(x);R1(z);R2(y);R3(y);W1(x);W2(z);W3(y);W2(y);

VTU : Jan.-19, Marks 6

Solution : Step 1 : We will represent the schedule S1 as follows

T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

Step (a) : We will find **conflicting operations**. Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the **operations belong to different transactions**.

ii) Both the operations are on **same data item**.

iii) **At least one** of the two operations is a **write operation**

The conflicting entries are as follows -

T1	T2	T3
R1(x)		
	R2(z)	
R1(z)		
		R3(x)
		R3(y)
W1(x)		
		W3(y)
	R2(y)	
	W2(z)	
	W2(y)	

Step (b) : Now we will draw precedence graph as follows -

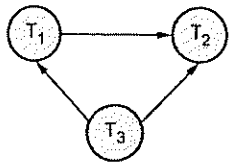


Fig. 6.5.7

As there is **no cycle** in the precedence graph, the given sequence is **conflict serializable**. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step (c) : A **serializability order** of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting**.

Step (d) : Find the vertex which has no incoming edge which is T3. Finally find the vertex having no outgoing edge which is T2. So in between them is T1. Hence the order will be T3- T1-T2

Step 2 : We will represent the schedule S2 as follows -

T1	T2	T3
R1(x)		
	R2(z)	
		R3(x)
R1(z)		
	R2(y)	
		R3(y)
W1(x)		
	W2(z)	
		W3(y)
	W2(y)	

We will find **conflicting operations**. Two operations are called as conflicting operations if all the following conditions hold true for them -

i) Both the **operations belong to different transactions**.

ii) Both the operations are on **same data item**.

iii) **At least one** of the two operations is a **write operation**

The conflicting entries are as follows -

T1	T2	T3
R1(x)		
	R2(z)	
		R3(x)
R1(z)		
	R2(y)	
		R3(y)
W1(x)		
	W2(z)	
		W3(y)
	W2(y)	

Step (b) : Now we will draw precedence graph as follows -

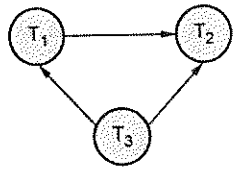


Fig. 6.5.8

As there is **no cycle** in the precedence graph, the given sequence is **conflict serializable**. Hence we can convert this non serial schedule to serial schedule. For that purpose we will follow these steps to find the serializable order.

Step (c) : **A serializability order** of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph. This process is called **topological sorting**.

Step (d) : Find the vertex which has no incoming edge which is T3. Finally find the vertex having no outgoing edge which is T2. So in between them is T1. Hence the order will be T3- T1-T2

6.5.4 View Serializability

- If a given schedule is found to be view equivalent to some serial schedule, then it is called as a **view serializable schedule**.
- **View Equivalent Schedule :** Consider two schedules S1 and S2 consisting of

transactions T1 and T2 respectively, then schedules S1 and S2 are said to be view equivalent schedule if it satisfies following three conditions :

- If transaction T1 reads a data item A from the database initially in schedule S2, then in schedule S2 also, T1 must perform the initial read of the data item X from the database. This is same for all the data items. In other words - the initial reads must be same for all data items.
- If data item A has been updated at last by transaction Ti in schedule S1, then in schedule S2 also, the data item A must be updated at last by transaction Ti.
- If transaction Ti reads a data item that has been updated by the transaction Tj in schedule S1, then in schedule S2 also, transaction Ti must read the same data item that has been updated by transaction Tj. In other words the Write-Read sequence must be same.

Steps to check whether the given schedule is view serializable or not

Step 1 : If the schedule is conflict serializable then it is surely view serializable because conflict serializability is a restricted form of view serializability.

Step 2 : If it is not conflict serializable schedule then check whether there exist any blind write operation. The blind write operation is a write operation without reading a value. If there does not exist any blind write then that means the given schedule is not view serializable. In other words if a blind write exists then that means schedule may or may not be view conflict.

Step 3 : Find the view equivalence schedule

Example 6.5.5 *Consider the following schedules for checking if these are view serializable or not.*

T1	T2	T3
		W(C)
	R(A)	
	W(B)	R(B)
R(C)		
		W(B)
W(B)		

Solution :

i) The initial read operation is performed by T_2 on data item A or by T_1 on data item C. Hence we will begin with T_2 or T_1. We will choose T_2 at the beginning.

ii) The final write is performed by T_1 on the same data item B. Hence T_1 will be at the last position.

iii) The data item C is written by T_3 and then it is read by T_1. Hence T_3 should appear before T_1. Thus we get the order of schedule of view serializability as $T_2 - T_1 - T_3$

Example 6.5.6 *Consider following two transactions :*

```
T1 : read(A)
     read(B)
     if A=0 then B:=B+1;
     write(B)
T2 : read(B);
     read(A);
     if B=0 then A:=A+1;
     write(A)
```

Let consistency requirement be A=0 V B=0 with A=B=0 the initial values.

1) Show that every serial execution involving these two transactions preserves the consistency of the Database ?

2) Show a concurrent execution of T1 and T2 that produces a non serializable schedule ?

3) Is there a concurrent execution of T1 and T2 that produces a serializable schedule ?

Solution : 1) There are two possible executions : $T_1 \rightarrow T_2$ or $T_2 \rightarrow T_1$

Consider case $T_1 \rightarrow T_2$ then

A	B
0	0
0	1
0	1

A∨B = A OR B = F∨T = T. This means consistency is met.

Consider case $T_2 \rightarrow T_1$ then

A	B
0	0
1	0
1	0

A∨B = A OR B = F∨T = T. This means consistency is met.

(2) The concurrent execution means interleaving of transactions T_1 and T_2. It can be

T_1	T_2
R(A)	
	R(B)
	R(A)
R(B)	If B=0 then
If A=0 then	A=A+1
B=B+1	W(A)
W(B)	

This is a non-serializable schedule.

(3) There is no concurrent execution resulting in a serializable schedule.

Solution : i) $r_1(x);r_3(x);w_1(x);r_2(x);w_3(x)$

The r_1 represents the read operation of transaction T_1, w_3 represents the write operation on transaction T_3 and so on. Hence from given sequence the schedule for three transactions can be represented as follows :

T_1	T_2	T_3
$r_1(x)$		
		$r_3(x)$
$w_1(x)$		
	$r_2(x)$	
		$w_3(x)$

Step 1 : We will use the precedence graph method to check the serializability. As there are three transactions, three nodes are created for each transaction.

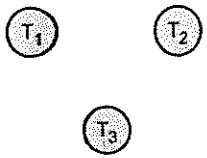


Fig. 6.5.9 Nodes

Step 2 : We will read from top to bottom. Initially we read $r_1(x)$ and keep on moving

bottom in search of write operation. Here all the transactions work on same data item i.e. x. Now we get a write operation in T_3 as $w_3(x)$. Hence the dependency is from T_1 to T_3. Therefore we draw edge from T_1 to T_3.

Similarly, for $r_3(x)$ we get $w_1(x)$ pair. Hence there will be edge from T_3 to T_1. Continuing in this fashion we get the precedence graph as,

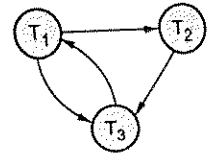


Fig. 6.5.10 Precedence graph

Step 3 : As cycle exists in the above precedence graph, we conclude that it is **not serializable.**

ii) $r_3(x);r_2(x);w_3(x);r_1(x);w_1(x)$

From the given sequence the schedule can be represented as follows :

T_1	T_2	T_3
		$r_3(x)$
	$r_2(x)$	
		$w_3(x)$
$r_1(x)$		
$w_1(x)$		

Step 1 : Read the schedule from top to bottom for pair of operations. For $r_3(x)$ we get $w_1(x)$ air. Hence edge exists from T_3 to T_1 in precedence graph.

There is a pair from $r_2(x) : w_3(x)$. Hence edge exists from T_2 to T_3.

There is a pair from $r_2(x) : w_1(x)$. Hence edge exists from T_2 to T_1.

There is a pair from $w_3(x)$: $r_1(x)$. Hence edge exists from T_3 to T_1.

Step 2 : The precedence graph will then be as follows -

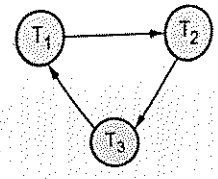


Fig. 6.5.11 Precedence graph

Step 3 : As there is no cycle in the above graph, the given schedule is **serializable.**

Step 4 : The searializability order for consistent schedule will be obtained by applying topological sorting on above drawn precedence graph. This can be achieved as follows,

Sub-Step 1 : Find the node having no incoming edge. We obtain T_2 is such a node. Hence T_2 is at the beginning of the serializability sequence. Now delete T_2. The graph will be,

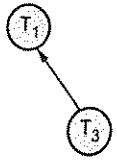


Fig. 6.5.12

Sub-Step 2 : Repeat sub-step 1, we obtain T_3 and T_1 nodes as a sequence.

Thus we obtain the sequence of transactions as T_2, T_3 and T_1. Hence the serializability order is $r_2(x);r_3(x);w_3(x):r_1(x);w_1(x)$

Example 6.5.7 *Consider the following schedules. The actions are listed in the order they are scheduled, and prefixed with the transaction name.*

$S_1 : T_1 : R(X),\ T_2 : R(X),\ T_1 : W(Y),\ T_2 : W(Y)\ T_1 : R(Y),\ T_2 : R(Y)$

$S_2 : T_3 : W(X),\ T_1 : R(X),\ T_1 : W(Y),\ T_2 : R(Z), T_2 : W(Z)\ T_3 : R(Z)$

For each of the schedules, answer the following questions :

i) What is the precedence graph for the schedule ?

ii) Is the schedule conflict-serializable ? If so, what are all the conflict equivalent serial schedules ?

iii) Is the schedule view-serializable ? If so, what are all the view equivalent serial schedules ?

Solution : (i) We will find **conflicting operations**. Two operations are called as conflicting operations if all the following conditions hold true for them -

- Both the **operations belong to different transactions.**
- Both the operations are on **same data item.**
- At least one of the two operations is a write operation

For S_1 : From above given example in the top to bottom scanning we find the conflict as

- $T_1 : W(Y), T_2 : W(Y)$ and
- $T_2 : W(Y), T_1 : R(Y)$

Hence we will build the precedence graph. Draw the edge between conflicting transactions. For example in above given scenario, the conflict occurs while moving from T_1: W(Y) to T_2:W(Y). Hence edge must be from T_1 to T_2. Similarly for second conflict, there will be the edge from T_2 to T_1.

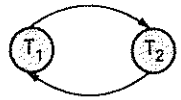


Fig. 6.5.13 Precedence graph for S_1

For S_2 : The conflicts are

- $T_3 : W(X),\ T_1 : R(X)$
- $T_2 : W(Z)\ \ T_3 : R(Z)$

Hence the precedence graph is as follows -

Fig. 6.5.14 Precedence graph for S_2

(ii)

- S_1 is **not conflict-serializable** since the dependency **graph has a cycle.**
- S_2 is conflict-serializable as the dependency graph is acylic. The order T_2-T_3-T_1 is the only equivalent **serial order.**

(iii)

- S_1 is not view serializable.
- S_2 is trivially view-serializable as it is conflict serializable. The only serial order allowed is T_2-T_3-T_1.

University Questions

1. *What is serializability? How can serializibility be ensured? Do you need to restrict concurrent execution of transaction to ensure serializability? Justify your answer.*

VTU - July-18, Marks 6

6.6 Transaction Support in SQL

VTU - July-18, Marks 4

- Using SQL statements various transactions occur with database systems.
- A Single SQL statement is always atomic.
- With SQL there is no explicit **Begin** Transaction statement. The transaction initiation is done implicitly when particular SQL statements are encountered.

- Every transaction must have explicit End statement. It can be wither COMMIT or ROLLBACK.
- Every transaction must have certain characteristics which are specified using SET TRANSACTION statement in SQL.
- Three characteristics are - 1. Access Mode 2. Diagnostic Area Size and 3. Isolation Level

(1) Access Mode :

- In SQL, the access mode can be specified as READ ONLY or READ WRITE.
- The default is READ WRITE unless the isolation level of READ UNCOMITTED is specified, in which case READ ONLY is assumed.

(2) Diagnostic Area Size

- The diagnostic area size n, specifies an integer value n, indicating the number of conditions that can be held simultaneously in the diagnostic area.

(3) Isolation Level

- The transaction should take place in a system in such a way that it is the only transaction that is accessing the resources in a database system at particular instance.
- Isolation levels defines the degree to which a transaction must be isolated from the data modifications made by any other transaction in the database system.
- The isolation level is denoted as <isolation>. There are four levels of transaction isolation defined by SQL -

(i) Serializable :

- This is the **highest isolation level.**
- **Serializable execution is defined** to be an execution of operations **in** which concurrently executing transactions **appears to be serially executing.**
- This is a default isolation level.

(ii) Repeatable Read :

- This is the **most restrictive isolation level.**
- The transaction **holds read locks on all rows** it references.
- It holds **write locks on all rows** it inserts, updates, or deletes.
- Since other transaction cannot read, update or delete these rows, it avoids non repeatable read.

(iii) Read Committed :

- This isolation level allows only **committed data to be read.**
- Thus it does **not allows dirty read** (i.e. one transaction reading of data immediately after written by another transaction).
- The transaction hold a **read or write lock** on the current row, and thus prevent other rows from reading, updating or deleting it.

(iv) Read Uncommitted :

- It is **lowest isolation level.**
- In this level, one transaction may read not yet committed changes made by other transaction.
- This level **allows dirty reads.**
- In this level **transactions are not isolated** from each other.

- The potential problems with lower isolation level are -
  1) **Dirty Read :** The dirty read is a situation in which one transaction reads the data immediately after the write operation of previous transaction.
  2) **Non Repeatable Read :** Allowing another transaction to write a new value between multiple reads of one transaction is called **non repeatable read.**
  3) **Phantoms :** This is a problem in which one of the transaction makes the changes in the database system and due to these changes another transaction can not read the data item which it has read just recently.
- Possible violation of serializability

Isolation Level	Type of Problems		
	Dirty Read	Non-repeatable Read	Phantoms
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITED	No	Yes	Yes
REPEATABL READ	No	No	Yes
SERIALIZABLE	No	No	No

- **Example of Transaction in SQL**

```
EXEC SQL whenever sqlerror go to LABEL;
EXEC SQL SET TRANSACTION
READ WRITE
DIAGNOSTICS SIZE 5
ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT
INTO STUDENT (RollNo,FNAME, LNAME, MARKS)
VALUES (100,'AAA','ZZZ',96);
EXEC SQL COMMIT;
GOTO THE_END;
LABEL: EXEC SQL ROLLBACK;
THE_END:
```

University Question

1. *Explain transaction support in SQL.* **VTU : July-18, Marks 4**

Part II : Concurrency Control in Databases

6.7 Concurrency Control

- One of the fundamental properties of a transaction is **isolation**.
- When several transactions execute concurrently in the database, however, the isolation property may no longer be preserved.
- A database can have multiple transactions running at the same time. This is called **concurrency**.
- To preserve the isolation property, the system must control the interaction among the concurrent transactions; this control is achieved through one of a variety of mechanisms called **concurrency control schemes.**
- **Definition of concurrency control :** A mechanism which ensures that simultaneous execution of more than one transactions does not lead to any database inconsistencies is called **concurrency control mechanism.**
- The concurrency control can be achieved with the help of various protocols such as - Lock based protocol, deadlock handling, multiple granularity, timestamp based protocol, and validation based protocols.

6.8 Two Phase Locking Techniques for Concurrency Control

VTU : July-18, Jan.-19, 20, Marks 6

6.8.1 Why Do We Need Lock ?

- One of the method to ensure the **isolation** property in transactions is to require that data items be accessed in a mutually exclusive manner. That means, while one transaction is accessing a data item, no other transaction can modify that data item.
- The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a **lock on that item.**
- Thus the lock on the operation is required to ensure the isolation of transaction.

6.8.2 Working of Lock

- **Concept of protocol :** The lock based protocol is a mechanism in which there is exclusive use of **locks** on the data item for current transaction.
- **Types of locks :** There are two types of locks used –

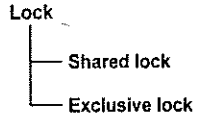


Fig. 6.8.1 Types of locks

i) **Shared lock :** The shared lock is used for reading data items only. It is denoted by Lock-S. This is also called as **read lock.**

ii) **Exclusive lock :** The exclusive lock is used for both read and write operations. It is denoted as Lock-X. This is also called as **write lock.**

- The **compatibility matrix** is used while working on set of locks. The **concurrency control manager** checks the compatibility matrix before granting the lock. If the two modes of transactions are compatible to each other then only the lock will be granted.
- In a set of locks may consists of shared or exclusive locks. Following matrix represents the compatibility between modes of locks.

	S	X
S	T	F
X	F	F

Fig. 6.8.2 Compatibility matrix for locks

Here T stands for True and F stands for False. If the control manager get the compatibility mode as True then it grant the lock otherwise the lock will be denied.

- **For example :** If the transaction T_1 is holding a shared lock in data item A, then the control manager can grant the shared lock to transaction T_2 as compatibility is True.

  But it cannot grant the exclusive lock as the compatibility is false. In simple words if transaction T_1 is reading a data item A then same data item A can be read by another transaction T_2 but cannot be written by another transaction.

- Similarly if an exclusive lock (i.e. lock for read and write operations) is hold on the data item in some transaction then no other transaction can acquire shared or exclusive lock as the compatibility function denotes F. That means of some transaction is writing a data item A then another transaction can not read or write that data item A.
- Hence the **rule of thumb is**

i) **Any number** of transactions can hold **shared lock** on an item.

ii) But **exclusive lock can be hold by only one transaction.**

- **Example of a schedule denoting shared and exclusive locks :** Consider following schedule in which initially A=100. We deduct 50 from A in T1 transaction and Read the data item A in transaction T2. The scenario can be represented with the help of locks and concurrency control manager as follows :

	T_1	T_2	Concurrency control manager
Exclusive lock	Lock-X(A)		
			Grant X(A,T_1) because in T_1 there is write operation.
	R(A)		
	A=A – 50		
	W(A)		
	Unlock(A)		
Shared lock		Lock-S(A)	
			Grant S(A,T_2) because in T_2 there is read operation
		R(A)	
		Unlock(A)	

6.8.3 Two Phase Locking Protocol

- The two phase locking is a protocol in which there are two phases :

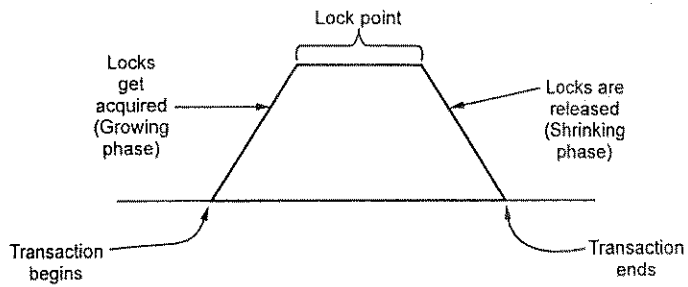
  i) **Growing phase (Locking phase) :** It is a phase in which the transaction may obtain locks but does not release any lock.

  ii) **Shrinking phase (Unlocking phase) :** It is a phase in which the transaction may release the locks but does not obtain any new lock.

- **Lock Point :** The **last lock position** or **first unlock position** is called lock point.
- For example -

Lock(A)
Lock(B)
Lock(C)
....
...
...
Unlock(A)
Unlock(B)
Unlock(C)

Lock Point



Consider following transactions

T_1	T_2
Lock-X(A)	Lock-S(B)
Read(A)	Read(B)
A=A–50	Unlock-S(B)
Write(A)	
Lock-X(B)	
Unlock-X(A)	
B=B+100	Lock-S(A)
Write(B)	Read(A)
Unlock-X(B)	Unlock-S(A)

The important rule for being a two phase locking is - **All lock operations precede all the unlock operations.**

In above transactions T_1 is in two phase locking mode but transaction T_2 is not in two phase locking. Because in T_2, the shared lock is acquired by data item B, then data item B is read and then the lock is released. Again the lock is acquired by data item A , then the data item A is read and the lock is then released. Thus we get lock-unlock-lock-unlock sequence. Clearly this is not possible in two phase locking.

Example 6.8.1 *Prove that two phase locking guarantees serializability.*

Solution :

- Serializability is mainly an issue of handling **write** operation. Because any inconsistency may only be created by **write** operation.
- Multiple reads on a database item can happen parallely.
- 2-Phase locking protocol restricts this unwanted read/write by applying **exclusive lock**.
- Moreover, when there is an **exclusive lock** on an item it will **only be released in shrinking phase**. Due to this restriction there is no chance of getting any inconsistent state.

The serializability using two phase locking can be understood with the help of following example :

Consider two transactions

T_1	T_2
R(A)	
	R(A)
R(B)	
W(B)	

Step 1 : Now we will **apply two phase locking**. That means we will **apply locks in growing** and shrinking phase

T_1	T_2
Lock-S(A)	
R(A)	
	Lock-S(A)
	R(A)
Lock-X(B)	
R(B)	
W(B)	
Unlock-X(B)	
	Unlock-S(A)

Note that above schedule is serializable as it prevents interference between two transactions.

The serializability order can be obtained based on the **lock point**. The lock point is either last lock operation position or first unlock position in the transaction.

The last lock position is in T_1, then it is in T_2. Hence the serializability will be $T_1 \rightarrow T_2$ based on lock points. Hence The **serializability sequence** can be $R_1(A);R_2(A);R_1(B);W_1(B)$

Advantages of two phase locking

(1) It ensures serializability.

Disadvantages of two phase locking protocol

(1) It leads to dealocks.

(2) It leads to cascading rollback.

Problems in two phase locking

The two phase locking protocol leads to two problems - Deadlock and cascading roll back.

1) **Deadlock :** The deadlock problem can not be solved by two phase locking. Deadlock is a situation in which when two or more transactions have got a lock and waiting for another locks currently held by one of the other transactions.

For example

T_1	T_2
Lock-X(A)	Lock-X(B)
Read(A)	Read(B)
A=A–50	B=B+100
Write(A)	Write(B)
Delayed, wait for T_2 to release Lock on B	Delayed, wait for T_1 to release Lock on A

2) Cascading Rollback : Cascading rollback is a situation in which a single transaction failure leads to a series of transaction rollback. For example -

T_1	T_2	T_3
Read(A)		
Read(B)		
C=A+B		
Write(C)		
	Read(C)	
	Write(C)	
		Read(C)

When T_1 writes value of C then only T_2 can read it. And when T_2 writes the value of C then only transaction T_3 can read it. But if the transaction T_1 gets failed then automatically transactions T_2 and T_3 gets failed.

The simple two phase locking does not solve the cascading rollback problem. To solve the problem of cascading Rollback two types of two phase locking mechanisms can be used.

6.8.3.1 Types of Two Phase Locking

1) Strict two phase locking : The strict 2PL protocol is a basic two phase protocol but **all the exclusive mode locks** be **held until the transaction commits.** That means in other words all the **exclusive locks** are **unlocked** only after the **transaction is committed.** That also means that if T_1 has exclusive lock, then T_1 will release the exclusive lock only after commit operation, then only other transaction is allowed to read or write. For example - Consider two transactions

T_1	T_2
W(A)	
	R(A)

If we apply the locks then

T_1	T_2
Lock-X(A)	
W(A)	
Commit	
Unlock(A)	
	Lock-S(A)
	R(A)
	Unlock-S(A)

Thus only after commit operation in T_1, we can unlock the exclusive lock. This ensures the strict serializability.

Thus compared to basic two phase locking protocol, the advantage of strict 2PL protocol is it ensures strict serializability.

2) Rigorous two phase locking : This is stricter two phase locking protocol. Here all locks are to be held until the transaction commits. The transactions can be seriealized in the order in which they commit.

Example - Consider transactions

T_1
R(A)
R(B)
W(B)

If we apply the locks then

T_1
Lock-S(A)
R(A)
Lock-X(B)
R(B)
W(B)
Commit
Unlock(A)
Unlock(B)

Thus the above transaction uses rigorous two phase locking mechanism.

Example 6.8.2 *Consider the following two transactions :*
T_1 : read(A)Read(B);
If A=0 then B=B+1;
Write(B)
T_2 : read(B); read(A)
If B=0 then A=A+1
Write(A)
Add lock and unlock instructions to transactions T_1 and T_2, so that they observe two phase locking protocol. Can the execution of these transactions result in deadlock ?

Solution :

T_1	T_2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Read(B)	Read(A)
if A=0 then B=B+1	if B=0 then A=A+1
Write(B)	Write(A)
Unlock(A)	Unlock(B)
Commit	Commit
Unlock(B)	Unlock(A)

This is lock-unlock instruction sequence help to satisfy the requirements for strict two phase locking for the given transactions.

The execution of these transactions result in deadlock. Consider following partial execution scenario which leads to deadlock.

T_1	T_2
Lock-S(A)	Lock-S(B)
Read(A)	Read(B)
Lock-X(B)	Lock-X(A)
Now it will wait for T_2 to release exclusive lock on A	Now it will wait for T_1 to release exclusive lock on B

University Questions

1. *What is two-phase locking protocol ? How does it guarantee serializability ?*
   VTU : July-18, Jan.-19, Marks 4
2. *What is 2PL? Explain with an example*
   VTU : Jan.-20, Marks 6
3. *How do you detect a deadlock during concurrent transaction execution ?*
   VTU : Jan.-20, Marks 6

6.9 Concurrency Control based on Time Stamp Ordering

VTU : July-18, 19, Marks 6

- The time stamp ordering protocol is a scheme in which the order of transaction is decided in advance based on their timestamps. Thus the schedules are serialized according to their timestamps.
- The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.
- A larger timestamp indicates a more recent transaction or it is also called as **younger transaction** while lesser timestamp indicates **older transaction.**
- Assume a collection of data items that are accessed, with read and write operations, by transactions.
- For each data item X the DBMS maintains the following values : –
  - **RTS(X) :** The timestamp on which object X was last read (by some transaction T_i , i.e., RTS(X):=TS(T_i)) [Note that : RTS stands for Read Time Stamp]

- **WTS(X)** : The timestamp on which object X was last written (by some transaction T_j , i.e., WTS(X):=TS(T_j)) [**Note that** : WTS stands for Write Time Stamp]

- For the following algorithms we use the following assumptions : A data item X in the database has a **RTS(X)** and **WTS(X)**. These are actually the timestamps of read and write operations performed on data item X at latest time.
- A transaction T attempts to perform some action (read or write) on data item X on some timestamp and we call that timestamp as **TS(T)**.
- By timestamp ordering algorithm we need to decide whether transaction T has to be aborted or T can continue execution.

Basic Timestamp Ordering Algorithm

Case 1 (Read) : Transaction T issues **a read(X) operation**

i) If TS(T) < WTS(X), then read(X) is rejected. T has to abort and be rejected.

ii) If WTS(X) ≤ TS(T), then execute read(X) of T and update RTS(X).

Case 2 (Write) : Transaction T issues a **write(X) operation**

i) If TS(T) < RTS(X) or if TS(T) < WTS(X), then write is rejected

ii) If RTS(X) ≤ TS(T) or WTS(X) ≤ TS(T), then execute write(X) of T and update **WTS(X)**.

Example for Case 1 (Read operation)

(i) Suppose we have two transactions T_1 and T_2 with timestamps 10 sec. and 20 sec. respectively.

10 Sec T_1	20 Sec T_2
R(X)	
	W(X)
R(X)	

RTS(X) and WTS(X) is initially = 0

Then RTS(X) = 10, when transaction T_1 executes

After that WTS(X) = 20 when transaction T_2 executes

Now if read operation R(X) occurs on transaction T_1 at TS(T_1) = 10 then

TS(T_1) i.e. 10 <WTS(X) i.e. 20, hence we have to reject second read operation on T_1 i.e.

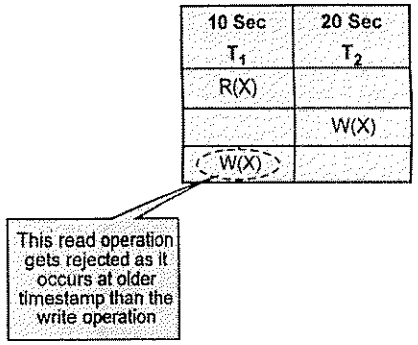


Fig. 6.9.1

ii) Suppose we have two transactions T_1 and T_2 with timestamps 10 sec. and 20 sec. respectively.

10 Sec T_1	20 Sec T_2
W(X)	
	R(X)

RTS(X) and WTS(X) is initially = 0

Then WTS(X) =10 as transaction T_1 executes.

Now if Read operation R(X) occurs on transaction T_2 at TS(T_2) = 20 then

TS(T_2) i.e. 20 >WTS(X) which is 10, hence we accept read operation on T_2. The transaction T_2 will perform read operation and now RTS will be updated as,

RTS(X) = 20

Example for Case 2 (Write Operation)

i) Suppose we have two transactions T_1 and T_2 with timestamps 10 sec. and 20 sec. respectively.

10 Sec. T_1	20 Sec. T_2
R(X)	
	W(X)
W(X)	

RTS(X) and WTS(X) is initially = 0

Then RTS(X) = 10, when transaction T_1 executes

After that WTS(X) = 20 when transaction T_2 executes

Now if write operation W(X) occurs on transaction T_1 at $TS(T_1)$ = 10 then

$TS(T_1)$ i.e. 10 <WTS(X), hence we have to reject second write operation on T_1 i.e.

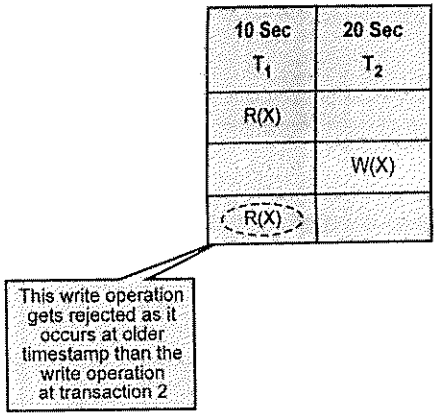


Fig. 6.9.2

ii) Suppose we have two transactions T1 and T2 with timestamps 10 sec. and 20 sec. respectively.

10 Sec T_1	20 Sec T_2
W(X)	
	W(X)

RTS(X) and WTS(X) is initially = 0

Then WTS(X) = 10 as transaction T_1 executes.

Now if write operation W(X) occurs on transaction T_2 at $TS(T_2)$ = 20 then

$TS(T_2)$ i.e. 20 >WTS(X) which is 10, hence we accept write operation on T_2. The transaction T_2 will perform write operation and now WTS will be updated as,

WTS(X) = 20

Advantages and disadvantages of time stamp ordering

Advantages

1) Schedules are serializable
2) No waiting for transaction and hence there is no deadlock situation.

Disadvantages

1) Schedules are not recoverable once transactions occur.
2) Same transaction may be continuously aborted or restarted.

University Question

1. *Discuss Time-stamp ordering protocol for concurrency control.*

VTU : July-18,19, Marks 6

6.10 Deadlock

VTU : Jan.-19, Marks 6

Deadlock is a specific concurrency problem in which two transactions depend on each other for something.

For example - Consider that transaction T_1 holds a lock on some rows of table **A** and needs to update some rows in the **B** table. Simultaneously, transaction T_2 holds locks on some rows in the **B** table and needs to update the rows in the **A** table held by transaction T_1.

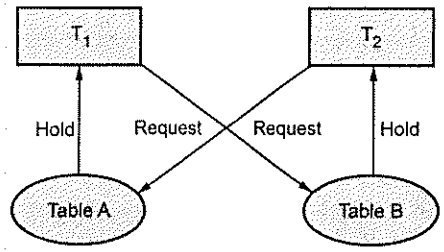


Fig. 6.10.1

Now, the main problem arises. Now transaction T_1 is waiting for T_2 to release its lock and similarly, transaction T_2 is waiting for T_1 to release its lock. All activities come to a halt state and remain at a standstill. This situation is called **deadlock** in DBMS.

Definition : Deadlock can be formally defined as - " A system is in deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set. "

There are **four conditions for a deadlock to occur**

A deadlock may occur if all the following conditions holds true.

1. **Mutual exclusion condition :** There must be at least one resource that cannot be used by more than one process at a time.
2. **Hold and wait condition :** A process that is holding a resource can request for

additional resources that are being held by other processes in the system.

3. **No preemption condition :** A resource cannot be forcibly taken from a process. Only the process can release a resource that is being held by it.
4. **Circular wait condition :** A condition where one process is waiting for a resource that is being held by second process and second process is waiting for third process ....so on and the last process is waiting for the first process. Thus making a circular chain of waiting.

Deadlock can be handled using two techniques -

1. Deadlock prevention 2. Deadlock detection and deadlock recovery

1. Deadlock prevention :

For large database, deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occur. The DBMS analyzes the operations whether they can create deadlock situation or not, If they do, that transaction is never allowed to be executed.

There are two techniques used for deadlock prevention -

(i) Wait-Die :

- In this scheme, if a transaction requests for a resource which is already held with a conflicting lock by another transaction then the DBMS simply checks the timestamp of both transactions. It **allows the older transaction to wait** until the resource is available for execution.
- Suppose there are two transactions T_i and T_j and let TS(T) is a timestamp of any transaction T. If T_2 holds a lock by some other transaction and T_1 is requesting for resources held by T_2 then the following actions are performed by DBMS :
  - Check if $TS(T_i) < TS(T_j)$ - If T_i is the older transaction and T_j has held some resource, then T_i is allowed to wait until the data-item is available for execution. That means if the older transaction is waiting for a resource which is locked by the younger transaction, then the older transaction is allowed to wait for resource until it is available.
  - Check if $TS(T_i) < TS(T_j)$ - If T_i is older transaction and has held some resource and if Tj is waiting for it, then T_j is killed and restarted later with the random delay but with the same timestamp.

Timestamp is a way of assigning priorities to each transaction when it starts. If timestamp is lower then that transaction has higher priority. **That means oldest transaction has highest priority.**

For example -

Let T_1 is a transaction which requests the data item acquired by transaction T_2. Similarly T_3 is a transaction which requests the data item acquired by transaction T_2.

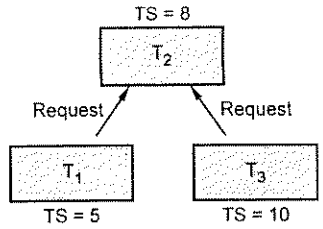


Fig. 6.10.2

Here $TS(T_1)$ i.e. Time stamp of T_1 is less than $TS(T_3)$. In other words T_1 is older than T_3. Hence T_1 is made to **wait** while T_3 is **rolledback**.

(ii) Wound- wait :

- In wound wait scheme, if the older transaction requests for a resource which is held by the younger transaction, then older transaction forces younger one to kill the transaction and release the resource. After some delay, the younger transaction is restarted but with the same timestamp.
- If the older transaction has held a resource which is requested by the younger transaction, then the younger transaction is asked to wait until older releases it.

Suppose T_1 needs a resource held by T_2 and T_3 also needs the resource held by T_2, with $TS(T_1) = 5$, $TS(T_2) = 8$ and $TS(T_3) = 10$, then T_1 being older waits and T_3 being younger dies. After the some delay, the younger transaction is restarted but with the same timestamp.

This ultimately prevents a deadlock to occur.

To summarize

	Wait-Die	**Wound-wait**
Older transaction needs a data item held by younger transaction	Older transaction waits	Younger transaction dies.
Younger transaction needs a data item held by older transaction	Younger transaction dies	Younger transaction dies.

2. Deadlock detection :

- In deadlock detection mechanism, an algorithm that examines the state of the system is invoked periodically to determine whether a deadlock has occurred or not. If deadlock is occurrence is detected, then the system must try to recover from it.
- Deadlock detection is done using wait for graph method.

Wait for graph

- In this method, a graph is created based on the transaction and their lock. If the created **graph has a cycle or closed loop, then there is a deadlock.**
- The wait for the graph is maintained by the system for every transaction which is waiting for some data held by the others. The system keeps checking the graph if there is any cycle in the graph.
- This graph consists of a pair G = (V, E), where V is a set of vertices and E is a set of edges.
- The set of vertices consists of all the transactions in the system.
- When transaction Ti requests a data item currently being held by transaction T_j, then the edge $T_i \rightarrow T_j$ is inserted in the wait-for graph. This edge is removed only when transaction T_j is no longer holding a data item needed by transaction T_i.

For example - Consider following transactions, We will draw a wait for graph for this scenario and check for deadlock.

T_1	T_2
R(A)	
	R(A)
W(A)	
R(B)	
	W(A)
W(B)	

We will use three rules for designing the wait-for graph -

Rule 1 : If T_1 has **Read** operation and then T_2 has **Write** operation then draw an edge $T_1 \rightarrow T_2$.

Rule 2 : If T_1 has **Write** operation and then T_2 has **Read** operation then draw an edge $T_1 \rightarrow T_2$.

Rule 3 : If T_1 has **Write** operation and then T_2 has **Write** operation then draw an edge $T_1 \rightarrow T_2$.

Let us draw wait-for graph

Step 1 : Draw vertices for all the transactions

Fig. 6.10.3

Step 2 : We find the Read-Write pair from two different transactions reading from top to bottom. If such as pair is found then we will add the edges between corresponding directions. For instance -

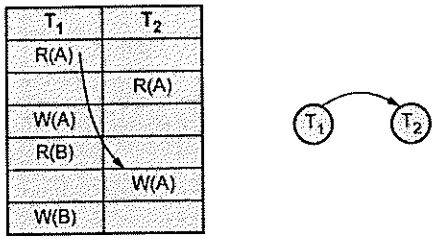


Fig. 6.10.4

Step 3 :

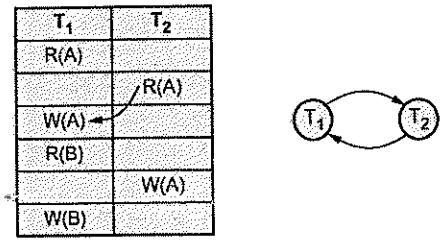


Fig. 6.10.5

As **cycle is detected** in the wait-for graph there is no need to further process. The **deadlock** is present in this transaction scenario.

Example 6.10.1 *Give an example of a scenario where two phase locking leads to deadlock.*

Solution : Following scenario of execution of transactions can result in deadlock.

T_1	T_2
Lock – S (A)	
	Lock – S (B)
	Read (B)
Read (A)	
Lock – X (B)	
	Lock – X (A)

These two instructions cause a deadlock situation.

In above scenario, transaction T_1 makes an exclusive lock on data item B and then transaction T_2 makes an exclusive lock on data item A. Here unless and until T_1 does not give up the lock (i.e. unlock) on B; T_2 cannot read / write it. Similarly unless and until T_2 does not give up the lock on A; T_1 cannot read or write on A.

This is a purely deadlock situation in two phase locking.

Example 6.10.2 *What is deadlock ? Consider following sequence of actions listed in order they are submitted to DBMS*

Sequence: S1:R1(A)W2(B);R1(B);R3(C);w2(c);W4(B);W3(A)

Draw waits-for graph in case of deadlock situation **VTU : Jan.-19, Marks 6**

Solution :

Step 1 : We will rewrite the schedule as,

T1	T2	T3	T4
R1(A)			
	W2(B)		
R1(B)			
		R3(C)	
	W2(C)		
			W4(B)
		W3(A)	

Step 2 :

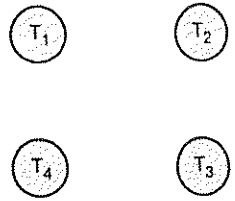
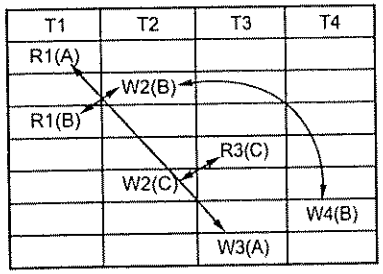


Fig. 6.10.6

As there are four transactions, there will be four nodes.

Step 3 : We find the Read-Write pair from two different transactions reading from top to bottom. If such as pair is found then we will add the edges between corresponding directions. For instance -



Hence the wait-for graph will be,

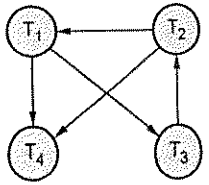


Fig. 6.10.7

The cycle is detected in wait-for graph as T1 → T3 → T2. That means deadlock is present in this transaction scenario.

6.11 Multi-version Concurrency Control Technique

- Normally the serializability can be maintained by either delaying an operation or aborting the transaction that issued the operation. For example, a read operation may be delayed because the appropriate value has not been written yet; or it may be rejected. Sometimes the value that was supposed to read has already been overwritten.

- These difficulties can be avoided if old copies of each data item were kept in system.
- In multi-version concurrency control scheme, each Write(X) operation creates a new version of data item X. When transaction issues Read(X) operation, the concurrency control manager selects one of the version of X to be read.
- The concurrency control scheme must ensure that the version to be read is selected in such a manner that the serializability is ensured.

6.11.1 Multi-version Timestamp Ordering

- Assume X_1 , X_2 , ..., X_n are the version of a data item X created by a write operation of transactions.
- Note that the new version of X_i is created only by a write operation.
- With each X_i a RTS (read timestamp) and a WTS (write timestamp) are associated.
- Let,
  - **RTS(X_i)** : The read timestamp of X_i is the largest of all the timestamps of transactions that have successfully read version X_i .
  - **WTS(X_i)** : The write timestamp of X_i is the largest of all the timestamps of transactions that have successfully written the value of version X_i .
- To ensure serializability following rules are used -

Rule 1 : **If transaction T issues read(X)**, find the version i of X that has the **highest** WTS(X_i) of all versions of X that is also **less than or equal to TS(T)**, then accept the read and update the RTS(X) respectively.

Rule 2 : **If transaction T issues write(X)** then find the version i of X has the highest WTS(Xi) of all versions of X that is also less than or equal to TS(T), and TS(T) < RTS(X_i), then abort and roll-back T;

Advantage :

1) Serializability is maintained.

Disadvantages :

1) More storage is needed.

2) To check unlimited growth of versions, a garbage collection is run periodically.

6.12 Validation Concurrency Control Technique

- The optimistic concurrency control algorithm is basically a validation based protocol.
- It works in three phases -
  - **Read Phase :**
    - In this phase, transaction T is read.
    - The values of various data items are read and stored in temporary variables.
    - All the operations are then performed on temporary variables without updating the actual database.
  - **Validation Phase :**
    - In this phase, the temporary variable value is validated against the actual data in the database and it checked whether the transaction T follows serializability or not.
  - **Write Phase :**
    - If the transaction T is validated then only the temporary results are written to database, otherwise the system rolls back.
- Each phase has following **different timestamps**
  - **Start (T_i) :**
    - It contains the timestamp when Ti starts the execution.
  - **Validation (Ti) :**
    - It contains the timestamp when transaction Ti finishes the read phase and starts its validation phase.
  - **Finish (Ti) :**
    - It contains the timestamp when transaction Ti finishes its write phase.
    - With the help of timestamp in validation phase, this protocol determines if the transaction will commit or rollback. Hence
      TS (T_i) = validation (Ti).
    - The serializability is determined at the validation process, it can't be determined in advance.
    - While executing the transactions, this protocol gives **greater degree of concurrency when there are less number of conflicts**. That is because the serializability order is not pre-decided (validated and then executed) and relatively less transactions will have to be rolled back.

Advantages

1) It is efficient
2) Serializability is maintained.

Disadvantages

1) It has to process all three phases, for each transaction.
2) More storage is needed to maintain data in local variables.

6.13 Granularity of Data Items and Multiple Granularity Locking

VTU : July-18,19, Marks 6

- In database management system there are **data at various levels** depending upon the **data sizes**.
- For example : A database contains a table and a table contains rows and each row contains some data value. Thus different levels of data can be database, tables, records.
- This can be represented in the form of tree. For example.

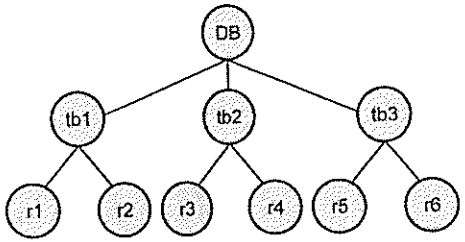


Fig. 6.13.1 Multiple granularity

- Each node **can be locked** individually. When a transaction locks a node, **in** either shared or exclusive mode, the transaction also **implicitly locks** all the descendants of that node in the same lock mode.
- For example - If transaction T1 gets an explicit lock on table tb1 in exclusive mode, then it has an implicit lock in exclusive mode on all the records belonging to that table. It does not need to lock the individual records r1 and r2.
- A **new mode of lock** is introduced along with exclusive and shared locks. This is called **intention mode lock**. Thus in addition to S and X lock modes, there **are** three **additional lock modes** with multiple granularity :
  - **Intention-Shared (IS) :** Explicit locking at a lower level of the tree **but** only with shared locks.

  - **Intention-Exclusive (IX) :** Explicit locking at a lower level with exclusive or shared locks.
  - **Shared and Intention-Exclusive (SIX) :** The sub-tree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive mode locks.
- The **compatibility matrix** for these lock modes are described below :

	IS	IX	S	SIX	X
IS	True	True	True	True	False
IX	True	True	False	False	False
S	True	False	True	False	False
SIX	True	False	False	False	False
X	False	False	False	False	False

- With help of intention lock modes on the transactions the multiple-granularity locking protocol ensures serializability.
- The protocol follows **following rules** -

1) Transaction Ti must follow the lock-compatibility matrix.
2) Transaction Ti must lock the root of the tree first, and it can lock it in any mode.
3) Transaction Ti can lock a node in S or IS mode only if Ti currently has the parent of the node locked in either IX or IS mode.
4) Transaction Ti can lock a node in X, SIX, or IX mode only if Ti currently has the parent of the node locked in either IX or SIX mode.
5) Transaction Ti can lock a node only if Ti has not previously unlocked any node. That means Ti is two phase.
6) Transaction Ti can unlock a node only if Ti currently has none of the children of the node locked.

Advantages

1) It enhances **concurrency**.
2) It ensures **serializability**.
3) It keeps **track of what to lock and how to lock**.
4) This type of locking can be hierarchically represented.
5) It makes it easy to decide which data item to be locked and which data item need to be unlocked.

Part III : Introduction to Database Recovery Protocol

6.14 Recovery Concepts

6.14.1 Purpose of Database Recovery

- The purpose of recovery is to bring the database into the last consistent stage prior to occurrence of failure.
- The recovery must preserve all the ACID properties of transaction. The ACID properties are - Atomicity, consistency, isolation and durability.
- Thus recovery ensures high availability of the database for transaction purpose.
- For example - If the system crashes before the amount transfer from one account to another then either one or both the accounts may have incorrect values. Here the database must be recovered before the modification takes place.

6.14.2 Types of Failure

There are three types of failures that occur commonly -

1) **Transaction failure :** Transactions may fail because of incorrect input, deadlock, incorrect synchronization.
2) **System failure :** System may fail because of addressing error, application error, operating system fault, RAM failure, etc.
3) **Media failure :** Disk head crash, power disruption, etc.

6.14.3 Transaction Log

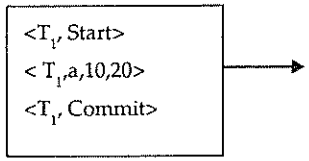
- For recovery from any type of failure data values prior to modification (**BFIM - BeFore Image**) and the new value after modification (**AFIM – AFter Image**) are required. These values and other information is stored in a sequential file called **Transaction log**.
- Log is the most commonly used structure for recording the modifications that as to be made in the actual database. Hence during the recovery procedure a **log file** is maintained.
- A log record maintains four types of operations. Depending upon the type of operations there are four types of log records-
  1. **<Start>** Log record : It is represented as $<T_i$, Start$>$
  2. **<Update>** Log record

  3. **<Commit>** Log record : It is represented as $<T_i$, Commit$>$
  4. **<Abort>** Log record : It is represented as $<T_i$, Abort$>$

- The log contains various fields as shown in following figure. This structure is for <update> operation

Transaction ID(T_i)	Data Item Name	Old Value of Data Item	New Value of Data Item

- For example : The sample log file is



Here 10 represents the old value before commit operation and 20 is the new value that needs to be updated in the database after commit operation

The log must be maintained on the stable storage and the entries in the log file are maintained before actually updating the physical database.

6.14.4 Concept of Data Caching

- Data items to be modified are first stored into database cache by the Cache Manager (CM) and after modification they are flushed (written) to the disk.
- The writing to the disk is controlled by **Modified** and **Pin-Unpin bits**.
  - **Pin-Unpin :** Instructs the operating system not to flush the data item.
  - **Modified :** Indicates the AFIM of the data item.

6.14.5 Data Update

The data can be updated using four ways. These are,

1) **Immediate Update :** In this method, as soon as the data is modified in cache, the data is updated on the disk also.
2) **Deferred Update :** All modified data items in the cache is written either after a transaction ends its execution or after a fixed number of transactions have completed their execution.
3) **Shadow Update :** The modified data items of cache are not overwritten to the disk, but a separate copy of modified data items is maintained at different location on the disk.

4) **In-place Update :** The disk version of the data item is overwritten by the cache version.

6.14.6 UNDO/REDO(Roll-back/Roll-forward)

During transaction execution, the updates are recorded only in the log and in the cache buffers. After the transaction reaches its commit point and the log is force written to disk and the updates are recorded in the database.

In order to maintain the atomicity of transaction, the operations can be redone or undone.

UNDO : This is an operation in which we restore all the old values (BFIM - BeFore Modification Image) onto the disk. This is called **roll-back operation.**

REDO : This is an operation in which all the modified values(AFIM - AFter Modification Image) are restored onto the disk. This is called **roll-forward operation.**

These operations are recorded in the log as they happen.

Difference between UNDO and REDO

Sr. No.	UNDO	REDO
1.	Makes a change go away.	Reproduces a change.
2.	Used for rollback and read consistency.	Used for rolling forward the changes.
3.	Protects the database from inconsistent reads.	Protects from data loss.

6.14.7 Steal/No-steal and Force/No-force

There are possible ways for flushing database cache to database disk :

1. **Steal :** Cache can be flushed before transaction commits.
2. **No-Steal :** Cache cannot be flushed before transaction commit.
3. **Force :** Cache is immediately flushed (forced) to disk.
4. **No-Force :** Cache is deferred until transaction commits.

6.14.8 Write Ahead Logging

- Before a block of data in main memory is output to the database, all log records pertaining to data in that block must have been output to stable storage. This rule is called the **write-ahead logging.**
- This rule is necessary because - In the event of a crash or ROLLBACK, the original content contained in the rollback journal is played back into the database file to revert the database file to its original state.

6.14.9 Check-pointing

- Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk.
- Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.
- The recovery system reads the logs backwards from the end to the last checkpoint.
- Performing a checkpoint consists of the following operations :
  - Suspending executions of transactions temporarily;
  - Writing (force-writing) all modified database buffers of committed transactions out to disk;
  - Writing a checkpoint record to the log; and
  - Writing (force-writing) all log records in main memory out to disk.
- A checkpoint record usually contains additional information, including a list of transactions active at the time of the checkpoint.
- Many recovery methods (including the deferred and immediate update methods) need this information when a transaction is **rolled back**, as all transactions active at the time of the checkpoint and any subsequent ones may need to be redone.
- Since checkpoints cause some loss in performance while they are being taken, their frequency should be reduced if fast recovery is not critical.
- If we need fast recovery check-pointing frequency should be increased. If the amount of stable storage available is less, frequent check-pointing is unavoidable.

6.15 NO-UNDO/REDO Recovery based on Deferred Update

1. Deferred Database Modification :

- In this technique, the database is not updated immediately.
- Only log file is updated on each transaction.
- When the transaction reaches to its commit point, then only the database is physically updated from the log file.
- In this technique, if a transaction fails before reaching to its commit point, it will not have changed database anyway. Hence there is no need for the UNDO operation. The REDO operation is required to record the operations from log file to physical database. Hence deferred database modification technique is also called as **NO UNDO/REDO algorithm.**

  i) UNDO (T_i) : The transaction T_i needs to be undone if the log contains <T_i,Start>

but does not contain <T_i,Commit>. In this phase, it restores the values of all data items updated by T_i to the old values.

ii) REDO (T_i) : The transaction Ti needs to be redone if the log contains both <T_i,Start> and <T_i,Commit>. In this phase, the data item values are set to the new values as per the transaction. After a failure has occurred log record is consulted to determine which transaction need to be redone.

- **For example :**

Consider two transactions T_1 and T_2 as follows :

T_1	T_2
Read (A, a)	Read (C, c)
a = a – 10	c = c – 20
Write (A, a)	Write (C, c)
Read (B, b)	
b = b + 10	
Write (B, b)	

If T_1 and T_2 are executed serially with initial values of A = 100, B = 200 and C = 300, then the state of log and database if crash occurs

a) Just after write (B, b)

b) Just after write (C, c)

c) Just after <T_2, commit>

The result of above three scenarios is as follows :

Initially the log and database will be,

Log	Database
<T_1, Start>	
<T_1, A, 90>	
<T_1, B, 210>	
<T_1, Commit>	
	A = 90
	B = 210
<T_2, Start>	
<T_2, C, 280>	
<T_2, Commit>	
	C = 280

a) Just after write (B, b)

- Just after write operation, no commit record appears in log. Hence no write operation is performed on database. So database retains only old values. Hence A = 100 and B = 200 respectively.
- Thus the system comes back to original position and no redo operation take place.
- The incomplete transaction of T_1 can be deleted from log.

b) Just after write (C, c)

- The state of log records is as follows :
- Note that crash occurs before T_2 commits. At this point T_1 is completed successfully, so new values of A and B are written from log to database. But as T_2 is not committed, there is no redo (T_2) and the incomplete transaction T_2 can be deleted from log.
- The redo (T_1) is done as < T_1, commit> gets executed. Therefore A = 90, B = 210 and C = 300 are the values for database.

c) Just after < T_2, commit>

The log records are as follows :

< T_1, Start>

< T_1, A, 90>

< T_1, B, 210>

< T_1, Commit>

< T_2, Start>

< T_2, 6, 280>

< T_2, Commit>

←—— **Crash occurs here**

Clearly both T_1 and T_2 reached at commit point and then crash occurs. So both redo (T_1) and redo (T_2) are done and updated values will be A = 90, B = 210, C = 280.

University Question

1. *Discuss the UNDO and REDO operations and recovery techniques that use each.*

VTU : July-18,19, Marks 6

6.16 Recovery Technique based on Immediate Update

In this technique, the database is updated during the execution of transaction even before it reaches to its commit point.

If the transaction gets failed before it reaches to its commit point, then the **ROLLBACK Operation** needs to be done to bring the database to its earlier consistent state. That means the effects of operations need to be undone on the database. For that purpose both Redo and Undo operations are both required during the recovery. This technique is known as **UNDO/ REDO** technique.

For example : Consider two transaction T_1 and T_2 as follows :

T_1	T_2
Read(A,a)	Read(C, c)
a = a – 10	c = c – 20
Write(A, a)	Write(C, c)
Read(B, b)	
b = b + 10	
Write(B, b)	

Here T_1 and T_2 are executed serially. **Initially** A = 100,B = 200 and C = 300.

If the crash occurs

i) Just after Write(B, b) ii) Just after Write(C, c)

iii) Just after <T_2,Commit>

Then using the immediate database modification approach the result of above three scenarios can be elaborated as follows :

The contents of **log** and **database** is as follows :

Log	Database
<T_1,Start>	
<T_1,A,100,90>	A = 90
<T_1,B,200,210>	B = 210
<T_1,Commit>	
<T_2,Start>	
<T_2,C,300,280>	C = 280
<T_2,Commit>	

The recovery scheme uses two recovery techniques -

i) **UNDO (T_i) :** The transaction T_i needs to be undone if the log contains <T_i,Start> but does not contain <T_i,Commit>. In this phase, it restores the values of all data items updated by T_i to the old values.

ii) **REDO (T_i) :** The transaction T_i needs to be redone if the log contains both <T_i,Start> and <T_i,Commit>. In this phase, the data item values are set to the new values as per the transaction. After a failure has occurred log record is consulted to determine which transaction need to be redone.

a) **Just after Write (B, b) :** When system comes back from this crash, it sees that there is <T_1, Start> but no <T_1, Commit>. Hence T_1 must be undone. That means old values of A and B are restored. Thus old values of A and B are taken from log and both the transaction T_1 and T_2 are re-executed.

b) **Just after Write (C, c) :** Here both the redo and undo operations will occur.

c) **Undo :** When system comes back from this crash, it sees that there is <T_2, Start> but no <T_2, Commit>. Hence T_2 must be undone. That means old values of C is restored.

Thus old value of C is taken from log and the transaction T_2 is re-executed.

d) **Redo :** The transaction T_1 must be done as log contains both the <T_1, Start> and <T_1, Commit>

So A = 90, B = 210 and C = 300

e) **Just after <T_2, Commit> :** When the system comes back from this crash, it sees that there are two transaction T_1 and T_2 with both start and commit points. That means T_1 and T_2 need to be redone. So A = 90, B = 210 and C = 280.

6.17 Shadow Paging

VTU : July-19, Jan.-20, Marks 8

- Shadow paging is a recovery scheme in which database is considered to be made up of number of fixed size disk pages.
- A directory or a page table is constructed with n number of pages where each i^{th} page points to the i^{th} database page on the disk. (Refer Fig. 6.17.1)

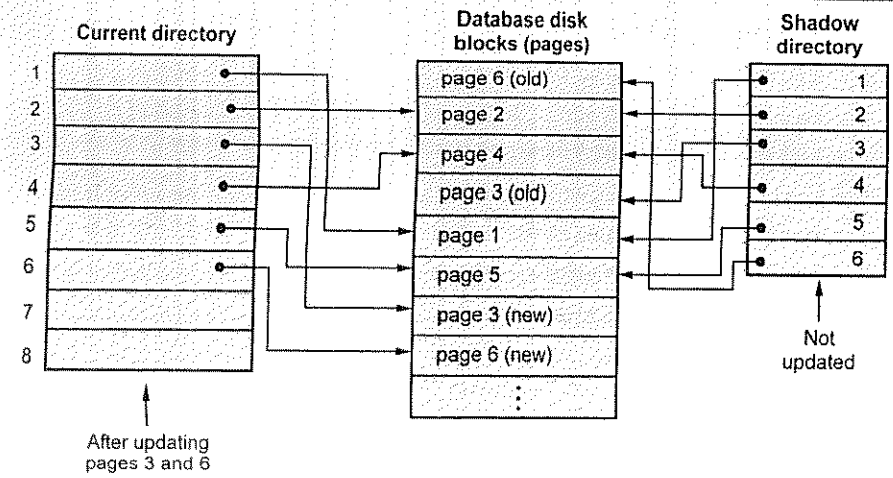


Fig. 6.17.1 Demonstration of Shadow Paging

- The directory can be kept in the main memory.
- When a transaction begins executing, the **current directory**-whose entries point to the most recent or current database pages on disk-is copied into a another directory called **shadow directory**.
- The shadow directory is then saved on **disk** while the current directory is used by the transaction.
- During the execution of transaction, the shadow directory is never modified.
- When a write operation is to be performed then the **new copy** of modified database page is created but the old copy of database page is never overwritten. This newly created database page is written somewhere else.
- The current directory will point to newly modified web page and the shadow page directory will point to the old web page entries of database disk.
- When the failure occurs then the modified database pages and current directory is discarded.
- The state of database before the failure occurs is now available through the shadow directory and this state can be recovered using shadow directory pages.
- This technique does not require any UNDO/REDO operation.

University Questions

1. *Explain how shadow paging helps to cover from transaction failure.* **VTU : July-19, Marks 5**
2. *Explain the various database recovery techniques with examples.* **VTU : Jan.-20, Marks 8**

6.18 Database Backup and Recovery from Catastrophic Failures

- A catastrophic failure is a sudden and total failure from which recovery is very complicated.
- The recovery manager of a DBMS must be equipped to handle catastrophic failures such as **disk crashes.**
- The **main technique** used to handle such crashes is that of **database backup.** That means maintain the backups periodically.
- The whole database and the log are periodically copied onto a cheap storage medium such as **magnetic tapes.**
- In case of a catastrophic system failure, the latest backup copy can be reloaded from the tape to the disk, and the system can be restarted.

❑❑❑

Notes

July - 2018 Database Management System [15CS53] Semester - V (CSE/ISE)	VTU Solved Paper (CBCS Scheme)

Time : 3 Hours] [Max. Marks : 80

Note : Answer any FIVE full questions, choosing ONE full question from each module.

Module - 1

Q.1 a) *Discuss the main characteristics of the database approach and how it differs from traditional file system.* **(Refer section 1.3)** [4]

b) *Describe the three - schema architecture. Why do we need mapping among schema levels ?* **(Refer section 1.7)** [4]

c) *Discuss various components of a DBMS, with a neat diagram.* **(Refer section 1.9)** [8]

OR

Q.2 a) *Define an entity and attribute. Explain the different types of attributes that occur in an ER - diagram model, with an example.* **(Refer section 1.11)** [6]

b) *Draw an ER diagram of an airline reservation system, taking into account at least five entities. Indicate all keys, constraints and assumptions that are made.* **(Refer example 1.14.1)** [10]

Module - 2

Q.3 a) *Explain the data types available for attribute specification in SQL.* **(Refer section 3.1.3)** [4]

b) *Explain briefly violations in entity integrity constraint, key and referential integrity constraints, with example.* **(Refer section 2.3)** [6]

c) *Consider the following RESORT database*

RESORT (resortno, resortname, resorttype, resortaddr, resortcity, numsuite)

SUITE(suiteno, resortno, suiteprice)

RESERVATION (reservationno. resortno, visitorno, checkin, checkout. totalvisitor, suiteno)

VISITOR (visitorno, firstname, lastname, visitoraddr)

i) Write the SQL to list full details of all the resorts on Los Angeles,

ii) Write the SQL to list full details of all the resorts having number of suites more than 30.

iii) Write the SQL to list visitors in ascending order by firstname.

(Refer example 4.2.27) [6]

OR

Q.4 a) *Explain how constraints are specified in SQL during table creation, with suitable example.* **(Refer section 3.2)** [4]

b) *Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course.*

STUDENT (SSn, Name, Major, bdate)

COURSE (Courseno, Cname, dept),

ENROLL (SSn, Courseno, Quarter, grade)

BOOK_ADOPTION (Courseno, Quarter, book_isbn)

TEXT (book_isbn, book_title, Publisher, Author)

Write the following queries in relational algebra on the database schema :

i) List the number of courses taken by all students named John Smith in winter 2009 (i.e. Quarter = W09).

ii) Produce a list of text books (include courseno, book_isbn, book_title) for courses offered by the 'CS' department that have used more than two books.

iii) List any department that has all its adopted books published by 'Pearson' publishing. **(Refer example 4.4.1)** [6]

c) *Give an example of mapping of generalization or specialization into relation schemas.* **(Refer section 2.10)** [6]

Module - 3

Q.5 a) *Discuss how each of the following constructs is used in SQL and discuss the various options for each construct :*

i) Nested queries ii) Aggregate functions iii) Triggers iv) Views and their updatability v) Schema change statements vi) Group by and having clause.

(Refer section 4.5) [6]

b) *Draw and explain 3 - tier architecture and technology relevant to each tier. Write the advantages of 3 - tier architecture.* **(Refer section 4.10)** [6]

c) *What is CGI ? Why was CGI introduced ? What are the disadvantages of an architecture using CGI scripts ?* **(Refer section 4.12)** [4]

OR

Q.6 a) *What is dynamic SQL and how is it different from embedded SQL ?* **(Refer section 4.6)** [4]

b) *What is SQLJ and how is it different from JDBC ?* **(Refer section 4.8)** [4]

c) *Consider the following company database :*

EMP (Name, Ssn , Salary. Superssn, dno)

DEPT (dnum. dname, mgrssn)

DEPT LOC (dnum, dlocation)

PROJECT (Pname, Pnumber, Plocation, dnum)

WORKS_ON (Essn, Pno, Hours)

DEPENDENT (Esgn. dept_name, sex)

Write SQL queries for the following :

i) Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.

ii) Retrieve the names of employees who make atleast 10,000 more than the employee who is paid the least in the company.

iii) A view that has the employee name, supervisor name and employee salary for each employee who works in the 'Research' department,

iv) A view that has the project name, controlling department name, number of employees and total hours worked per week on the project for each project with more than one employee working on it. **(Refer example 4.4.1)** [8]

Module - 4

Q.7 a) *Discuss insertions, deletion and modification anomalies. Why are they considered bad ? Illustrate with examples.* **(Refer section 5.1)** [4]

b) *Define multivalued dependency. Explain fourth normal form, with an example.* **(Refer section 5.12)** [6]

c) *Consider the universal relation R = {A, B, C, D, E. F, G, H, I, J} and the set of functional dependencies F = {{A, B} → {C}, {A} → {D, E}, {B} → {F}, {F} → {G, H}, {D} → {I, J}}, What is key of R ? Decompose R into 2NF and then 3NF relations.* **(Refer section 5.10.1)** [6]

OR

Q.8 a) *Define Non - additive join property of a decomposition and write an algorithm of testing for non - additive join property.* **(Refer section 5.14)** [4]

b) *A relation R (A, C, D, E, H) satisfies the following FDs A → C, AC → D, E → AD, E → H. Find the canonical cover for this set of FD's.* **(Refer example 5.3.5)** [6]

c) *Consider two set of functional dependencies :*

F = {A → C, AC → D, E → AD, E → H} and G = {A → CD, E → AH}. Are they equivalent ? **(Refer section 5.3.2)** [6]

Module - 5

Q.9 a) *Discuss ACID properties of a database transaction.* **(Refer section 6.3)** [4]

b) *Explain transaction support in SQL.* **(Refer section 6.6)** [6]

c) *Discuss the UNDO and REDO operations and the recovery techniques that use each.* **(Refer section 6.13)** [6]

OR

Q.10 a) *What is two - phase locking protocol ? How does it guarantee serializability ?* **(Refer section 6.8)** [4]

b) *What is serializability ? How can serializability be ensured ? Do you need to restrict concurrent execution of transaction to ensure serializability ? Justify your answer.* **(Refer section 6.5)** [6]

c) *Discuss the time - stamp ordering protocol for concurrency control.* **(Refer section 6.9)** [6]

January - 2019 **Database Management System** **[15CS53]** **Semester - V (CSE/ISE)**	**VTU** **Solved Paper** **(CBCS Scheme)**

Time : 3 Hours] **[Maximum Marks : 80**

Module - 1

Q.1 a) *What are the responsibilities of the DBA and database designer ?* **(Refer section 1.4)** [6]

b) *With neat diagram, explain 'three tier schema architecture".* **(Refer section 1.7)** [5]

c) *Discuss the different types of user friendly interfaces and the types of user who typically use each.* **(Refer section 1.9)** [5]

OR

Q.2 a) *Explain with block diagram the different phases of database design.* **(Refer section 1.10)** [8]

b) *Draw an ER-diagram of movie database. Assume your own, entities (minimum 4) attributes and relationships.* **(Refer example 1.14.2)** [8]

Module - 2

Q.3 a) *Discuss the characteristics of relations.* **(Refer section 2.1)** [6]

b) *Outline the steps to convert the basic ER model to relational database schema.* **(Refer section 2.10)** [6]

c) *Define the following* [4]

i) Relation state ii) Relation schema iii) Arity iv) Domain

(Refer section 2.1)

OR

Q.4 a) *Discuss the various types of set theory operations with example.* **(Refer section 2.7)** [8]

b) *Consider the two tables, show the results of the following :*

T_1		
A	B	C
10	a	5
15	b	8
25	a	6

T_2		
P	Q	R
10	b	6
25	c	3
10	b	5

i) $T_1 \rtimes_{T_1.B = T_2.Q} T_2$

ii) $T_1 \rtimes_{T_1.A = T_2.P} T_2$

iii) $T_1 \bowtie_{(T_1.A = T_2.P) \text{ AND } (T_1.C = T_2.R)} T_2$

iv) $T_1 - T_2$ **(Refer example 2.9.14)** [8]

Module - 3

Q.5 a) *How does SQL implement the entity integrity constraints of the relational data model ? Explain with an example.* **(Refer section 3.2)** [4]

b) *Discuss : i) Shared variables ii) Communication variables.* **[6]**

Ans. :

i) Shared variables are those variables which are used in both C program and in embedded SQL statements. They are preceded by colon.

ii) Communication variables are those variables which are used to communicate exception or error conditions. SQLCODE and SQLSTATE are communication variables.

c) *Explain with examples in SQL :* **[6]**

i) Drop command ii) Delete command iii) Update command **(Refer section 4.5)**

OR

Q.6 a) *With program segment, explain retrieving of tuples with embedded SQL in C.* **(Refer section 4.6)** **[6]**

b) *Consider the following tables :* **[10]**

works (Pname, Cname, Salary)

lives (Pname, Street, City)

located-In (Cname, City)

Write the following queries in SQL :

i) List the names of the people who work for the company 'Wipro' along with the cities they live in.

ii) Find the names of the persons who do not work for 'Infosys'.

iii) Find the people whose salaries are more than that of all of the 'oracle' employees.

iv) Find the persons who works and lives in the same city. **(Refer example 4.2.28)**

Module - 4

Q.7 a) *What do you mean by closure of attribute ? Write an algorithm to find closure of attribute.* **(Refer section 5.2)** **[6]**

b) *Explain any two informal quality measures employed for a relation schema design.* **(Refer section 5.1)** **[4]**

c) *Given below are two sets of FDs for a relation R (A, B, C, D, E). Are they equivalent ?* **[6]**

i) $A \rightarrow B, AB \rightarrow C, D \rightarrow AC, D \rightarrow E$ *ii)* $A \rightarrow BC, D \rightarrow AE$

(Refer example 5.3.3)

OR

Q.8 a) *What do you mean by multi valued dependency ? Explain the 4NF with example.* **(Refer section 5.12)** **[6]**

b) *Suggest and explain three different techniques to achieve 1NF. using suitable example.* **(Refer section 5.8)** **[4]**

c) *Consider the following, relation for CARSALE (CAR-NO, Date-Sold, Salesman No, Commission, Discount)* **[6]**

Assume a car can be sold by multiple salesman and hence primary key is

{CAR_No, Salesman_No},

Additional dependencies are

Date_Sold $\rightarrow$ *Discount*

Salesman_No $\rightarrow$ *Commission*

i) Is this relation in 1NF, 2NF or 3NF ? Why or why not ?

ii) How would you normalize this completely ? **(Refer section 5.10.5)**

Module - 5

Q.9 a) *Discuss the ACID properties of a transaction.* **(Refer section 6.3)** **[4]**

b) *What are the anomalies occur due to interleave execution ? Explain them with example.* **(Refer section 6.1)** **[6]**

c) *Consider the three transactions* T_1, T_2 *and* T_3 *and schedules* S_1 *and* S_2 *given below. Determine whether each schedule is serializable or not ? If a schedule is serializable write down the equivalent serial schedule (S).*

$T_1 : R_1(x); R_1(z); W_1(x);$

$T_2 : R_2(x); R_2(y); W_2(z); W_2(y);$

$T_3 : R_3(x); R_3(y); W_3(y);$

$S_1 : R_1(x); R_2(z); R_1(z); R_3(x); R_3(y); W_1(x); W_3(y); R_2(y); W_2(z); W_2(y);$

$S_2 : R_1(x); R_2(z); R_3(x); R_1(z); R_2(y); R_3(y); W_1(x); W_2(z); W_3(y); W_2(y);$

(Refer example 6.5.4) **[6]**

OR

Q.10 a) *Describe the problems that occur when concurrent execution uncontrolled. Give examples.* **(Refer section 6.1)** **[6]**

b) *What is two phase locking ? Describe with the help of an example.* **(Refer section 6.6)** **[4]**

c) *What is deadlock ? Consider the following sequences of actions listed in the order they are submitted to the DBMS.* **[6]**

Sequence $S_1 : R_1(A); W_2(B); R_1(B); R_3(C); W_2(C); W_4(B); W_3(A)$

Draw waits for graph in case of deadlock situation. **(Refer example 6.10.2)**

July - 2019
Database Management System
[15CS53]
Semester - V (CSE/ISE)

VTU Solved Paper (CBCS Scheme)

Time : 3 Hours] [Max. Marks : 80

Module - 1

Q.1 a) *Define DBMS. Discuss the advantages of DBMS over the traditional file system.* **(Refer section 1.3)** [8]

b) *Explain the component modulus of DBMS and their interaction, with the help of a diagram.* **(Refer section 1.9)** [8]

OR

Q.2 a) *Define the following with example :* [8]
i) Weak entity type ii) Participation constraints
iii) Cardinality ratio iv) Recursive relationship **(Refer sections 1.11 and 1.12)**

b) *Draw an ER Diagram of Banking system taking into account at least five entities, indicate all keys, constraints and assumptions that are made.* **(Refer example 1.14.3)** [8]

Module - 2

Q.3 a) *What is meant by integrity constraint ? Explain the importance of referential integrity constraint. How referential integrity constraints is implemented in SQL ?* **(Refer section 3.2)** [8]

b) *Consider the following Movie database :*
Movie (Title, director, Myear, Rating)
Actors (Actor, Aage)
Acts (Actor, title)
Directors (Director, dage)
Write the following queries in relational algebra on the database given :
i) Find movies made by "Hanson" after 1997.
ii) Find all actors and directors.
iii) Find "Coen's" movie with "Mc Dormand".
iv) Find (director, actor) pairs where the director is younger than the actor.
(Refer example 2.9.13) [8]

OR

Q.4 a) *Discuss insertion, deletion and modification anomalies. Why are they considered bad ? Illustrate with an example.* **(Refer section 5.1)** [8]

b) *Write the SQL queries for the following relational schema ;*
Sailors (Sid, Sname, Rating, Age)
Boats (Bid, Bname, color)
Reserve (Sid, Bid, Day)
i) Retrieve the Sailor's name who have reserved red and green boat.
ii) Retrieve the number of boats which are not reserved.
iii) Retrieve the Sailor's name who have reserved boat number 103.
iv) Retrieve the Sailor's name who have reserved all boats.
(Refer example 4.2.29) [8]

Module - 3

Q.5 a) *How are triggers and assertions defined in SQL ? Explain.* **(Refer section 4.3)** [8]

b) *How are views created and dropped ? Explain how the views are implemented and updated.* **(Refer section 4.4)** [8]

OR

Q.6 a) *Explain the single-tier and client-server architecture, with a neat diagram.* **(Refer section 4.10.1)** [8]

b) *Explain the following :*
i) Embedded SQL ii) Database stored procedure **(Refer section 4.9)** [8]

Module - 4

Q.7 a) *Which normal form is based on the concept of transitive functional dependency ? Explain the same with an example.* **(Refer section 5.10)** [8]

b) *What is the need for normalization ? Consider the relation :*
Emp – proj = {SSn, Pnumber, Hours, Ename, Pname, Plocation}.
Assume {SSn, Pnumber} as primary key.
The dependencies are :
{SSn, Pnumber} → Hours
SSn → Ename
Pnumber → {Pname, Plocation}
Normalize the above relation to 3NF. **(Refer example 5.10.4)** [8]

OR

Q.8 a) *What is functional dependency ? Find the minimal cover using the minimal cover algorithm for the following functional dependency.*
$F = \{AB \rightarrow D, B \rightarrow C, AE \rightarrow B, A \rightarrow D, D \rightarrow EF\}$ **(Refer section 5.3.6)** [8]

b) *Consider two sets of functional dependency.*
$F = \{A \rightarrow C, AC \rightarrow D, E \rightarrow AD, E \rightarrow H\}$ *and* $G = \{A \rightarrow CD, E \rightarrow AH\}$.
Are they equivalent ? **(Refer section 5.3.2)** [8]

Module - 5

Q.9 a) *Discuss the ACID properties of a database transaction.* **(Refer section 6.3)** [4]

b) *Why Concurrency control is needed ? Demonstrate with an example.* **(Refer section 6.1)** [12]

OR

Q.10 a) *Discuss the UNDO and REDO operations and the recovery techniques that use each.* **(Refer section 6.13)** [6]

b) *Discuss the time-stamp ordering protocol for concurrency control.* **(Refer section 6.9)** [5]

c) *Explain how shadow paging helps to recover from transaction failure.* **(Refer section 6.15)** [5]

January - 2020 **Database Management System** **[17CS53]** **Semester - V (CSE/ISE)**	**VTU** **Solved Paper** **(CBCS Scheme)**

Time : 3 Hours] **[Maximum Marks : 100**

Note : Answer any FIVE full questions, choosing ONE full question from each module.

Module - 1

Q.1 a) *Compare DBMS and early file systems, bringing out the major advantages of the database approach.* **(Refer section 1.3)** [6]

b) *With a neat block diagram, explain the architecture of a typical DBMS.* **(Refer section 1.7)** [10]

c) *What are the responsibilities of the DBA and database designers ?* **(Refer section 1.4)** [4]

OR

Q.2 a) *Define the following terms :*
i) Data model ii) Schema iii) Instance iv) Canned transaction. **(Refer section 1.6)** [8]

b) *Draw an ER diagram to represent the election information system based on the following description :*
In the Indian national election, a state is divided into a number of constituencies depending upon the population of the state. Several candidates contest elections in each constituency. Candidates may be from some party or independent. The election information system must record the number of votes obtained by each candidate. The system also maintains the voter list and a voter normally belongs to a particular constituency.
Note that the party details must also be taken care in the design. **(Refer example 1.14.4)** [12]

Module - 2

Q.3 a) *Define the following terms :*
i) Key ii) Super key iii) Candidate key iv) Primary key v) Foreign key **(Refer section 2.2)** [5]

b) *Enumerate the steps involved in converting the ER constructs to corresponding relational tables.* **(Refer section 2.10)** [7]

c) *Considering the schema*
Sailors (sid, sname, rating, age)
Boats (bid, bname, color)
Reserves (sid, bid, day)
Write relational algebraic queries for the following :
i) Find names of sailors who have reserved boat # 103.
ii) Find names of sailors who have reserved a red boat.
iii) Find names of sailors who have reserved a red or green boat.
iv) Find names of sailors who have reserved all boats. **(Refer example 2.9.15)** [8]

OR

Q.4 a) *Explain with examples, the basic constraints that can be specified when a database table is created in SQL.* **(Refer section 3.2)** [12]

b) *Write SQL queries for the following relational schema :*

CUSTOMER (CID, CNAME, EMAIL, ADDR, PHONE)
ITEM (ITEM_NO. , ITEM_NAME, PRICE, BRAND)
SALES (CID, ITEM_NO. #ITEMS, AMOUNT, SALE_DATE)
SUPPLIER (SID, SNAME, SPHONE, SADDR)
SUPPLY (SID, ITEM_NO. ,SUPPLY_DATE, QTY.)

i) List the items purchased by customer 'Prasanth".

ii) Retrieve items supplied by all suppliers starting from 1st Jan 2019 to 30th Jan 2019.

iii) Get the details of customers whose total purchase of items worth more than 5000 rupees.

iv) List total sales amount, total items, average sale amount of all items.

v) Display customers who have not purchased any items.

(Refer example 4.2.30) **[8]**

Module - 3

Q.5 a) *What are the assertions and triggers in SQL ? Write a SQL program to create an assertion to specify the constraints that the salary of an employee must not be greater than the salary of the department. The employee works for in the COMPANY database.* **(Refer section 4.3)** **[7]**

b) *Write a trigger in SQL to call a stored procedure INFORM_SUPERVISOR ( ) whenever a new record is inserted or updated, check whether an employee's salary is greater than the salary of his her direct supervisor in the COMPANY database.* **(Refer example 4.3.2)** **[7]**

c) *How do you create a view in SQL ? Give examples. Can you update a view table ? If yes, how ? If not, Why not ? Discuss.* **(Refer section 4.4)** **[6]**

OR

Q.6 a) *With real world examples, explain the following :*

i) JDBC **(Refer section 4.9)** *ii) Correlated queries* **(Refer section 4.1)**

iii) Stored procedure **(Refer section 4.5)**

iv) Schema change statements in SQL. **(Refer section 4.9)** **[12]**

b) *Write a complete high language program (In Java or C) to display the rows of a customer table created in oracle having < custid, custname, balance > columns with embedded SQL.* **(Refer example 4.6.2)** **[8]**

Module - 4

Q.7 a) *What are the problems caused by insertion, updation and deletion anomalies ? Discuss with an example.* **(Refer section 5.1.2)** **[6]**

b) *For the below given relation R (A,B,C,D,E) and its instance, check whether FDs given hold or not. Give reasons.* **(Refer example 5.2.1)**

i) $A \rightarrow B$ ii) $B \rightarrow C$ iii) $D \rightarrow E$ iv) $CD \rightarrow E$ **[4]**

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_2	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_3
a_2	b_3	c_3	d_2	e_2

c) *Using the minimal cover algorithm, find the minimal cover for the following FDs :* $F = \{AB \rightarrow C, A \rightarrow D, BD \rightarrow C, D \rightarrow BG, AE \rightarrow F\}$ **(Refer example 5.3.7)** **[10]**

OR

Q.8 a) *Normalize the below relation upto 3NF :* **(Refer example 5.10.6)** **[10]**

Module	Dept	Lecturer	Text
M1	D1	L1	T1
M1	D1	L1	T2
M2	D1	L1	T1
M2	D1	L1	T3
M3	D1	L2	T4
M4	D2	L3	T1
M4	D2	L3	T5
M5	D2	L4	T6

b) *Define multi valued dependency and join dependency. Explain 4NF and 5NF with examples.* **(Refer section 5.13)** **[10]**

Module - 5

Q.9 a) *Describe the database inconsistency problems : Lost update, dirty read and blind write.* **(Refer section 6.1)** **[6]**

b) *With a neat diagram, explain the various states of a transaction execution.* **(Refer section 6.2)** [7]

c) *Check whether the below schedule is conflict serializable or not. {b2, r2(X), b1, r1(X), W1(X), r1(Y), W1(Y), W2(X), e1, c1, e2, c2}.* **(Refer example 6.5.3)** [7]

OR

Q.10 a) *What is 2PL ? Explain with an example.* **(Refer section 6.8)** [6]

b) *How do you detect a deadlock during concurrent transaction execution ?* **(Refer section 6.8)** [6]

c) *Explain the various database recovery techniques, with examples.* **(Refer section 6.15)** [8]

❑❑❑

SOLVED MODEL QUESTION PAPER

(As Per 2018 Pattern)

Database Management System

Semester - V (CSE / ISE)

Time : 3 Hours] **[Max. Marks : 100**

Note : Answers any FIVE full questions, choosing ONE full question from each module.

Module - 1

Q.1 a) *Discuss the main characteristics of the database approach and how it differs from traditional file system.* **(Refer section 1.3)** [6]

b) *What are the responsibilities of the DBA and database designer ?* **(Refer section 1.4)** [6]

c) *Explain with block diagram the different phases of database design.* **(Refer section 1.10)** [8]

OR

Q.2 a) *Draw an ER-diagram of movie database. Assume your own, entities, attributes and relationships.* **(Refer example 1.14.2)** [10]

b) *What are the responsibilities of the DBA and database designer?* **(Refer section 1.4)** [10]

Module - 2

Q.3 a) *Explain the concept of Relational model constraints and relational database schemas.* **(Refer section 2.2)** [14]

b) *Discuss the characteristics of relations.* **(Refer section 2.1)** [6]

OR

Q.4 a) *Discuss the various types of set theory operations with example.* **(Refer section 2.7)** [10]

b) *Explain unary relational operations with suitable example.* **(Refer section 2.5)** [10]

Module - 3

Q.5 a) *Explain with examples, the basic constraints that can be specified when database table is created in SQL.* **(Refer section 3.2)** [12]

b) *Explain the data types available for attribute specification in SQL.* **(Refer section 3.1)** [8]

OR

Q.6 a) *How are views created and dropped ? Explain how the views are implemented and updated.* **(Refer section 4.4)** **[8]**

b) *Explain the concept of join operations using SQL on tables with the help of examples.* **(Refer section 4.1.6)** **[12]**

Module - 4

Q.7 a) *Explain aggregate functions in SQL.* **(Refer section 4.1.7)** **[10]**

b) *We have following relations :*

Supplier (S#, sname, status, city)

Parts (P#, pname, color, weight, city)

SP(S#, P#, quantity)

Answer the following queries in SQL,

i) Find name of parts whose color is 'red'.

ii) Find parts name whose weight less than 10 kg.

iii) Find all parts whose weight from 10 to 20 kg.

iv) Find average weight of all parts.

v) Find S# of supplier who supply part 'p2'. **(Refer example 4.2.6)** **[10]**

OR

Q.8 a) *Write a trigger in SQL to call a stored procedure INFORM_SUPERVISOR ( ) whenever a new record is inserted or updated, check whether an employee's salary is greater than the salary of his her direct supervisor in the COMPANY database.* **(Refer example 4.3.2)** **[12]**

b) *Explain schema change statements in SQL with examples.* **(Refer section 4.5)** **[8]**

Module - 5

Q.9 a) *Using the minimal cover algorithm, find the minimal cover for the following FDs : $F = \{AB \rightarrow C, A \rightarrow D, BD \rightarrow C, D \rightarrow BG, AE \rightarrow F\}$.* **(Refer example 5.3.7)** **[12]**

b) *Describe the database inconsistency problems : Lost update, dirty read and blind write.* **(Refer section 6.1)** **[8]**

OR

Q.10 a) *What is 2PL ? Explain with an example.* **(Refer section 6.8)** **[10]**

b) *Define multi valued dependency and join dependency. Explain 4NF and 5NF with examples.* **(Refer section 5.13)** **[10]**

❑❑❑

Notes