Uttar Pradesh Rajarshi Tandon
Open University

UGCS-104

Database Management System
(DBMS)

INDEX

PEPE-102

Uttar Pradesh Rajarshi Tandon
Open University

UGCS-104

Database Management System
(DBMS)

BLOCK

1

PEPE-102

Curriculum Design Committee

Dr. P. P. Dubey, **Coordinator**
Director, School of Agri. Sciences, UPRTOU, Allahabad
Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Allahabad
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg., MNNIT, Allahabad
Prof. P. K. Mishra **Member**
Dept. of Computer Science, Baranas Hindu University, Varanasi
Mr. Prateek Kesrwani **Member Secretary**
Academic Consultant-Computer Science
School of Science, UPRTOU, Allahabad

Course Design Committee

Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Prayagraj
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg., MNNIT, Allahabad, Prayagraj
Prof. P. K. Mishra **Member**
Dept. of Computer Science, Baranas Hindu University, Varanasi
Faculty Members, School of Sciences
Dr. Ashutosh Gupta, Director, Scool of Science, UPRTOU, Prayagraj
Dr. Shruti, Asst. Prof., (Statistics), Scool of Science, UPRTOU, Prayagraj
Ms. Marisha Asst. Prof., (Computer Science), Scool of Science, UPRTOU, Prayagraj
Mr. Manoj K Balwant Asst. Prof., (Computer Science), Scool of Science, UPRTOU, Prayagraj
Dr. Dinesh K Gupta Academic Consultant (Chemistry), Scool of Science, UPRTOU, Prayagraj

Course Preparation Committee

Mr. Sanjeev Gangwar* **Author**
Dept. of Computer Applications
VBS Purvanchal University, Jaunpur
Ms. Marisha[#] **Author**
Assistant Professor- Computer Science
School of Science, UPRTOU, Prayagraj
Dr. Ashutosh Gupta **Editor**
Director, School of Sciences,
UPRTOU, Prayagraj
Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Prayagraj
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg.,
MNNIT, Allahabad, Prayagraj
Prof. P. K. Mishra **Member**
Dept. of Computer Science
Baranas Hindu University, Varanasi
Dr. Dinesh K Gupta, **SLM Coordinator**
Academic Consultant- Chemistry School of Science, UPRTOU, Prayagraj

Note: Author's * -Block 1 and 2, # - Block-3

ISBN : 978-93-83328-29-1

Reprinted and Published by Vinay Kumar Registrar, Uttar Pradesh Rajarshi Tondon Open University, 2024

Printed By : Chandrakala Universal Pvt. Ltd 42/7 Jawahar Lal Nehru Road Prayagraj

UNIT-1

OVERVIEW OF DATABASE MANAGEMENT

Structure

1.1 INTRODUCTION

An organization must have accurate and reliable data for effective decision making. Thus the organization maintains data and records of its various operations electronically. A database is a collection of data typically describing the activities of one or more related departments e.g. a university database might contain the information about the following: entities such as students, faculties, courses and classrooms and relationships such as student's enrolment in courses, faculty teaching courses and the use of rooms for courses. The organization has to store these data effectively.

1.2 OBJECTIVES

After the end of this unit, you should be able to:

- Understand the role of a database management system in on organization.
- Know the design, implementation and use of database management systems.
- Describe the basic purpose and functions of a DBMS.
- Know the advantages and disadvantages of DBMS.
- Understand various database models which are used to design a correct, new database information system for a business organization.

1.3 TRADITIONAL FILE ORIENTED APPROACH

File processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to the appropriate files. Before DBMS came along, organizations usually stored information in such systems.

Traditionally, data was stored and processed on multiple files with the help of a program for each application. This is known as file-based approach of database management. A file may be defined as systematized self-containing collection of records. It may consist of data (data file) or it may contain a sequence of basic statements, each user work with a different program that handles its own independent data.

Traditional file oriented approach has the following disadvantages:

(1) Data redundancy (or duplication): Data redundancy is storage of the same piece of data in more than one place in the computer system. Lets us take an example of college where student record for examination is stored in other file and his library record is stored in different file that creates many duplicate values like roll Number, Name and Father Name.

(2) Data inconsistency (or loss of data integrity): Data inconsistency is a condition that occurs between files when similar data is kept in different formats in two different files, or when matching of data must be done between files. As a result of the data inconsistency, these files duplicate some data such as roll Number, Name and Father Name, compromising data integrity.

(3) Program-data dependence: File data is stored within each of the applications that use that data eg student record program may have several files relevant to student, faculty, and course. These files are integrated into the program.

(4) Poor data control: File systems have no centralised control of the data descriptions. Tables and field names may be used in different locations to mean different things. For example, the student record files may list a student as having a single 'Name' field that is made up of student's Initial and last name e.g. vijay gupta. The Examination department may keep the student's name in three separate fields; 'First name', 'Initial' &'Last Name'. This may make it difficult to compare the data in the two files or at least require additional time in programming the comparison.

(5) Security problems: Security to file system is not secured in conventional file management system. But in DBMS security is of high respect as the access paths are only open to part of the database.

(6) Limited data sharing: This dependence of the data on the program means that the files are not necessarily suitable for a new program that is being developed. The new program may need its data in another form or require additional data that is not held.

1.4 INTRODUCTION TO DATABASE

The term database is made up of two separate words, i.e., data and base. Database is a base for data, i.e., an assembled group of data. A database allows easy and efficient storage, retrieval and modification of data, regardless of the amount of data being manipulated, essentially, database is a computerized record keeping system.

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

Database is the combination of two words:

DBMS= Database + Management system

A simplified database environment is shown in Figure 1.1:

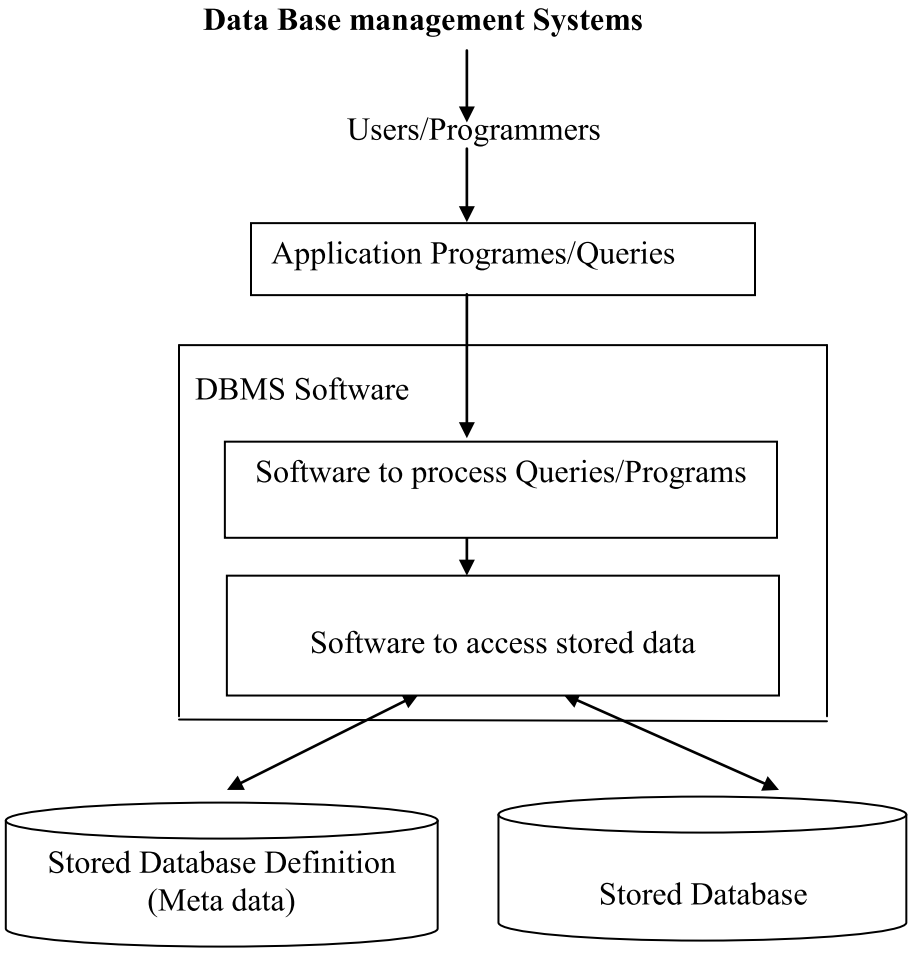


Fig. 1.1: A Simplified database environment

Data bases and database systems have become an essential component of everyday life in modern society.

1.4.1 Need to store data: A DBMS is software that allows access to data contained in the database. Its objective is to provide a convenient and effective method of defining, storing and retrieving the information contained in the database. The DBMS interfaces with application programs so that the data contained in the database can be used by multiple application and users.

1.4.2 Limitations of manual methods:

- Problems of speed
- Problems of accuracy
- Problems of consistency and reliability
- Problems of poor response time
- Problems of work-load handling capability
- Problems of meeting ad hoc information needs
- Problems of cost

1.4.3 Why computerized data processing?

- Advantage of speed
- Advantage of accuracy
- Advantage of reliability and consistency
- Advantage of storage and retrieval efficiency
- Advantage of on-line-access
- Advantage of cost

1.5 COMPONENTS OF DBMS

A database management system (DBMS) consists of several components. Each component plays very important role in the database management system environment. The major components of database management system are:

- Software
- Hardware
- Data
- Procedures
- Database Access Language

Software: The main component of a DBMS is the software. It is the set of programs used to handle the database and to control and manage the overall computerized database. Some examples of database software are Oracle, Microsoft Access, Microsoft SQL Server, SAP and MySQL.

Hardware: Hardware consists of a set of physical electronic devices such as computers (together with associated I/O devices like disk drives), storage devices, I/O channels, electromechanical devices that make interface between computers and the real world systems etc, and so on. It is impossible to implement the DBMS without the hardware devices. In a

network, a powerful computer with high data processing speed and a storage device with large storage capacity is required as database server.

Data: Data is the most important component of the DBMS. The main purpose of DBMS is to process the data. In DBMS, databases are defined, constructed and then data is stored, updated and retrieved to and from the databases. The database contains both the actual (or operational) data and the metadata (data about data or description about data).

Procedures: Procedures refer to the instructions and rules that help to design the database and to use the DBMS. The users that operate and manage the DBMS require documented procedures on hot use or run the database management system. The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings

AS

BEGIN

Dbms_output.put_line(‘Hello World’);

END;
```

When above code is executed using SQL prompt, it will produce the following result:

```
Procedure created.
```

The above procedure named 'greetings' can be called with the EXECUTE keyword as:

```
EXECUTE greetings;
```

The above call would display:

```
Hello World

PL/SQL procedure successfully completed.
```

Database Access Language: The database access language is used to access the data to and from the database. The users use the database access language to enter new data, change the existing data in database and to retrieve required data from databases. The user writes a set of appropriate commands in a database access language and submits these to the DBMS. The DBMS translates the user commands and sends it to a specific part of the DBMS called the Database Jet Engine. The database engine generates a set of results according to the commands submitted by user, converts

these into a user readable form called an Inquiry Report and then displays them on the screen. The database administrators may also use the database access language to create and maintain the databases. The most popular database access language is SQL (Structured Query Language). Relational databases are required to have a database query language.

Users: The users are the people who manage the databases and perform different operations on the databases in the database system. There are three kinds of people who play different roles in database system

1. Application Programmers
2. Database Administrators
3. End-Users

1. **Application Programmers**: The people who write application programs in programming languages (such as Visual Basic, Java, or C++) to interact with databases are called Application Programmer.
2. **Database Administrators:** A person who is responsible for managing the overall database management system is called database administrator or simply DBA.
3. **End-Users**: The end-users are the people who interact with database management system to perform different operations on database such as retrieving, updating, inserting, deleting data etc.

1.6 APPLICATION OF DBMS

1. Banking: for maintaining customer information, accounts, loans and banking transactions.
2. Universities: for maintaining student records, course registration and grades.
3. Railway Reservation: for checking the availability of reservation in different trains, tickets etc.
4. Airlines: for reservation and scheduled information.
5. Telecommunication: for keeping records of calls made, generating monthly bills etc.
6. Finance: for storing information about salary, sales and purchase information.
7. Sales: for storing customer, product and purchase information.

Check your Progress 1

Q1. What is data?

Q2. Name any two features of DBMS.

Q3. Name two database languages.

Q4. Differentiate between DDL and DML.

Q5. Draw the overall structure of DBMS and explain its various components.

1.7 ADVANTAGES OF DBMS

Some of the major advantages provided by the DBMS are:

1. **Improved data sharing:** Sharing of data allows the existing applications to use the data in the database. It also helps in developing new applications, which will use the same stored data. The DBMS helps in creating an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.

2. **Improved data security:** If more users access the data, greater is the risk of data security breaches. Corporations invest considerable amounts of time, effort, and money to e

3. nsure that corporate data are used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.

4. **Better data integration:** Data integrity refers to ensuring that the data in the database is accurate. Since, in DBMS, the data is centralized and is used by a number of users at a time, it is essential to enforce integrity controls.

5. **Minimized data inconsistency:** Data inconsistency exists when different versions of the same data appear in different places. For example, data inconsistency exists when a company's sales department stores a sales representative's name as "Sanjeev Agarwal" and the company's personnel department stores that same person's name as "Sanjeev Gangwar," or when the company's regional sales office shows the price of a product as Rs. 55.86 and its national sales office shows the same product's price as Rs. 55.96. The probability of data inconsistency is greatly reduced in a properly designed database.

6. **Efficient System:** It is very common to change the contents of stored data. These changes can easily be made in database management system. The cost of developing and maintaining system is also lower.

7. **Improved decision making:** Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. The quality of the information generated depends on the quality of the underlying data. Data quality is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives.

8. **Reduction in Data redundancy:** Data redundancy refers to the duplication of data. In non-database systems, each application has its own separate files. This can often lead to redundancy in stored data, which results in wastage of space. A DBMS does not maintain separate copies of the same data. All the data is kept at one place and various applications refer to the data from this centrally controlled system.

1.8 DISADVANTAGES OF DBMS

1. **Increased costs:** Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.

2. **Problems Associated with centralization:** A centralized database is a database that is located, stored, and maintained in a single location. This location is most often a central computer or system, for example a desktop or server CPU, or a mainframe computer. Centralization increases the security problems and disruption due to the downtimes and failures. Centralized databases are highly dependent on network connectivity. The slower the internet connection is, the longer the database access time needed will be.

3. **Maintaining Regular updation:** To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components

4. **Complexity of Backup and Recovery:** DBMS provide the centralization of the data, which requires the backup of data so that in case of failure, the data can be recovered. Hence backup problem is also the drawback.

1.9 ACID PROPERTIES IN DBMS

The Atomicity, Consistency, Isolation, Durability (ACID) properties are an important concept in DBMS because it allows secure sharing of data. There are four ACID properties available in DBMS which are as follows:

(1) Atomicity: It ensures that when an update occurs in a database either all or none of the update is made available to other users except the user who is performing the update.

(2) Consistency: It ensures that any change to a value of an instance is consistent with the change to other values in the same instance.

(3) Isolation: It is used when concurrent transactions occur. The concurrent transaction is a transaction that occurs at the same time. Multiple users accessing shared objects is an example of concurrent transaction.

(4) Durability: It ensures that the updates of committed transactions are maintained and never get lost.

1.10 DATABASE SYSTEM VERSUS FILE SYSTEMS

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. On the other hand file processing system is supposed by a conventional operating system. The system stores permanent records in various files and its need different application program to extract records and add records to the appropriate files. The basic difference between DBMS and file processing system is mentioned below:

DBMS	Traditional file oriented approach
Less Data Redundancy : In non-database systems each application program has its own private files. In this case, the duplicated copies of the same data are created in many places. In DBMS, all data of an organization is integrated into a single database file. The data is recorded in only one place in the database and it is not duplicated.	**More Data Redundancy :** Since different programs create the file and application programs, the various files are likely to have different formats and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). This redundancy leads to higher storage and access cost
Data Consistency: By controlling the data redundancy, the data consistency is obtained. If a data item appears	**Data Inconsistency:** Files that represent the same data may become

only once, any update to its value has to be performed only once and the updated value is immediately available to all users. If the DBMS has controlled redundancy, the database system enforces consistency.	inconsistent. This may happen because an update is applied to some of the files but not to others.
Efficient data access: In Database management system, data is stored in tables. A single database contains multiple tables and relationships can be created between tables (or associated data entities). This makes easy to retrieve and update data.	**Difficulty in accessing data:** Conventional file processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
Data Isolation: Isolation specifies when and how the changes implemented in an operation become visible to other parallel operations. A database acquires locks on data to maintain a high level of isolation	**Data Isolation:** Data are scattered in various files in different formats. Writing new program to retrieve appropriate data is difficult.
Data Security: If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce *access controls* that govern what data is visible to different classes of users.	**Security Problems:** Not every user of the database system should be able to access all the data. For example, in a banking system, payroll personnel need to see only that part of the database that has information about the various bank employees. They do not need access to information about customer accounts. So security constraints are difficult to manage.
Concurrent Access and Crash Recovery: DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.	**Concurrent access anomalies:** In order to improve the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such

	an environment, interaction of concurrent updates may result in inconsistent data.

1.11 THREE VIEW OF DATA OR DATA ABSTRACTION

The major purpose of a database system is to provide users with an abstract view of the system. The system hides certain details of how data is stored and created and maintained. Complexity should be hidden from database users.

There is several level of abstraction:

(1) Physical Level

(2) Conceptual(Logical Level)

(3) View Level

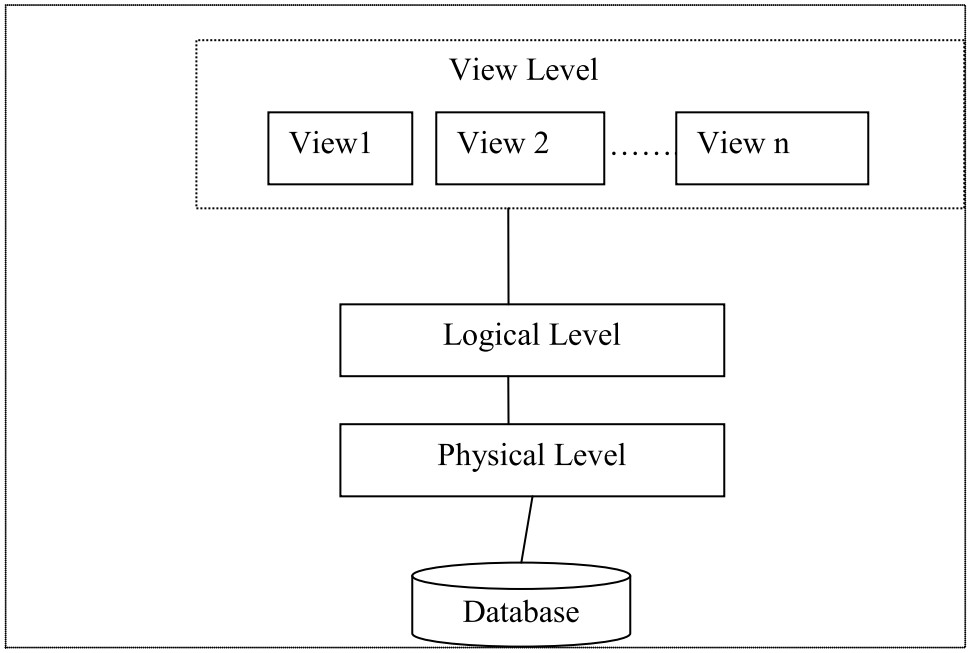


Fig. 1.2: Three levels of data abstraction

Physical Level (Internal Level): This is the lowest level in data abstraction. This level describes how the data is actually stored in the physical memory like magnetic tapes, hard disks etc. In this level the file organization methods like hashing, sequential, B+ tree comes into picture. At this level, developer would know the requirement, size and accessing

frequency of the records clearly. So designing this level will not be much complex for him.

Logical Level (Conceptual Level): This is the next level of abstraction. It describes the actual data stored in the database in the form of tables and relates them by means of mapping. This level will not have any information on what a user views at external level. This level will have all the data in the database. Any changes done in this level will not affect the external or physical levels of data. That is any changes to the table structure or the relation will not modify the data that the user is viewing at the external view or the storage at the physical level.

View Level (External view): This is the highest level in data abstraction. At this level users see the data in the form of rows and columns. This level illustrates the users how the data is stored in terms of tables and relations. Users view full or partial data based on the business requirement. The users will have different views here, based on their levels of access rights. For example, student will not have access to see Lecturers salary details, one employee will not have access to see other employees details, unless he is a manager. Any changes/ computations done at this level will not affect other levels of data. That means, if we retrieve the few columns of the STUDENT table, it will not change the whole table.

1.12 DATABASE LANGUAGES

Database languages are used for read, update and store data in a database. There are several such languages that can be used for this purpose; one of them is SQL (Structured Query Language). Database languages are of following types:

1.12.1 DDL (Data Definition Language)

DDL Stands for "Data Definition Language." A DDL is a language used to define data structures within a database. It is typically considered to be a subset of SQL, the Structured Query Language, but can also refer to language that defines other types of data. A Data Definition Language has a pre-defined syntax for describing data. For example, to build a new table using SQL syntax, the CREATE command is used, followed by parameters for the table name and column definitions. The DDL can also define the name of each column and the associated data type. Once a table is created, it can be modified using the ALTER command. If the table is no longer needed, the DROP command will delete the table.

Since DDL is a subset of SQL, it does not include all the possible SQL commands. For example, commands such as SELECT and INSERT are considered as a part of the Data Manipulation Language (DML).

Some examples of DDL Commands are:

- CREATE - to create objects in the database
- ALTER - alters the structure of the database

- DROP - delete objects from the database
- TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed
- COMMENT - add comments to the data dictionary
- RENAME - rename an object

1.12.2 DML (Data Manipulation Language)

DML stands for "Data Manipulation language". It is the area of SQL that allows changing data within the database. The DML can be used to

1. Insert the new data in the database.
2. Modify the data, already stored in the database.
3. Deletion of the data
4. Retrieve the information

It is of two types:

1. **Procedural DML**: it requires a user to specify what data are needed and how to get those data. It also requires writing the procedures/methods to specify what data is needed.
2. **Non-Procedural DML: i**t is also called as declarative DML and it require a user to specify what data are needed without specifying how to get those data. Non-procedural DML is the collection of commands for data manipulation including insert, update, delete and query the data.

Some examples of DML Commands are:

- SELECT - retrieve the required data from the a database
- INSERT - insert new data into a table
- UPDATE - updates the existing data within a table
- DELETE – delete some selected or all records from a table.
- MERGE - UPSERT operation (insert or update)
- CALL - call a PL/SQL or Java subprogram
- EXPLAIN PLAN - explain access path to data

1.13 SCHEMA AND INSTANCES

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction.

(1) Physical schema

(2) Conceptual schema

(3) External schema

(1) **Physical schema:** physical schema describes the database design at the physical level. It specifies additional storage details. Essentially the physical schema summarizes how the relations described in the conceptual schema are actually stored on secondary devices such as disks and tapes.

(2) **Conceptual schema:** the conceptual schema sometimes called logical schema describes the stored data in terms of the data model of the DBMS. In a relational DBMS the conceptual schema describes all relations that are stored in the database.

(3) **External Schema:** External Schema is a schema that represents the structure of data used by applications. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level. A database may also have several schemas at the view level, sometimes called **subschemas** that describe different views of the database

1.14 DATA MODELS

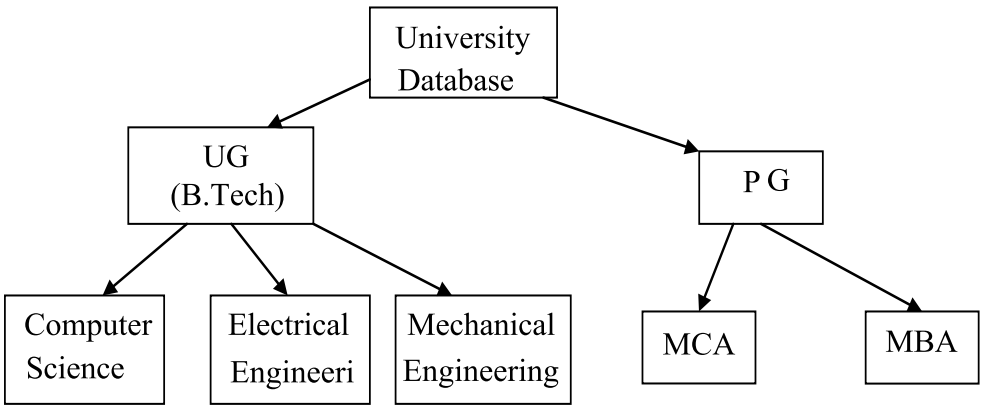
Data model is a collection of conceptual tools for describing data, data relationship, data semantics and consistency constraints. It consists of two parts:

- A mathematical notation for describing the data and relationships.
- A set of operations used to manipulate that data.

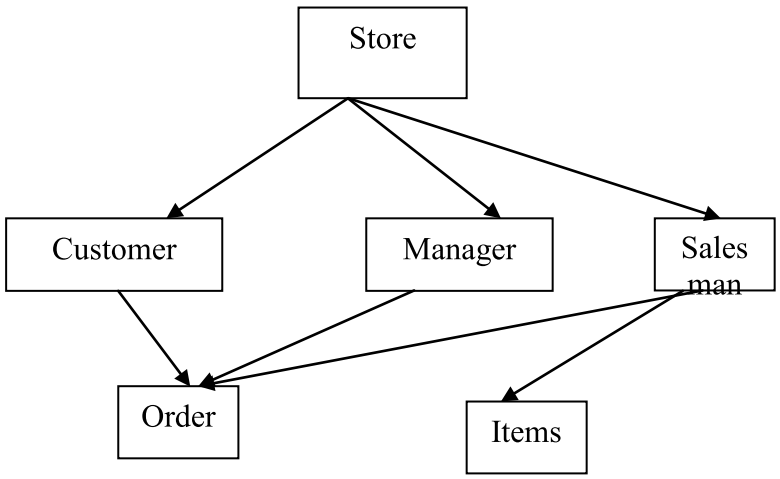
Every database and database management system is based on a particular database model. A database model consists of rules and standards that define how data is organized in a database. there are four basic types of database models: Hierarchical, Network, Relational and Object-oriented database model.

1.14.1 Hierarchical Database model: The data is sorted hierarchically, using a downward tree. This model uses pointers to navigate between stored data. It was the first DBMS model. It was developed by the IBM for its IMS (Information Management System) database. This model uses

parent-child relationship, that is, one-to-many relationship. The main advantage of hierarchical database is that data access is quite predictable in structure, and therefore, both retrieval and updates can be highly optimized by a DBMS. The hierarchical data model is shown in Figure 1.3.



1.14.2 Network Database Model: Like the hierarchical model, this model uses pointers toward stored data. However, it does not necessarily use a downward tree structure. This model allows having 1:1(one-to-one), 1: M (one-to-many) and M: M (Many-to-Many) relationships. The primary drawback of networked database is that it can be quite complicating to maintain all the links. The hierarchical data model is shown in Figure 1.4



Fig, 1.4: Network Model

1.14.3 Relational Database Model: Relational database management system is based on the relational model developed by E.F. Codd. A relational database represents all data in the database as simple two dimensional tables called relations that are the logical equivalent of files.

Each row (record) of a relational table, called tuple, represents a data entity with columns of the table representing attributes (fields). The allowable values for these attributes are called the domain.

Roll_no	Name	Address
230456	Sanjeev Gangwar	Paschim Vihar,Delhi
230457	Vijay Sharma	Premnagar, Bareilly
230458	Ajay Gupta	Lajpatnagar, Lucknow

Table 1: Student Table

1.14.4 Object Oriented Database Model: The object-oriented data model is another data model that has seen increasing attention. The object-oriented model can be seen as extending the E-R model with notions object-oriented data model. The object-relational data model combines features of the object oriented data model and relational data model. Semi structured data models permit the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The extensible markup language (XML) is widely used to represent semi structured data.

Check your Progress 2

Q1. How many levels do database system architecture consists of?

Q2. Name the type of data independence.

Q3. Name two database languages.

Q4. Differentiate between DDL and DML.

Q5. Explain Three Tier architecture with suitable diagram.

1.15 SUMMARY

- A Database Management System (DBMS) is a collection of interrelated data and a set of programs to access those data.
- DBMS can also be define as interface between the application program and the operating system to access or manipulate the database.
- A database model is a collection of conceptual tools for describing data, data relationship, data semantics and consistency constraints.

They are of four types: Hierarchical, Network, Relational and Object-oriented database model.

- ❖ The collection of information stored in the database at particular moments is called an instance of the database.
- ❖ The overall design of the database is called the database schema. It is of three types: Physical Schema, conceptual schema and External Schema.
- ❖ Data Definition language (DDL) is used to create, modify and delete database.
- ❖ Data Manipulation Language (DML) is used to access or manipulate database. Here we can update, Add, insert and modify the database. It is of two types: procedural DML and Non-procedural DML.

1.16 OBJECTIVE TYPE QUESTIONS

Choose the correct or best alternative in the following:

Q.1 In the relational modes, cardinality is termed as:

(A) Number of tables. **(B)** Number of constraints.

(C) Number of tuples. **(D)** Number of attributes.

Ans: A

Q.2 The view of total database content is

(A) Conceptual view. **(B)** Internal view.

(C) External view. **(D)** Physical View.

Ans: A

Q3. DML is provided for

(A) Description of logical structure of database.

(B) Addition of new structures in the database system.

(C) Manipulation & processing of database.

(D) Definition of physical structure of database system.

Ans: C

Q4. Architecture of the database can be viewed as

(A) Two levels. **(B)** Four levels.

(C) Three levels. **(D)** One level.

Ans: C

Q.5 The database schema is written in

(A) HLL **(B)** DML

(C) DDL **(D)** DCL

Ans: C

Q.6 In the architecture of a database system external level is the

(A) Physical level. **(B)** Logical level.

(C) Conceptual level **(D)** view level.

Ans: D

Q.7 In a Hierarchical model records are organized as

(A) Graph. **(B)** List.

(C) Links. **(D)** TREE

Ans: D

Q.8 A logical schema

(A) is the entire database**.**

(B) is a standard way of organizing information into accessible parts.

(C) describes how data is actually stored on disk.

(D) both (A) and (C)

Ans: A

Q.9 A relational database developer refers to a record as.

(A) A criteria. **(B)** A relation.

(C) A tuple. **(D)** an attribute.

Ans: C

Q.10 An advantage of the database management approach is.

(A) Data is dependent on programs.

(B) Data redundancy increases.

(C) Data is integrated and can be accessed by multiple programs.

(D) None of the above.

Ans: C

1.17 SELECTED EXERCISES

1. Why would you choose a database system instead of simply storing data in operating system files?
2. Give the structure of DBMS and explain the role of various users in it.
3. What do you mean by External Schema, Conceptual Schema and Internal schema?
4. What do you mean by Data Abstraction? Explain its types in details.
5. Explain different languages that are supported to manage the data in DBMS.
6. Explain ACID Properties in database.
7. Explain any two DML commands with example.
8. What is the difference between physical data independence and logical data independence?
9. Explain the merits and demerits of data base system.
10. Distinguish between procedural and non-procedural DML's.

UNIT-2

DATABASE MODELS AND IMPLEMENTATION

Structure

2.1 INTRODUCTION

The main objective of database system is to highlight only the essential features and to hide the storage and data organization details from the user. This is known as **data abstraction**. A database model provides the necessary means to achieve data abstraction. A **database model** or simply a **data model** is an abstract model that describes how the data is represented and used. A data model consists of a set of data structures and conceptual tools that is used to describe the structure (data types, relationships, and constraints) of a database.

A data model not only describes the structure of the data, it also defines a set of operations that can be performed on the data. A data model generally consists of data model theory, which is a formal description of how data may be structured and used, and data model instance, which is a practical data model designed for a particular application. The process of applying a data model theory to create a data model instance is known as data modeling.

Data model means, to give a SHAPE to the data. A Data model makes it easier to understand the data. We can also define the data model as "The collection of high-Level data description that hides many low level storage details".

The data models are divided into three different groups. They are,

1. Object - Based Logical Models
2. Record - Based Logical Models
3. Physical Data Models

In this unit, we will discuss these data models in detail.

2.2 OBJECTIVES

At the end of this unit, you should be able to:

- Understand the difference between database approaches and file processing approach.
- Know the various data models.
- Understand the requirements for conceptual data modeling.
- Understand data model and its ability to be used as a blueprint to build a physical database.
- Build a database from a data model that will result in a better database implementation.

2.3 FILE MANAGEMENT SYSTEM

Data are stored in files in all information systems. Files are collections of similar records. Data storage is build around the corresponding application that uses the files as shown in Figure 2.1. Important features of file processing system are

- Where data are stored to individual files is a very old, but often used approach to system development.
- Each program (system) often had its own unique set of files.

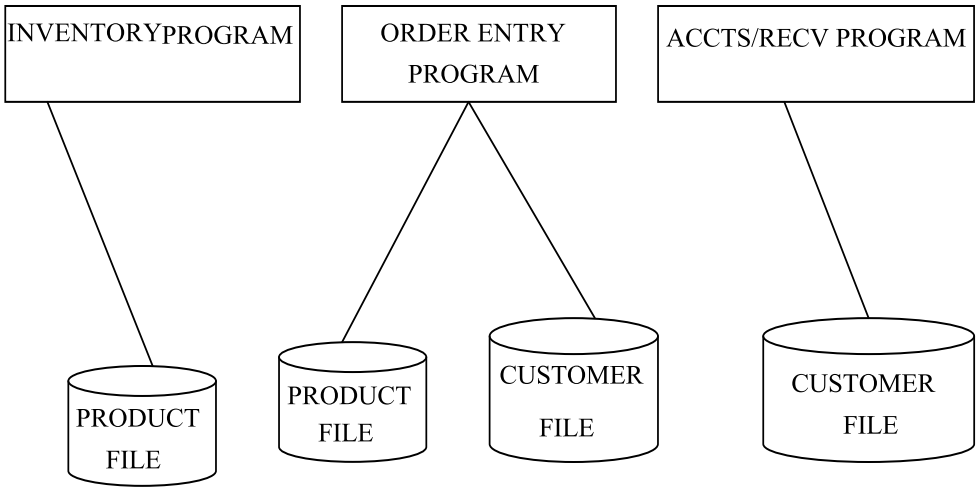


Fig. 2.1: FILE PROCESSING SYSTEM

2.3.1 What is a file?

- A File is a collection of data about a single entity.
- Files are typically designed to meet needs of a particular department or user group.
- Files are also typically designed to be part of a particular computer application

2.3.2 Advantages

- ❖ Files are relatively easy to design and implement since they are normally based on a single application.
- ❖ The processing speed is faster than other ways of storing data.

2.3.3 Disadvantages

- Duplication of data: In file system approach each user has to maintain a separate copy of the same file that results in redundant information.
- Inconsistency: In file system approach, the same information is stored in more than one file. When a record is deleted or updated in one file, the corresponding record should be deleted from all the files which are associated with that record. If any file is left unchanged then the data in the files becomes inconsistent and may provide wrong information when required.
- Lengthy program and system development time: In file system approach, if any new functionality is required than to implement that functionality the complete file system and the associated programs have to modified, which is a time consuming task.
- Excessive program maintenance when the system changed. Duplication of data items in multiple files. Duplication can affect

on input, maintenance, storage and possibly data integrity problems.

- Inflexibility and non-scalability: Since the conventional files are designed to support single application, the original file structure cannot support the new requirements.
- Accessing of information is slower: To access required information in file system approach the user has to manually perform the access process to obtain a sequence of information and then arrange them to derive the result.

2.4 DATABASE MANAGEMENT SYSTEM

Database management system is a software of collection of small programs to perform certain operation on data and manage the data.

Two basic operations performed by the DBMS are:

- Management of Data in the Database
- Management of Users associated with the database.

Management of the data means to specify that how data will be stored, structured and accessed in the database. Management of database users means to manage the users in such a way that they can perform any desired operations on the database. A DBMS also ensures that a user cannot perform any operation for which he is not allowed and also an authorized user is not allowed to perform any action which is restricted to that user. In general DBMS is a collection of programs performing all necessary actions associated to a database.

2.5 DATABASE APPROACH VS. FILE PROCESSING APPROACH

Consider an organization/enterprise that is organized as a collection of departments/offices. Each department has certain data processing "needs", many of which are unique to it. In the **file processing approach**, each department would "own" a collection of relevant data and software applications to manipulate that data.

For example, a university's Registrar's Office would maintain (most likely, with the aid of programmers employed by the university's "computer center") data (and programs) relevant to student grades and course enrollments. The A.O Office would maintain data (and programs) regarding fees owed by students for tuition, room and board, etc.

One result of this approach is, typically, **data redundancy**, which not only wastes storage space but also makes it more difficult to keep changing data up-to-date, as a change to one copy of some data item must be made to all of them (called **duplication-of-effort**). **Inconsistency**

results when one (or more) copies of a datum are changed but not others. (e.g., if you change your address, informing the department's office should suffice to ensure that your grades are sent to the right place, but does not guarantee that your next bill will be, as the copy of your address "owned" by the A.O's Office might not have been changed.)

While in the **database approach**, a single repository of data is maintained that is used by all the departments in the organization.

2.5.1 Main Characteristics of database approach

1. Self-Description

A database system not only includes the data stored that is of relevance to the organization but also a complete definition/description of the database's structure and constraints. This **meta-data** (i.e., data about data) is stored in the so-called **system catalog**, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy).

The system catalog is used not only by users (e.g., who need to know the names of tables and attributes, and sometimes data type information and other things), but also by the DBMS software, which certainly needs to "know" how the data is structured/organized in order to interpret it in a manner consistent with that structure. Recall that a DBMS is *general purpose*, as opposed to being a specific database application. Hence, the structure of the data cannot be "hard-coded" in its programs, but rather must be treated as a "parameter" in some sense.

2. Program-Data Independence

DBMS architecture can be used to explain the concept of data independence which is the ability to change the representation of data at one level of a database system without the compulsion of changing the data representation at the next higher level. Two types of data independence can be defined:

Logical data independence: It is the ability to change the representation of data at the conceptual level without having to change the representation of data at the external level. For example if you want to expand the database by adding a record type of data item, you will have to change the conceptual level. The change in the conceptual level can be made accordingly and the external level that refers to the remaining data need not be changed.

Physical data independence: It is the ability to change the representation of data at the internal level without having to change the representation of data at the conceptual or external level. Changes to the internal level may be needed, if some physical files are to be recognized. For example if you want to improve the performance of retrieval or update of a database you may need to create additional access structures. This may result in file reorganization. If the data stored in the database does not change you will not have to change the conceptual level.

In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. (e.g., consider a student file in a C program which uses array of structures: it gives a detailed description of the records in a file.)

If, for some reason, we decide to change the structure of the data (e.g., by adding another field Blood Group), **every** application in which a description of that file's structure is hard-coded must be changed!

In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described (in the system catalog) separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data (i.e., providing a means by which to determine boundaries between records and between fields within records) so that they interpret that data properly.

In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified. (This is referred to as *program-data independence*.)

3. Multiple Views of Data

Different users (e.g., in different departments of an organization) have different "views" or perspectives on the database. For example, from the point of view of A.O's Office , student data does not include anything about which courses were taken or which grades were earned. (This is an example of a **subset** view.)

As another example, a Registrar's Office employee might think that PERCENTAGE is a field of data in each student's record. In reality, the underlying database might calculate that value each time it is called for. This is called **virtual** (or **derived**) data.

A view designed for an academic advisor might give the appearance that the data is structured to point out the prerequisites of each course. A good DBMS has facilities for defining multiple views. This is not only convenient for users, but also addresses security issues of data access.

4. Data Sharing and Multi-user Transaction Processing

As we know that the simultaneous access of computer resources by multiple users/processes is a major source of complexity. The same is true for multi-user DBMS's.

Arising from this is the need for **concurrency control**, which is supposed to ensure that several users trying to update the same data do so in a "controlled" manner so that the results of the updates are as though they were done in some sequential order (rather than interleaved, which could result in data being incorrect).

This gives rise to the concept of a **transaction**, which is a process that makes one or more accesses to a database and which must have the appearance of executing in *isolation* from all other transactions (even ones

that access the same data at the "same time") and of being *atomic* (in the sense that, if the system crashes in the middle of its execution, the database contents must be as though it did not execute at all).

Applications such as airline reservation systems are known as online transaction processing applications.

CHECK YOUR PROGRESS 1

Q1. Define file.
Q2. Define data independence.
Q3. What is a transaction?
Q4. What is data model?
Q5. Define the following term
(i) Data redundancy
(ii) Data consistency

2.6 DATA MODELS

Data model is a collection of conceptual tool for describing data relationship. Data Model is used to describe the overall data base schema or structure. A data model provides a way to describe a design of a database at view level, logical level, and physical level.

The data models mainly classified in four categories,

(1) Object based Logical Model

(2) Record Based Logical Model

(3) Logical Model

(4) Physical Model

2.6.1 Object Based Logical Model

In the recent years, the object-oriented paradigm has been applied to database technology, creating two new data models known as object-oriented data model and object-relational data model. The object-oriented data model extends the concepts of object-oriented programming language with persistence, versioning, concurrency control, data recovery, security, and other database capabilities. On the other hand, the object-relational data model is an extension of relational data model. It combines the features of both the relational data model and object-oriented data model.

- It is used in describing data at logical & view level.
- This model is based on a collection of objects; an object contains value stored in instance variable.
- Objects that contain the same types of values and the same methods are grouped into class.

The Object based logical models are described in the different following models.

(a) The E-R (Entity-Relationship) Model

(b) The Object-Based Logical Model

(c) The Semantic Data Model

(d) The Functional Data Model

(a) E-R Model

The entity is a "Thing" or “Object" in the real world that is distinguishable from other objects. The E-R model is based on the collection of basic objects called **Entities** and the **Relationship** among them. Consider the following diagram shown in Figure 2.2.

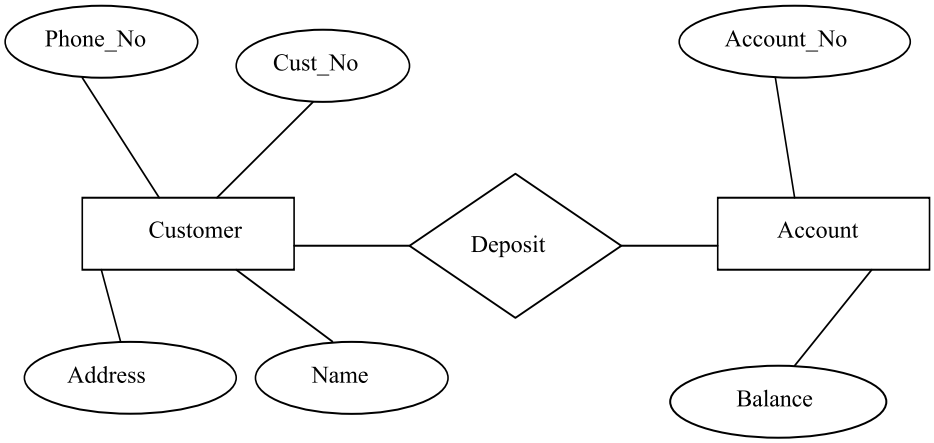


Fig. 2.2: ER diagram for banking system

In the above diagram, RECTANGLES represents ENTITIES, DIAMONDS represents RELATIONSHIP among those ENTITIES. LINES represent links of Entities to Relationships.

The entity-relationship (E-R) model is discussed in detail in Unit 3.

(b) Object - Oriented Model

The Object oriented model is based on a collection of objects. An object contains values stored in instance variables and bodies of code that operates on the object. (These bodies of code are called methods).

Objects that contain the same types of values and the same methods are grouped together into classes.

Features of object oriented model are:

- Object oriented model describe real life entity easily.

- Basic elements of object oriented model are class, methods, and object.
- Using object we can call methods (Functions) of class.
- In Object Oriented Model, each different type of information is stores in different files.

For example consider the Customer purchase Items scenario in which customer related information's are stores in one class, Purchase Related information's are stores in one class and Item related information's are stores in one class.

In object oriented model each information stores in different class. So that if in future, requirement changes we can easily handle that kind of situation

Advantages: -

- Data Abstraction: It hides the internal representation or complexity.
- In future if requirement change then it require changing the code of method. In this case we can change code without changing the internal code of method.
- Inheritance: Because of its inheritance property, we can re-use the attributes and functionalities. It reduces the cost of maintaining the same data multiple times.

(c) Semantic Data Model

A Semantic data model is a more high level data model that makes it easier for a user to give **Starting Description** of the data in an organization. (Semantic is nothing but the meaning). These models contain a wide verity of relations that helps to describe a real application scenario. A DBMS cannot support all these relations directly. So it is build with only few relations known as relational model in DBMS. A widely used semantic data model is the Entity-Relationship (ER) data model which allows us to graphically denote entities and relationship between them.

(d) Functional Data Model

The functional data model makes it easier to define functions and call them where ever necessary to process data.

2.6.2 Record - Based Logical Models

In this type of models, the data is kept in the form of RECORDS (documents). These models describe data at Logical and View Levels. When compared with object based data models, the record based logical models specify the overall logical structure of the database and provides higher-level implementation.

- Record base logical model are used in describing data and view levels.
- Record base model are so popular because the database is structured in fixed format record of several times.
- Each record type defines a fixed no of fields and each fields of a fixed length. The used of fixed length record simplify the physical level implementation of database.

Following are the three types of record based logical models:

(a) Hierarchical Model

(b) Network Model

(c) Relational Model

(a) Hierarchical Model

The hierarchical data model is the oldest type of data model, developed by IBM in 1968. This data model organizes the data in a tree-like structure, in which each child node (also known as dependents) can have only one parent node. The database based on the hierarchical data model comprises a set of records connected to one another through links. The link is an association between two or more records. The top of the tree structure consists of a single node that does not have any parent and is called the root node.

The root may have any number of dependents; each of these dependents may have any number of lower level dependents. Each child node can have only one parent node and a parent node can have any number of (many) child nodes. It, therefore, represents only one-to-one and one-to-many relationships. The collection of same type of records is known as a record type.

Figure 2.3 represents a hierarchical schema which has three record types and two Parent-Child Relationship (PCR) types. Department, Employee and Project are the record types in Figure 2.3.

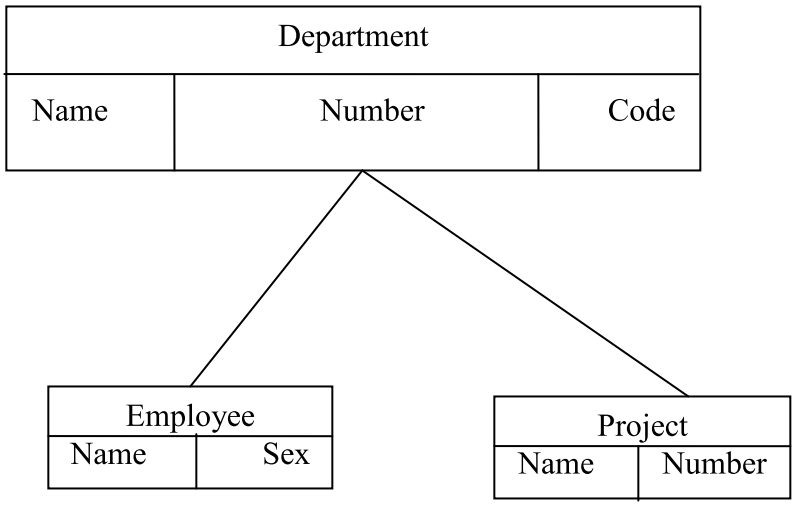


Fig. 2.3: Hierarchical Model

Each record type can have a set of data items or fields. For example record type Department can have department name, department number and department code as the field or data items. Parent-Child Relationship type can be represented by listing pair in parentheses. For example in Figure 2.3 there are two Parent-Child Relationship types which can be represented as (Department, Employee) and (Department, Project). In Figure 2.3 each occurrence of the (Department, Employee) Parent-Child Relationship type relates one department record to the records of the many employees who works in that department. The occurrence of (Department, Project) Parent-Child Relationship type relate a document record to the records of projects controlled by that department.

Figure 2.4 represents the tree like structure of the hierarchical schema shown in Figure 2.3.

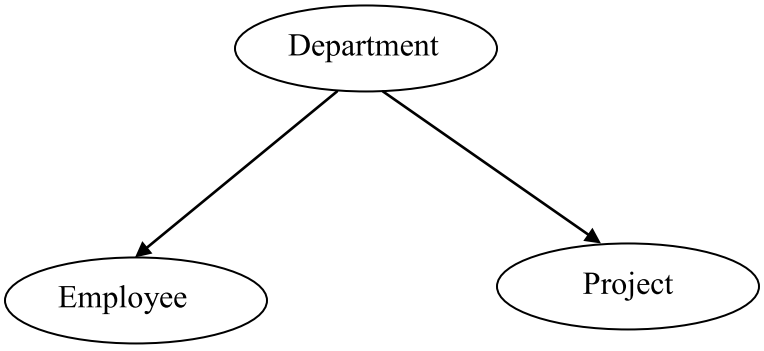


Fig. 2.4: Tree Representation of Hierarchical Schema

In tree like structure, a record type is represented by node of the tree and parent-child relationship type is represented by arc of the tree.

The main advantage of the hierarchical data model is that the data access is quite predictable in the structure and, therefore, both the retrieval and updates can be highly optimized by the DBMS. However, the main drawback of this model is that the links are 'hard coded' into the data structure, that is, the link is permanently established and cannot be modified. The hard coding makes the hierarchical model rigid. In addition, the physical links make it difficult to expand or modify the database and the changes require substantial redesigning efforts.

Points to remember

- In the hierarchical model, records are organized as trees rather than arbitrary graphs.
- The record type at the top of the tree is usually known as root.
- In general, the root may have any number of dependents and each of these dependents may have any number of low level dependents and so on.

(b) Network Model

The first specification of network data model was presented by Conference on Data Systems Languages (CODASYL) in 1969, followed by the second specification in 1971. It is powerful but complicated. In a network model the data is also represented by a collection of records, and relationships among data are represented by links. However, the link in a network data model represents an association between precisely two records. Like hierarchical data model, each record of a particular record type represents a node. However, unlike hierarchical data model, all the nodes are linked to each other without any hierarchy. The main difference between hierarchical and network data model is that in hierarchical data model, the data is organized in the form of trees and in network data model, the data is organized in the form of graphs.

The main advantage of network data model is that a parent node can have many child nodes and a child can also have many parent nodes. Thus, the network model permits the modeling of many-to-many relationships in data. The main limitation of the network data model is that it can be quite complicated to maintain all the links and a single broken link can lead to problems in the database. In addition, since there are no restrictions on the number of relationships, the database design can become complex. Figure 2.5 shows the network model of online book database.

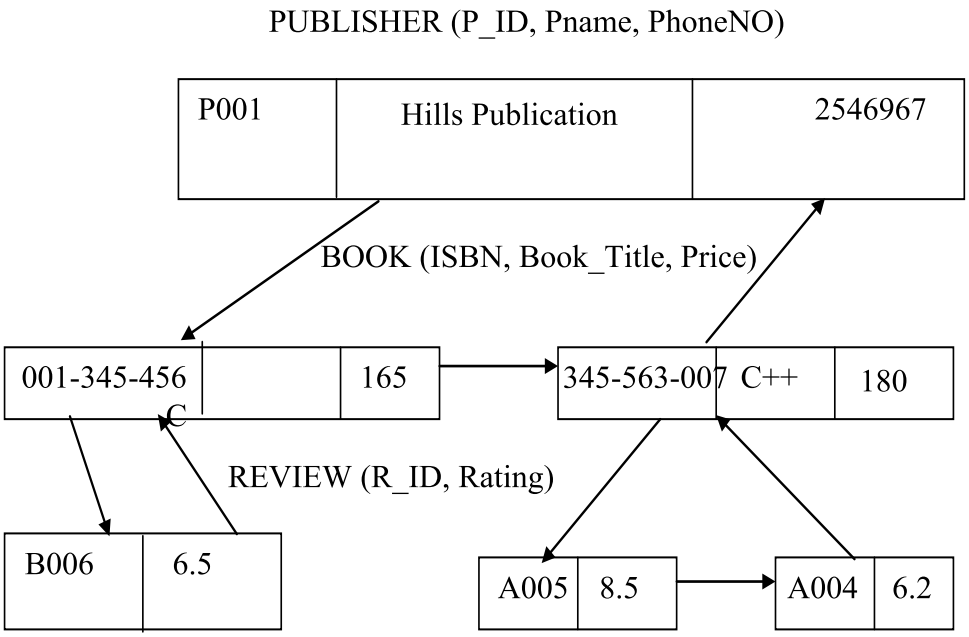


Fig. 2.5: Network data model for Online Book database

Point to Remember

- The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child.
- Data in the network are represents by collection of records and relationship among data are represented by links. This links can view as a pointer. The records in the database are organized as a collection of arbitrary graph.

(c) Relational Model

The relational data model was developed by E. F. Codds in 1970. In the relational data model, unlike the hierarchical and network models, there are no physical links. All data is maintained in the form of tables (generally, known as relations) consisting of rows and columns. Each row (record) represents an entity and a column (field) represents an attribute of the entity. The relationship between the two tables is implemented through a common attribute in the tables and not by physical links or pointers. This makes the querying much easier in a relational database system than in the hierarchical or network database systems. Thus, the relational model has become more programmers friendly and much more dominant and popular in both industrial and academic scenarios. Oracles, Sybase, DB2, Ingres, and Informix, MS-SQL Server are few of the popular relational DBMSs.

Figure 2.6 shows the relational model of Online Book database. The relational model represents both Data (entities) and Relationships among that data in the form of **Tables.** Each table contains multiple columns and each column contains a unique name.

BOOK

ISBN	Book_Title	Category	Price	Copyright_Date	Year	Page_count	P_ID
001-345-456	C	Novel	165	2007	2006	260	P001
345-563-007	C++	Textbook	180	2009	2008	670	B006

PUBLISHER

P_ID	Pname	Address	Phone	Email
P001	Hills	USA	2546967	Hills@gmail.com

REVIEW

R_ID	ISBN	Rating
B006	001-345-456	6.5
A005	345-563-007	8.5
A004	345-563-007	6.2

Fig. 2.6: Relational data model for Online Book database

Point to Remember:

- The relational model uses a collection of a table to represent data and the relationship among those data.
- Each table is consist of multiple columns and each column has a unique name.
- The Relational Model is a depiction of how each piece of stored information relates to the other stored information. It shows how tables are linked, what type of links are between tables, what keys are used, what information is referenced between tables. It's an essential part of developing a normalized database structure to prevent repeat and redundant data storage.

Advantages:

(1) Ease of use: relational data model is easy to implement as data is stored in the forms of tables.

(2) Flexibility: relation data model is flexible as changes in the database structure need not be changed in all related application programs.

(3) Security: By splitting data into tables, certain tables can be made confidential. When a person logs on with their username and password, the system can then limit access only to those tables whose records they are authorized to view.

(4) Data Independence: Data independence is achieved more easily with normalization structure used in a relational database than in the more complicated tree or network structure.

2.6.3 Logical Model: A logical data model describes the data in as much detail as possible, without regard to how they will be physical implemented in the database. Features of a logical data model are:

- all entities and relationships among them.
- All attributes for each entity are specified.
- The primary key for each entity is specified.
- Foreign keys (keys identifying the relationship between different entities) are specified.
- Normalization occurs at this level.

The steps for designing the logical data model are as follows:

1. Specify primary keys for all entities.
2. Find the relationships between different entities.
3. Find all attributes for each entity.
4. Resolve many-to-many relationships.
5. Normalization.

An example of a logical data model is shown in Figure 2.7

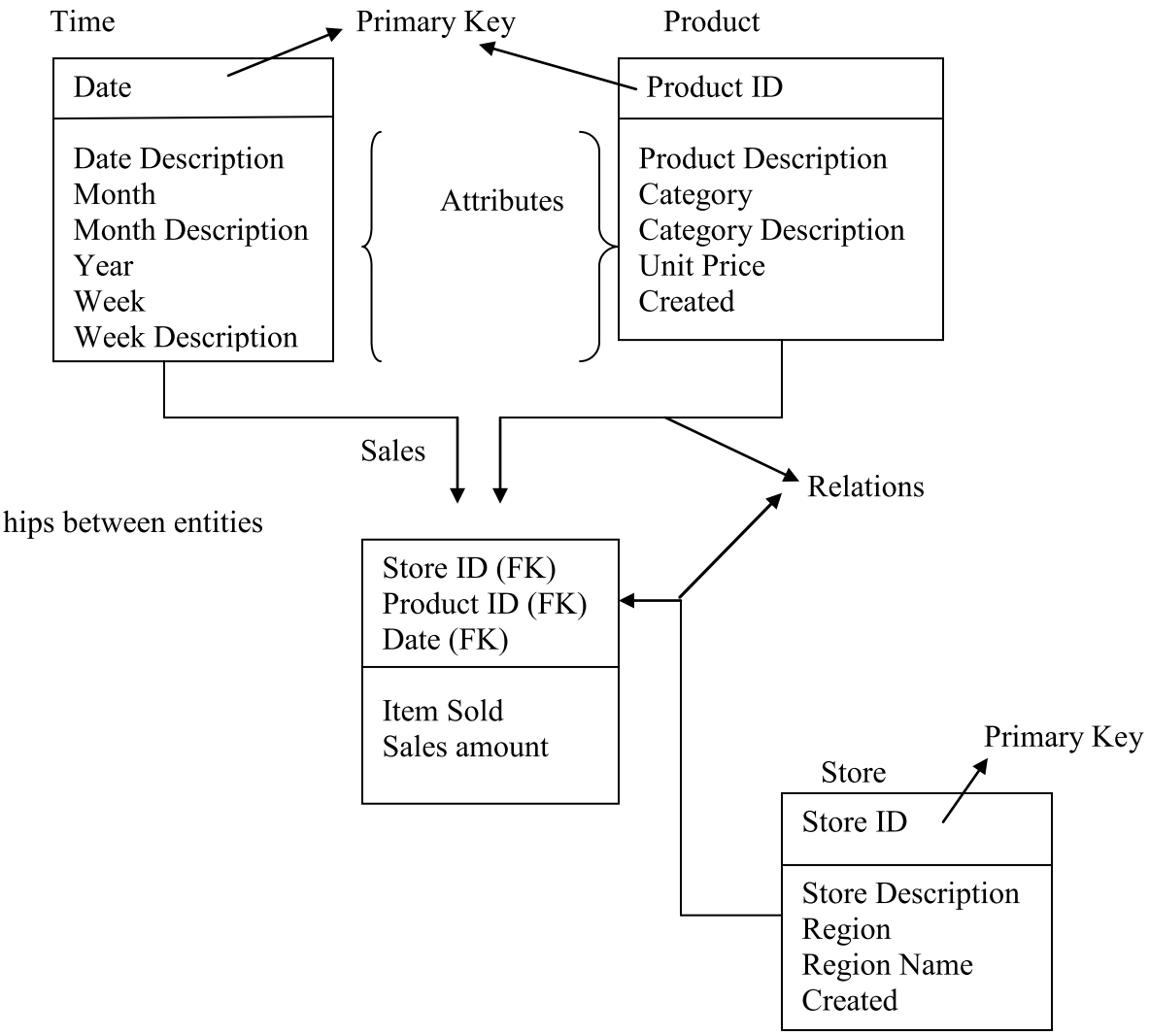


Fig. 2.7: Logical Data Model

2.6.4 Physical Model

Physical data model represents how the model will be built in the database. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables. Features of a physical data model include:

- Specification of all tables and columns.
- Foreign keys are used to identify relationships between tables.
- Denormalization may occur based on user requirements.

- Physical considerations may cause the physical data model to be quite different from the logical data model.
- Physical data model will be different for different RDBMS. For example, data type for a column may be different between MySQL and SQL Server.

The steps for physical data model design are as follows:

1. Convert entities into tables.
2. Convert relationships into foreign keys.
3. Convert attributes into columns.
4. Modify the physical data model based on physical constraints / requirements.

An example of a physical data model is shown in Figure 2.8.

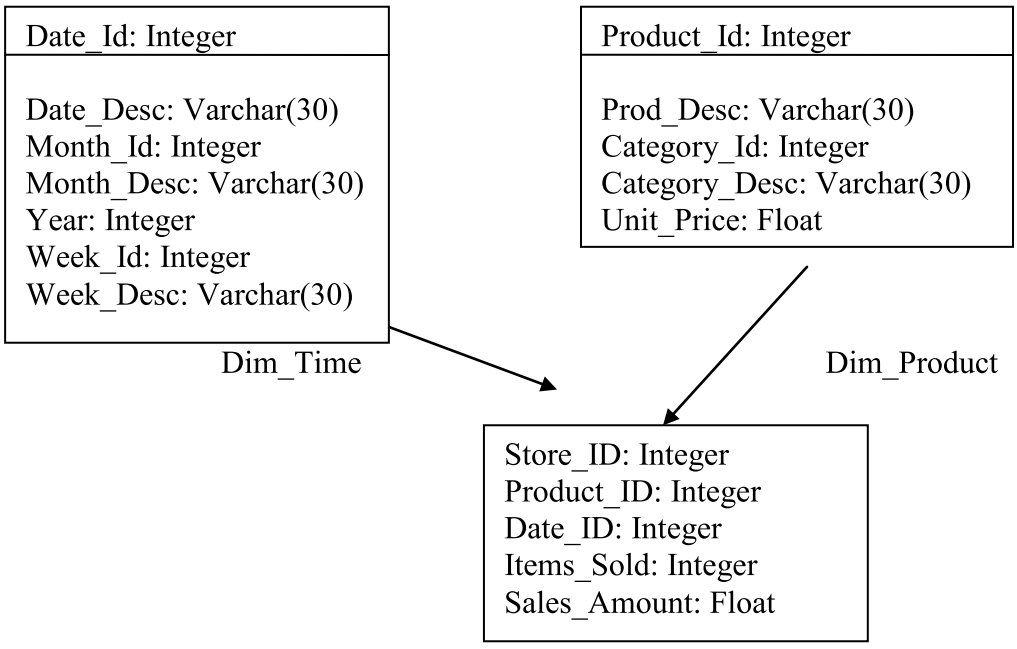


Fig. 2.8: Physical Data Model

CHECK YOUR PROGRESS 2

Q1. What is ER diagram?

Q2. What is Table?

Q3. Define the term PCR.

Q4. Define Logical Model.

Q5. Describe Hierarchical Data Model with suitable example

2.7 SUMMERY

- The main objective of database system is to highlight only the essential features and to hide the storage and data organization details from the user. This is known as data abstraction.
- A database model or simply a data model is an abstract model that describes how the data is represented and used.
- The Data Models are divided into four different groups: Object - Based Logical Models, Record - Based Logical Models , Logical Model and Physical Data Models.
- Database management system is software of collection of small programs to perform certain operation on data and manage the data.
- Data are stored in files in all information systems. Files are collections of similar records.
- Data model is collection of conceptual tool for describing for data relationship. Data Model is used to describe the overall data base schema or structure.
- The Object based logical models are described in the different following models.: The E-R (Entity-Relationship) Model, The Object-Based Logical Model, The Semantic Data Model The Functional Data Model
- In record based logical model the data is kept in the form of RECORDS (documents). These models describe data at Logical and View Levels.
- Record based logical models are divided into three categories: Hierarchical model, Network Model and Relational Model.
- Physical data model represents how the model will be built in the database. A physical database model shows all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables.

2.8 OBJECTIVE TYPE QUESTIONS

Choose the correct or best alternative in the following:

Q1. The conceptual model is

(A) Dependent on hardware.

(B) Dependent on software.

(C) Dependent on both hardware and software.

(D) Independent of both hardware and software.

Ans: D

Q 2. Which database level is closest to the users?

(A) External **(B)** Internal

(C) Physical **(D)** Conceptual

Ans: A

Q3. Which of the following is another name for weak entity?

(A) Child **(B)** Owner

(C) Dominant **(D)** All of the above

Ans: A

Q4. Which of the following is record based logical model?

(A) Network Model **(B)** Object oriented model

(C) E-R Model **(D)** None of these

Ans: A

Q5. In E-R Diagram relationship type is represented by

(A) Ellipse **(B)** Dashed ellipse

(C) Rectangle **(D)** Diamond

Ans: D

Q6. Hierarchical model is also called

(A) Tree structure **(B)** Plex Structure

(C) Normalize Structure **(D)** Table Structure

Ans: A

Q7. Which two files are used during operation of the DBMS?

(A) Query languages and utilities

(B) DML and query language

(C) Data dictionary and transaction log

(D) Data dictionary and query language

Ans: C

Q8. In relation schema, each tuple is divided into fields called

(A) Relations **(B)** Domain

(C) Queries **(D)** All of the above

Ans: B

Q9. The relational database environment has all of the following components except.

(A) Users **(B)** separate files

(C) Database **(D)** query languages

Ans: B

Q10. The view of total database content is

(A) Conceptual view **(B)** Internal view

(C) External view **(D)** Physical view

Ans: A

2.9 SELECTED EXERCISES

1. What is data model? Explain object based and record based data models.

2. Explain the disadvantages of a file processing system.

3. Explain the difference between Database approaches and file oriented approach.

4. Explain the following

(a) Network Model

(b) Hierarchical Model

(c) Relational Model

5. Explain the concepts of relational data model. Also discuss its advantages and disadvantages.

6. Write short notes on the following

(a) Object-based logical models

(b) Record based logical models

(c) Physical data models

7. What are the different data models present and explain briefly?

UNIT-3

ENTITY RELATIONSHIP MODEL

Structure

3.1 INTRODUCTION

The entity-relationship (E-R) model is a high-level data model. It is based on a perception of a real world that consists of a collection of basic objects, called entities, and of relationships among these objects. It was developed to facilitate database design by allowing specification of an enterprise schema, which represent the overall logical structure of a database.

The ER model is beneficial for a database designer in various ways, which are:

- The constructs used in the E-R model can easily be transformed into relational tables.
- Simple and easy to understand and is, therefore used by a database designer to communicate a database design to the users according the database.
- It is used as a design plan by a database developer to implement a data model in database management system.

3.2 OBJECTIVES

At the end of this unit, you should be able to:

- Understand the objective of an entity relationship diagram.
- Know that an ER diagram is a logical representation of an organization's data, and consists of three primary components: Entities, Attributes and relationships.
- Discuss different Mapping constraints.
- Know the keys that will be used to establish the relationships in the database.
- Understand how to prepare ER diagram for organizational data and convert ER diagram into tables.

3.3 DATABASE DESIGN AND ER DIAGRAMS

The database design process can be divided into three steps.

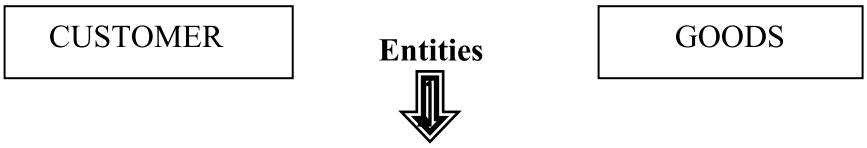
1. Requirement Analysis: The first step in designing a database application is to understand what data is to be stored in the database, what application must be built in top of it, and what operations are most frequent and subject to performance requirements. In other words, we must find out what the users want from the database.

2. Conceptual database Design: The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints known to hold over this data. The ER model is one of several high level or semantic, data models used in database design.

3. Logical Database Design: We must choose a database to convert the conceptual database design into a database schema in the data model of the chosen DBMS. Normally we will consider the Relational DBMS and therefore, the task in the logical design step is to convert an ER schema into a relational database schema.

3.3.1. Entity: Entities are specific objects or things in the mini-world that are represented in the database. It includes all those "things" about which data is collected. An entity may be a tangible object such as a employee, student, a place or a part. It may also be non-tangible such as an event, a job title or a customer account. For example if we say that a customer buys goods, it means customer and goods are entities.

Pictorially, entities are represented in rectangles.



An Entity Set: An entity set is a set of entities of the same type that share the same properties. The set of all persons who are customer at a given bank can be defined as the entity set customer.

3.3.2. Attributes: An entity is represented by a set of attributes, that is, descriptive properties possessed by all members of an entity set. Attributes are properties used to describe an entity**.** In a database, entities are represented by tables and attributes by columns. For example an EMPLOYEE entity may have a Name, Address, Sex, BirthDate and

Department entity may have a DepartmentName, DepartmentNo, DepartmentLocation and Address field shown in Figure 31.

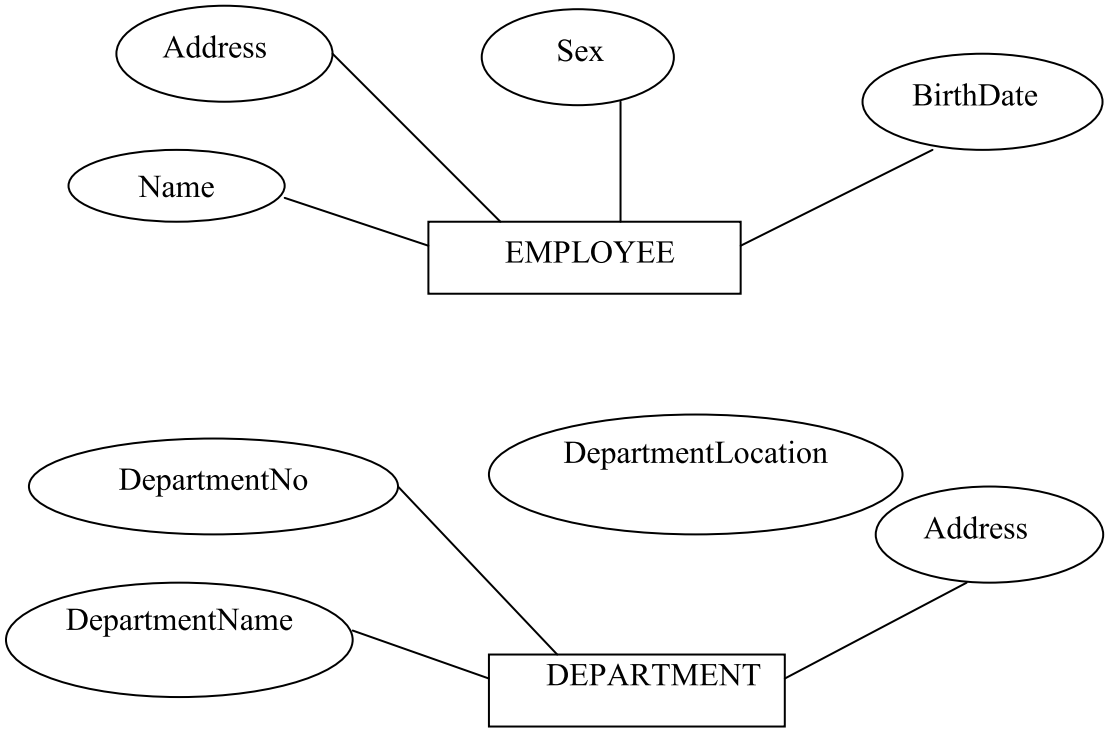


Fig 3.1: Attributes for entity EMPLOYEE and DEPARTMENT

3.3.3 Types of Attributes:

(1) Simple (Atomic) and Composite attributes

(2) Single Valued and Multi Valued attributes

(3) Stored and Derived Attributes

(4) Null Values

(1) Simple (Atomic) and Composite Attributes: Attribute that are not divisible are called simple or atomic attribute. For example street-number, street-name etc. Whereas composite attributes are those attributes which can be divided into sub-parts (i.e. other attributes). For example an attribute name could be considered as a composite attribute consisting of first name, middle name, and last name. For example:

Name (firstname, middlename, lastname) and similarly Address (HouseNo, Street, City, State, ZipCode, Country) are composite attribute.

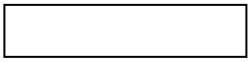
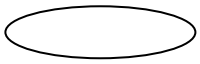
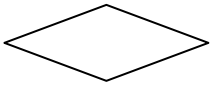
(2) Single-valued and Multi-valued Attributes: An attribute having only one value for a particular entity is called single-valued attributes, whereas attributes may have multiple values for a specific entity is called multi-valued attributes. For example the loan number attributes for a specific loan entity refers to only one loan number, such type of attributes are single-valued whereas phone number attributes, where the employees may have zero, one or several phone numbers. This type of attributes is said to be multi-valued.

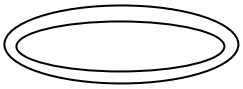
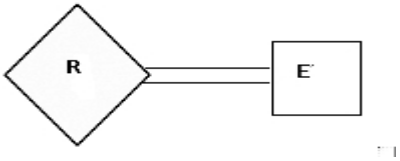
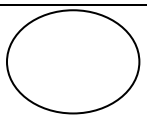
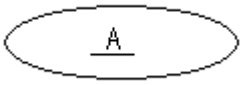
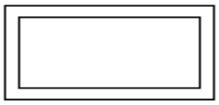
(3) Stored and Derived Attributes: The value for this type of attribute can be derived from the values of other related attributes or entities. For example the age and date of birth of a person. For a particular person entity, the value of age can be determined from the current (today's) date and the value of that person's Birthdate. The Age attribute is hence called a **derived attribute** and is said to be derivable from the Birthdate attribute, which is called **stored attribute.**

(4) Null Values: In some cases a particular entity may not have an applicable value for an attribute. For example, a college degree attribute applies only to persons with college degrees. For such situation, a special value called **null** is created.

3.3 ER Diagram (ERD):

Entity relationship diagram is the graphical representation of the E-R model. You can describe logical structure of a database graphically by using ERD. The main elements of Entity relationship diagram are:

Symbol	Meaning	Description
	Entity Set	Rectangle: it represents entity sets. The label of a rectangle is the name of an entity.
	Attributes	Ellipse: it represents attributes.
	Relationship Sets.	Diamond: it represents relationship sets.
	Link	Lines: it represents link between attributes to

		entity sets and entity sets to relationship sets. A solid line between two entity sets represents a relationship.
	Multivalued Attributes	Double ellipse: it represents multivalued attributes
	Derived Attributes	Dashed ellipse: it represents derived attributes.
	Total participation of entity set in relationship	Double line: it represents participation of an entity E in a relationship set R.
	Optional Entity	Circle: it represents an optional entity.
	Primary Key Attribute	Underline: it indicates primary key attribute
	Weak entity sets.	Double rectangles represent weak entity sets.

For example, Figure 3.3 shows the entity relationship diagram representing the Person, Car and Accident entities and the relationship between them.

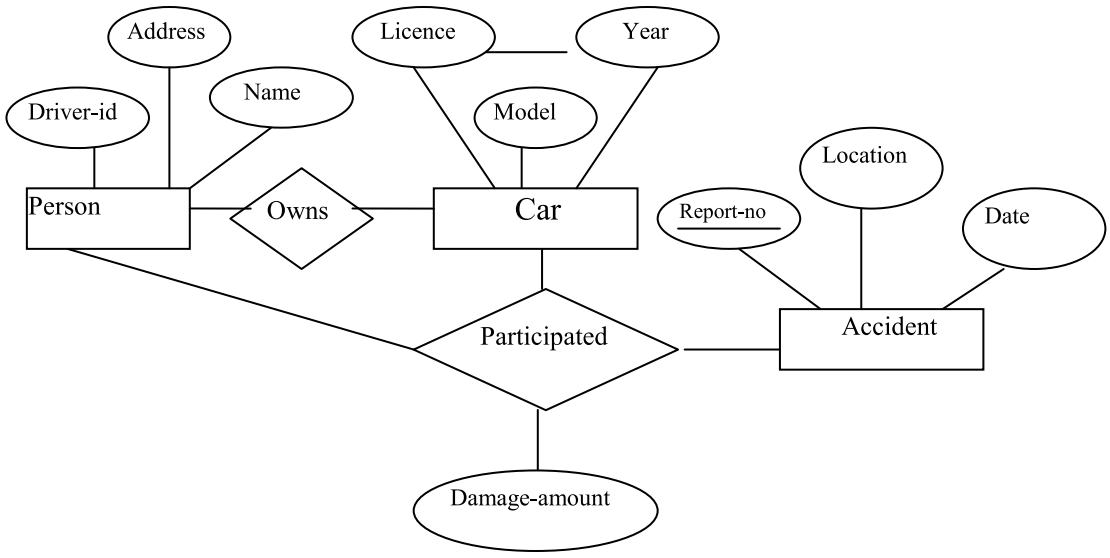


Fig 3.3: E-R diagram for a Car insurance company

3.4 Weak and Strong Entity Set: An entity set that does not posses sufficient attributes to form a primary key is called a weak entity set. One that does have a primary key is called a strong entity set. You can also represent a week entity set using entity relationship diagram. Figure 3.4 shows representation of a week entity set.

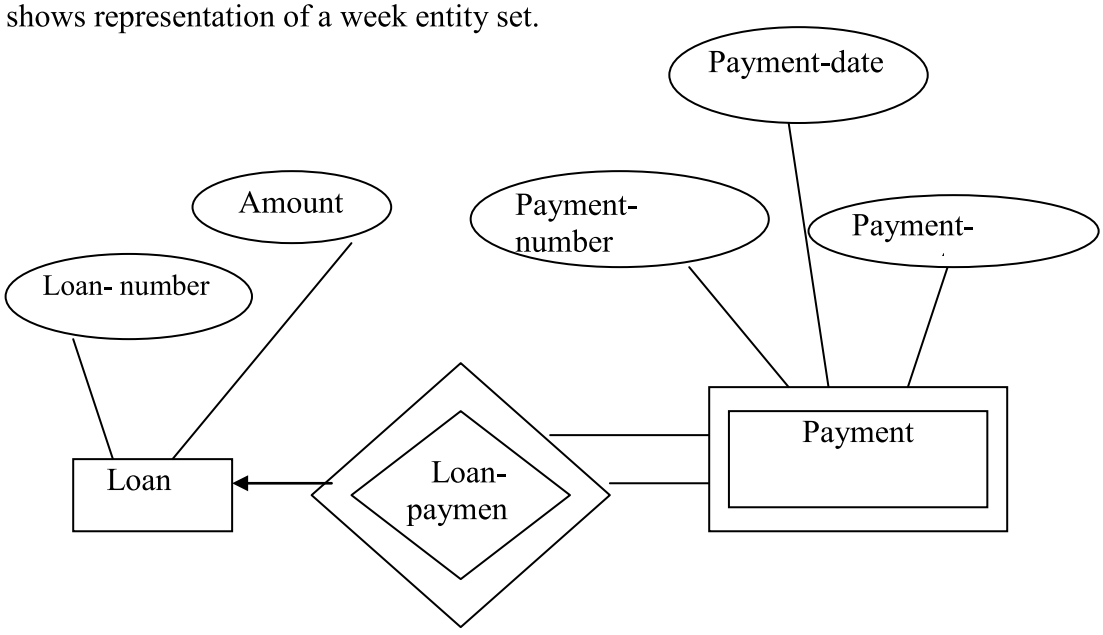


Fig 3.4 E-R diagram with a weak entity set

In figure 3.4, payment loan is the week entity set with attribute Payment-number, Payment-date and Payment-amount. This implies that the week entity set is dependent on the entity set, loan through the relationship set Loan-payment. The relationship between the week entity set, Payment and the entity set Loan is represented by using double outlined diamond for the relationship set Loan-payment. The entity relationship diagram also represents the use of double lines, which indicate total participation. Total participation of the entity set Payment represents that each payment is related to a Loan using the relationship set Loan-payment.

"CHECK YOUR PROGRESS 1"

Q1. List the various types of attributes of an entity set.
Q2. What is Entity-Relationship diagram?
Q3. What is weak and strong entity set?
Q4. Define main elements of Entity-Relationship diagram.
Q5. What is entity and attribute? Give some examples of entites and attributes in a manufacturing environment. Why are relationships between entites important?

3.5 MAPPING OR CARDINALITY CONSTRAINTS: To understand cardinality constraints, you need to consider binary relationship set R between entity sets A & B. Mapping cardinalities or cardinality ratios express the number of entities to which another entity can be associated via a relationship set. Mapping cardinalities are most useful in describing binary relationship sets. For a binary relationship set, the mapping cardinality must be one of the following types:

1. One-to-one
2. One-to-many
3. Many-to-one
4. Many-to-many

(1) One-to-one(1:1) : An entity in A is associated with almost one entity in B, and an entity in B is associated with almost one entity in A. in Figure 3.5 Employee and Department are two entities, an employee in the organization belongs to a single department. Thus this relationship represents a one-to-one relationship.

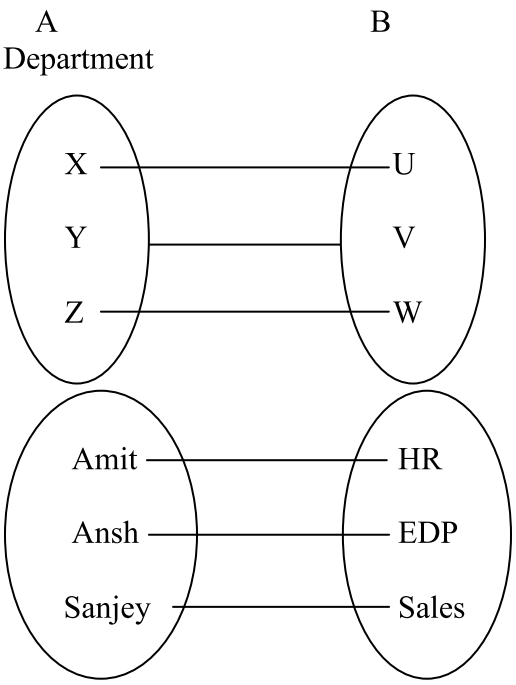


Fig. 3.5 One-to-one relationship

(2) One-to-many(1:N) : An entity in A is associated with any number (zero or more) of entities in B. in entity in B however can be associated at most one entity in A. In Figure 3.6, each department can have many employees but an employee is related to a single department only. The relationship between the department and employee is 1:N relationship.

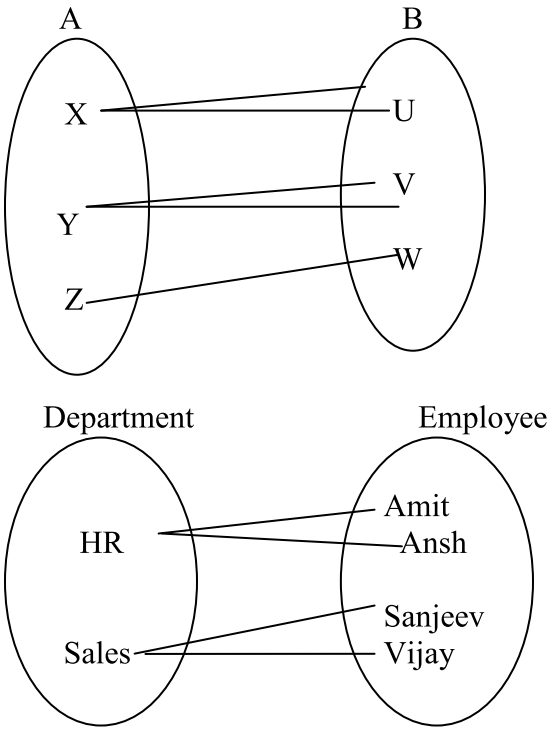


Fig. 3.6 One-to-many relationship

(3) Many-to-one (M: 1): An entity in A is associated with at most one entity in B. An entity in B however can be associated with any number (zero or more) of entities in A.

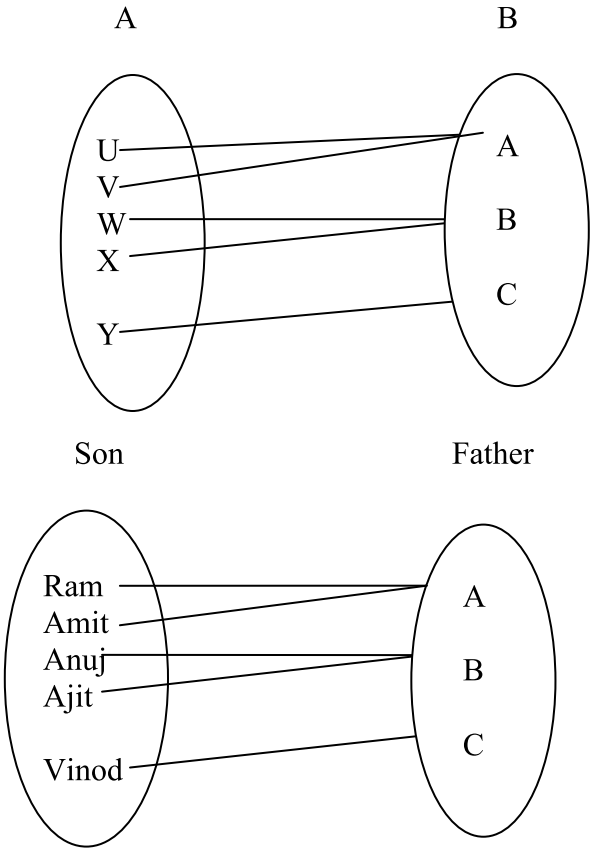


Fig. 3.7 Many to one relationship

Figure 3.7 shows many to one relationship between Son and Father. A Father may have many sons but a son has only one Father.

(5) **Many-to-many (M: N):** 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. Figure 3.8 shows a many-to-many relationship between Employee and Project. An employee may be assigned to many projects; each project must have many employee.

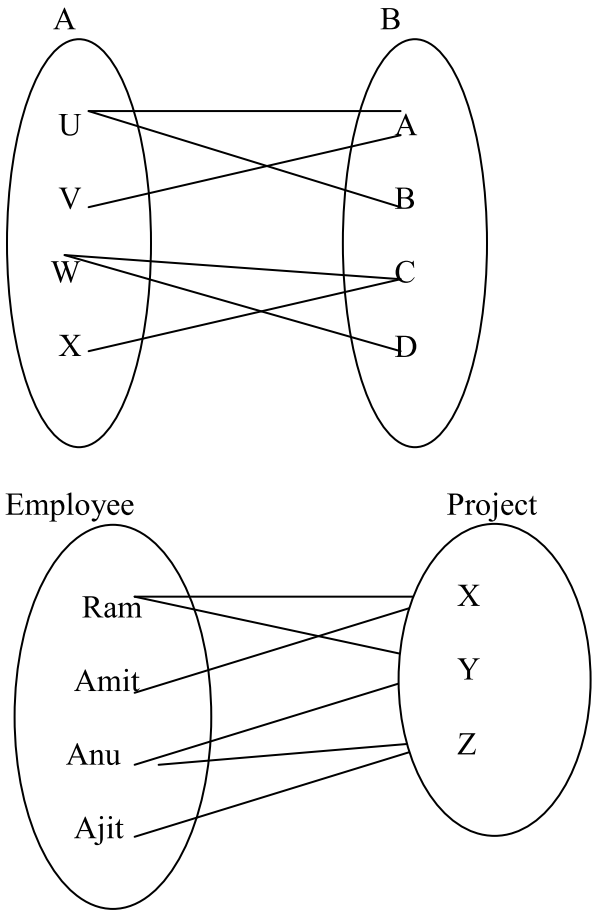


Fig 3.8 Many-to-many

3.6 RELATIONSHIPS AND RELATIONSHIP SET

A relationship is an association among two or more entities. For example, an employee **Works_at** a department, a student **Enrolls** in a course. Here, Works_at and Enrolls are called relationships.

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called descriptive attributes**.**

3.6.1 Degree of a Relationship: The degree of a relationship is the number of participating entity types. A relationship type of degree two is called **Binary,** a degree three is called **Ternary,** and degree four is **quaternary.**

Example for Binary Relationship

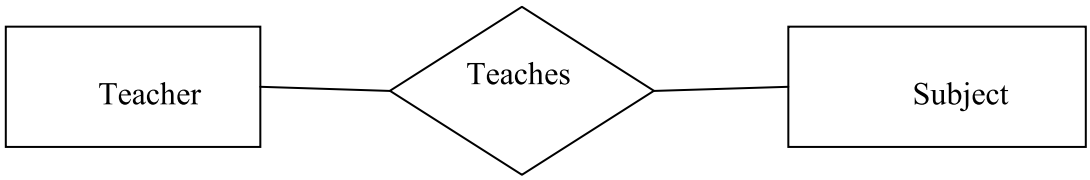


Fig. 3.9: Binary Relationship

In Figure 3.9, Teaches is the binary relationship between entities Teacher and Subject.

Example for Ternary Relationship

In Figure 3.10, The University might need to record which teacher taught which subjects in which courses. So, CST is the ternary relationship between entities Course, Subject and Teacher.

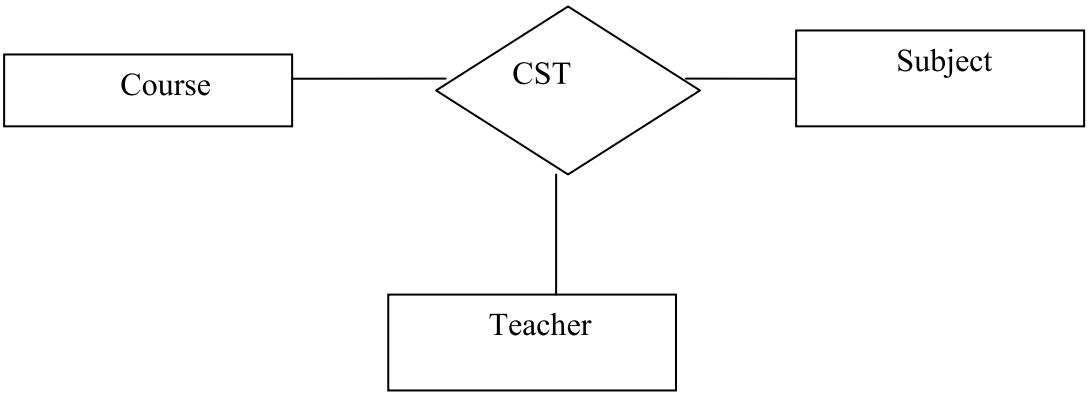


Fig 3.10: Ternary Relationship

Example for quaternary Relationship

In Figure 3.11, a Solicitor arranges a bid on behalf of a buyer supported by a financial Institution

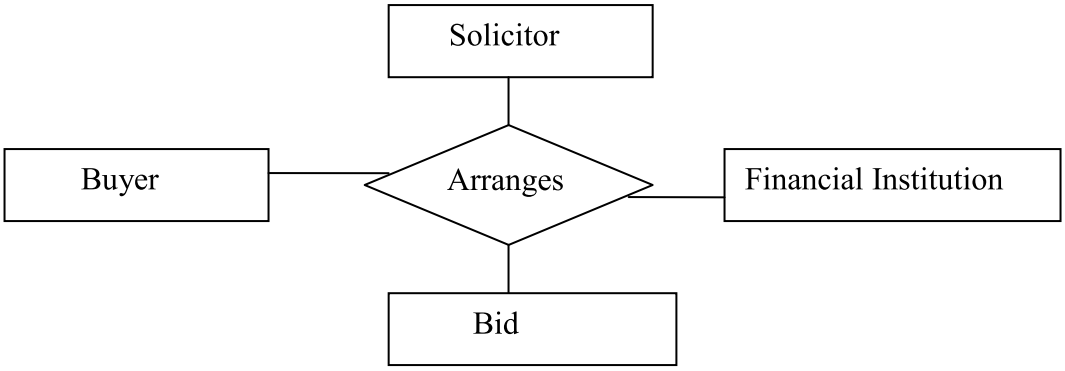


Fig. 3.11: Quaternary Relationship

3.7 KEY

The integrity of the information stored in a database is controlled by keys. A key or key field is a column value in a table that is used to either uniquely identify a row of data in a table or establish a relationship with another table.

In database management system, a key is a field that you use to sort data that is arranging the records in ascending or descending data. It is also referred as sort key, index or key word. Most database system allows more than one key so that records can be stored in various ways. Keys can be of different types:

3.7.1 Primary Key: A primary key is a column (or columns) in a table that uniquely identifies the rows in that table. For example in the STUDENT table of Table 3.1, RollNo is the primary key as each student have unique roll number.

Roll No	First Name	Last Name
1	Amit	Gupta
2	Sanjeev	Gangwar
3	Harshit	Gupta
3	Ajay	Singh

Table 3.1: STUDENT

The values placed in primary key columns must be unique for each row thus no duplicates can be tolerated. In addition, nulls are not allowed in primary key columns.

3.7.2 Super Key: Super key is the combination of more than one attribute that is used to identify every tuple (row) of the table uniquely. A table may contain more than one super keys depending on the possible combinations of the attributes in the table. You can use primary key of a table to make a super key. For example, different combinations of RollNo, Name, Branch, DOB and Year attributes of STUDENT table of Table 3.2 form different super keys for STUDENT table.

RollNo	Name	Branch	Dob	Year
1	Amit Gupta	CS	12/07/1982	I
2	Sanjeev Gangwar	IT	25/10/1981	II
3	Harshit Gupta	EC	06/04/1983	III
3	Ajay Gupta	EE	28/07/1980	II

Table 3.2: STUDENT

Now from the above table

Super Key1= {RollNo, Name, Branch}

Super Key2= {Name, Branch, Dob}

Super Key3= {Name, Dob, Year}

3.7.3 Candidate key: In the relational model of databases, a **candidate key** of a relation is a minimal superkey for that relation; that is, a set of attributes such that

1. The relation does not have two distinct tuples (i.e. rows or records in common database language) with the same values for these attributes (which means that the set of attributes is a superkey)
2. There is no proper subset of these attributes for which (1) holds (which means that the set is minimal).

The constituent attributes are called **prime attributes**. Conversely, an attribute that does not occur in ANY candidate key is called a **non-prime attribute**.

Note: - A combination of one or more fields whose value uniquely identifies a record in a table. That is, no two records in a table can have the same key value. Every field is a candidate key but there cannot be more than one primary key in a table.

From the above student table.

Candidate key1= {RollNo, Name}

Candidate key2= {Name, Dob}

Candidate key3= {Name, Branch}

3.7.4 Foreign Key: Foreign key is that column of the table, which is used to maintain relationship with other tables of database. Foreign key of a table should be defined as primary key in another table. For example, Stud_Activity is an attribute of Student table of Table 3.3 that is primary key of another table Activity which is shown in Table 3.4

Stud_Roll	Stud_name	Stud_class	Stud_activity
101	Amit	Twelfth	Cricket
102	Vijay	Eleventh	Football
103	Sanjeev	Twelfth	Hockey

Table 3.3: Student

Stud_activity	No_of_participants	Incharge_of_activity
Cricket	40	Sushil
Football	30	Yogendra
Hockey	20	Ankit

Table 3.4: Activity

In the above table Stud_activity is the primary key of the Activity table . Stud_activity attribute of Activity table is inherited in the Student table. Stud_activity is used to show records of participants of different activities in Student table. Thus you can use this attribute to build a relation between Student table and Activity table.

3.7.5 Alternate Key: Any candidate key(s) other than the one chosen as a primary key is known as alternate key.

3.7.6 Composite Key: Composite key is a key composed of more than one column. Sometimes, it is also known as concatenated key or structured key.

"CHECK YOUR PROGRESS 2"

Q1. Define a relation.

Q2. Differentiate between primary key & candidate key.

Q3. Explain degree of a relationship.

Q4. Define cardinality constraints.

Q5. What are the two types of constraints in E.R diagram? Explain.

3.8 EXTENDED E-R MODEL

The extended entity relationship model (EER model) includes all the modeling concepts of the E-R model and the concepts of specialization, generalization, attributes inheritance and aggregation.

There are basically four concepts of EER-model

- Specialization
- Generalization
- Attribute Inheritance
- Aggregation

3.8.1 Specialization (top down design process):

Specialization is the process of taking subsets of the higher-level entity set to form lower-level entity sets. It is a process of defining a set of subclasses of an entity type, which is called the super class of the specialization. The process of defining subclass is based on the basis of some distinguish characteristics of the entities in the super class.

For example suppose the bank wishes to divide accounts into two categories, checking account and saving account- savings account need a minimum balance but the bank may set interest rates differently for

different customers, offering better rates to favored customers. The bank could then create two specializations of account, namely saving account and checking account.

We can apply specialization repeatedly to refine a design schema. For instance, bank employees may be further classified as one of the following:

- Officer
- Teller
- Secretary

Each of these employee types is described by a set of attributes that includes all the attributes of entity set employee plus additional attributes. For example, Officer entities may be described further by the attribute office-number, teller entities by the attributes station-number and hours-per-week, and secretary entities by the attribute hours-per- week. Further. In terms of an E-R diagram, specialization is depicted by a triangle component labeled ISA. The label ISA stands for "is a" and represents, for example, that a customer "is a" person shown in Figure 3.13. The ISA relationship may also be referred to as a superclass-subclass relationship. Higher and lower-level entity sets are depicted as regular entity sets — that is, as rectangles containing the name of the entity set

3.8.2 Generalization (bottom-up design process):

A generalization hierarchy is a form of abstraction that specifies that two or more entities that share common attributes can be generalized into a higher-level entity type called a super type or generic entity. The lower level of entities becomes the subtypes, or categories, to the super type. Subtypes are dependent entities.

Generalization is used to emphasize the similarities among lower-level entity set and to hide differences. It makes ER diagram simpler because shared attributes are not repeated. Generalization is denoted through a triangular component labeled 'IS A".

Differences in the two approaches may be characterized by their starting point and overall goal. Specialization stems from a single entity set; it emphasizes differences among entities within the set by creating distinct lower-level entity sets. These lower-level entity sets may have attributes, or may participate in relationships, that do not apply to all the entities in the higher-level entity set. Indeed, the reason a designer applies specialization is to represent such distinctive features. If Customer and Employee neither have attributes that person entities do not have nor participate in different relationships than those in which Person entities participate, there would be no need to specialize the person entity set.

Generalization proceeds from the recognition that a number of entity sets share some common features (namely, they are described by the same attributes and participate in the same relationship sets). On the basis of their commonalities, generalization synthesizes these entity sets into a single, higher-level entity set. Generalization is used to emphasize the similarities among lower-level entity sets and to hide the differences; it also permits an economy of representation in that shared attributes are not repeated.

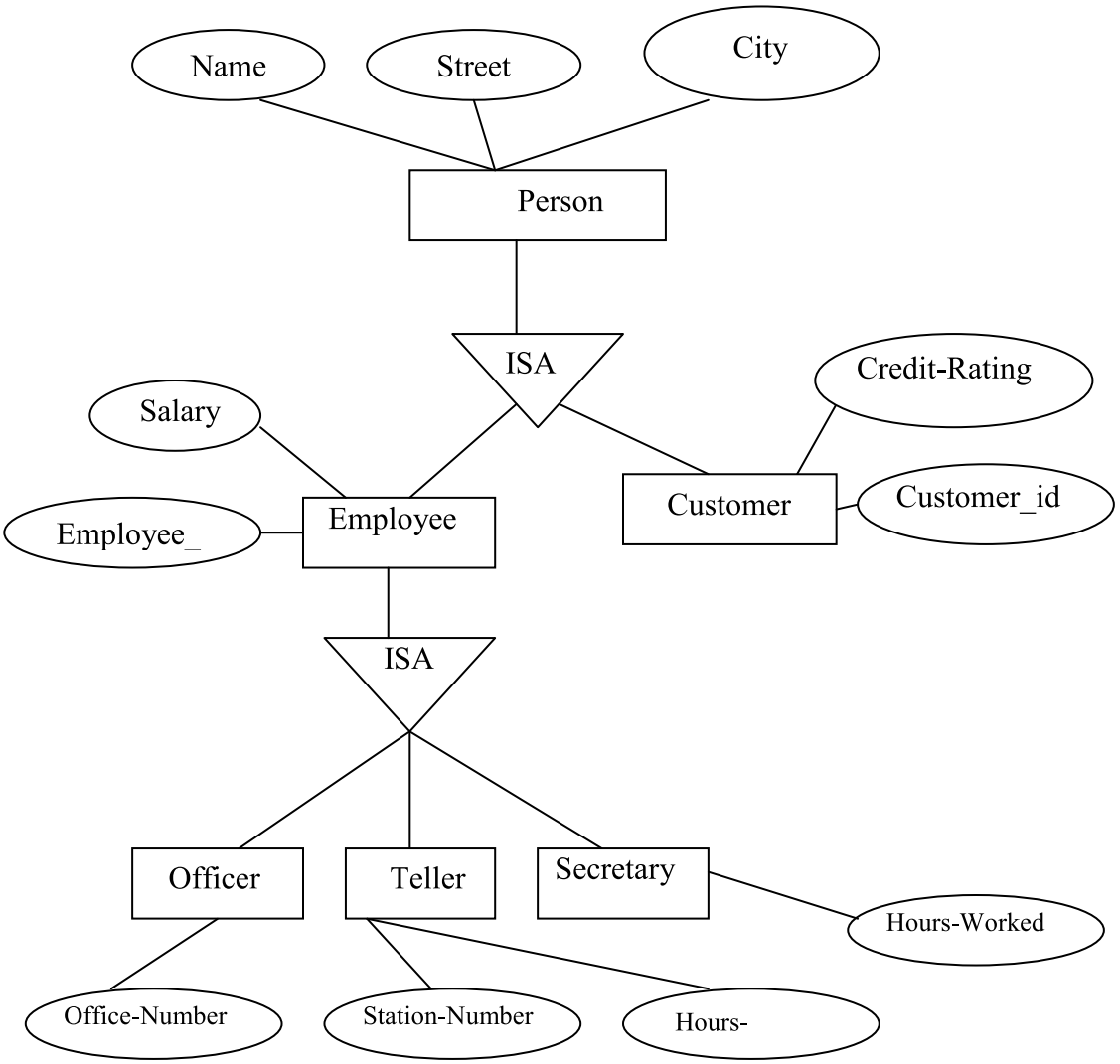


Fig. 3.13 E-R diagram with generalization and specialization

3.8.3 Attribute Inheritance

A crucial property of the higher- and lower-level entities created by specialization and generalization is **attribute inheritance**. The attributes

of the higher-level entity sets are said to be **inherited** by the lower-level entity sets. For example, customer and employee inherit the attributes of person. Thus, customer is described by its name, street, and city attributes, and additionally a Credit-Rating attribute; employee is described by its name, street, and city attributes, and additionally employee-id and salary attributes.

Figure 3.13 depicts a **hierarchy** of entity sets. In the figure, employee is a lower-level entity of set of person and a higher-level entity set of the officer, teller, and secretary entity sets. In a hierarchy, a given entity set may be involved as a lower-level entity set in only one ISA relationship; that is, entity sets in this diagram have only **single inheritance**. If an entity set is a lower-level entity set in more than one ISA relationship, then the entity set has **multiple inheritance**, and the resulting structure is said to be a "Lattice".

3.8.4 Aggregation: Aggregation is the process of compiling information on an object, thereby abstracting a higher-level object. One limitation of the E-R model is that it cannot express relationships among relationships. Aggregation is a process when relation between two entity is treated as a single entity. As shown in Figure 3.14, the relation between Center and Course is acting as an Entity in relation with Visitor.

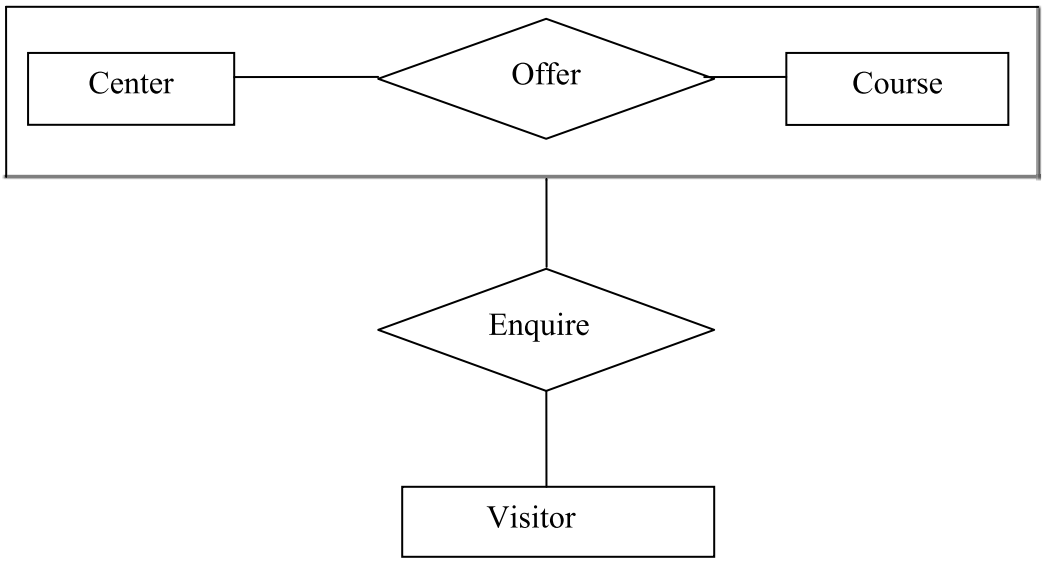


Fig. 3.14 E-R diagram with aggregation

3.9 REDUCTION OF E-R DIAGRAMS INTO TABLES

Database can be represented with the help of E-R notations and these notations (diagrams) can be reduced to collection of tables. For every

entity set, relationship set in a database can be represented in tabular form. The name of the table is the name of corresponding entity set of relationship name. Each table would have columns, the number of columns would be equal to the number of attributes each entity set or relationship set have.

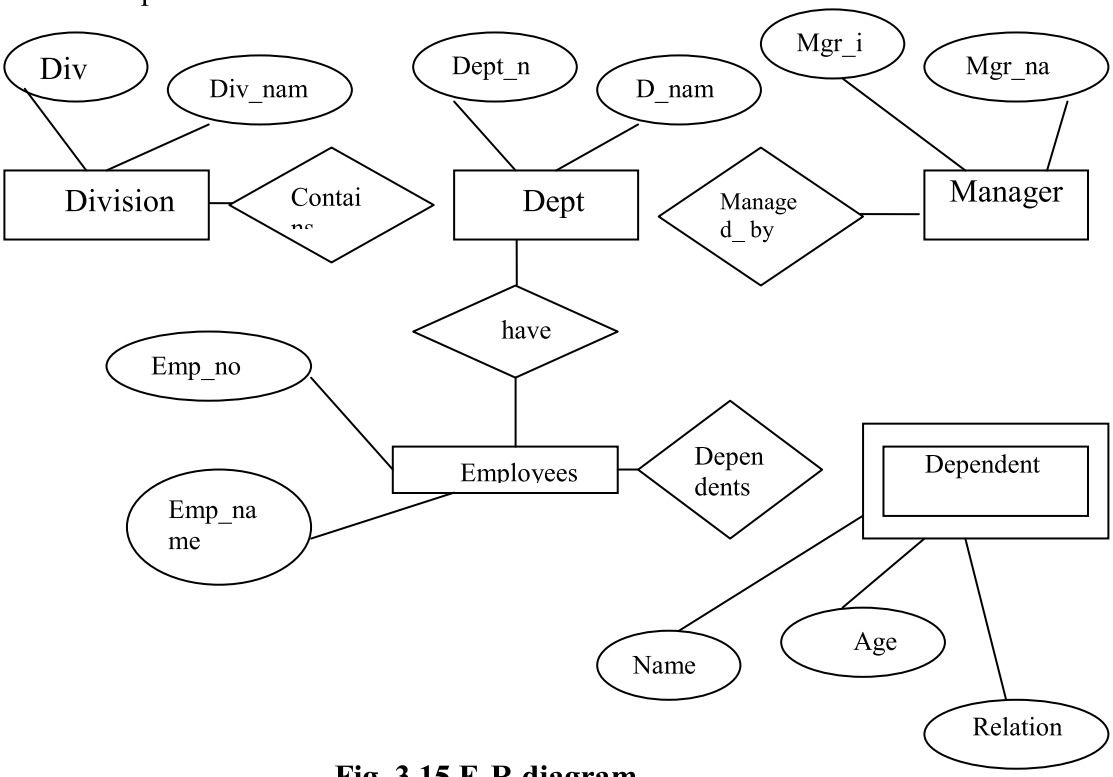


Fig. 3.15 E-R diagram

The ER diagram of above problem is divided into three sections

(1) Strong entity sets(Division, Dept, manager, Employees)
(2) Weak entity set (Dependents)
(3) Relation sets(contains, managed_by, have, dependents_of)

(1) Reducing Strong Entity Sets into Tables: Consider the entity sets Division of the E-R diagram of Figure 3.15. This entity set have two attributes Div_id and Div_name. We can represent this entity set by a table called Division with two columns.

Div_id	Div_name
D1	Engineering
D2	Pharmacy

Table: Division

Similarly entity set 'Dept' have attributes Dept_no and D_name, which will be represented by table Dept.

Dept_no	D_name
Dept01	Computer Science
Dept02	Information Technology

Table: Dept

Entity set Manager has also two attributes Mgr_id amd Mgr_name which will be represented by table Manager and entity set Employee have two attributes Emp_no and Emp_name which will be represented by table employee.

Mgr_id	Mgr_name
MG01	Ravi
MG02	Vijay

Table: Manager

Emp_no	Emp_name
EMP01	Dinesh
EMP02	Akshay
EMP03	Suresh

Table: Employee

(2) Reducing Weak Entity Sets into Tables: In E-R Diagram shown in Figure 3.15 we have only one weak entity set Dependent. This entity set has three attributes Name, Age, and Relation. The entity set 'Dependent' is represented by table 'Dependent' shown below

Table Dependent

Name	Age	Relation
Manu	34	Father
Ashish	12	Brother

(3) Reducing Relationship sets into Tables: In E-R diagram shown in Figure 3.15 we have four relationship sets **Contains, Managed by, have, Dependents_of.**

Contains relationship set involves two entity sets Division with primary key Div_id and Dept with primary key Dept_no.

Div_id	Dept_no
D1	Dept01
D2	Dept02

Table: Contains Relation

Managed_by relationship set involves two entity sets Dept with primary key Dept_no and manager with primary key Mgr_id. Thus managed_by relationship table contains attributes Dept_no of entity set Dept and Mgr_id of entity set Manager.

Dept_no	Mgr_id
Dept01	MG01
Dept02	MG02

Table: Managed_by relation

have relationship set involves two entity sets Dept with primary key Dept_no and Employee with primary key Emp_no. Thus have relationship table consist of primary key of Dept and Employee.

Dept_no	Emp_no
Dept01	EMP01
Dept02	EMP02

Table: have relation

3.10 RELATIONSHIPS OF HIGHER DEGREE

Degree of relationship refers to the number of participating entities in a relationship. If there are two entities involved in relationship then it is referred to as binary relationship. If there are three entities involved then it is called as ternary relationship and so on. It is represented by a diamond symbol. All relation has three components:

As shown in Figure 3.16 customer & Goods are entities. Customer entity has three attributes Code, Name & address with Primary key Code. Entity Goods have two attributes Code & Price. Relationship exists between entity Customer & Goods is Goods Bought.

(1) **Name:** It is the title or entity identifier such as Goods Bought shown in Figure 3.16.

(2) **Degree:** It represents the number of attributes (field) associated with the table or relation.

(3) **Cardinality:** The **degree of relationship** (also known as cardinality) is the number of occurrences in one entity which are associated (or linked) to the number of occurrences in another.

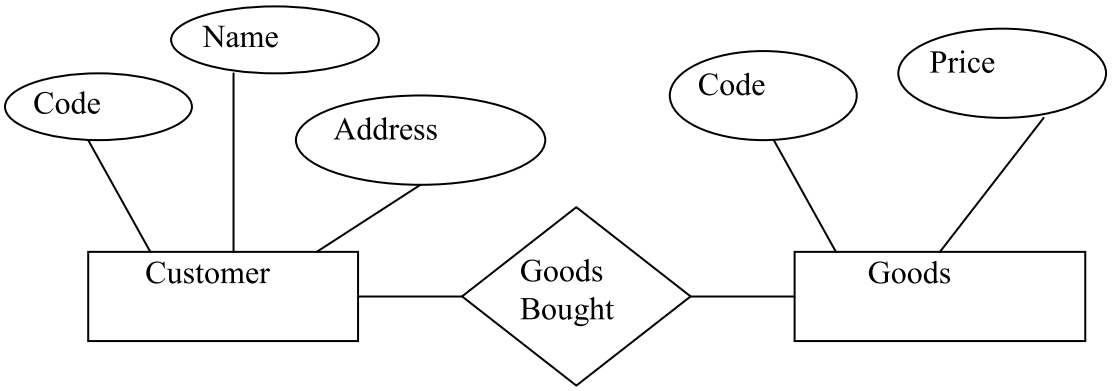


Fig. 3.16 Entities, Attributes, relationships

Relationships are of three types: one-to-one, one-to-many and many-to-many.

3.10.1 One-to-one Relationship (1:1): In one-to-one relationship, one record in a table is related to only one record in another table. For example in Figure 3.17, one student can enroll only for one course and a course will also have only one Student.

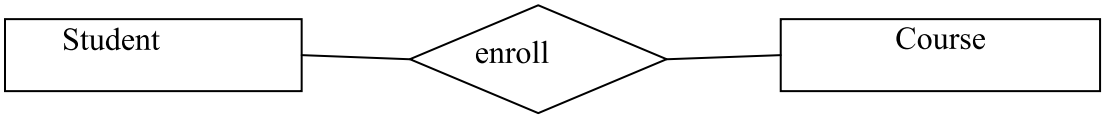


Fig. 3.17 One-to-one relationship

3.10.2 One-to-many Relationship (1: M): In one-to-many relationship, one record in a table (parent table) can be related to many records in another table (child table) .For example in Figure 3.18, a father may have more than one child but the child has only one father.

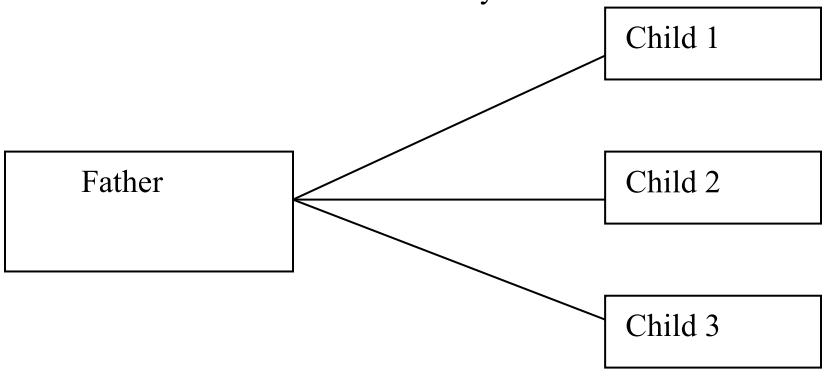


Fig. 3.18 One-to-many relationship

3.10.3 Many-to-many Relationship (M: M): In many-to-many relationship, one record in a table can be related to one or more records in another table, and one or more records in the second table can be related to one or more records in the first table. For example in Figure 3.19, an employee may work on several projects at the same time and a project has a team of many employees.

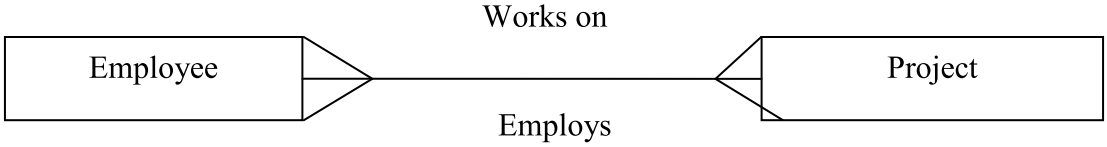


Fig. 3.19 Many-to-many relationship

"CHECK YOUR PROGRESS 3"

Q1. Explain Extender ER model.

Q2. Define the concept of aggregation. Give an example of where this concept is useful.

Q3. Explain multivalued dependencies with example.

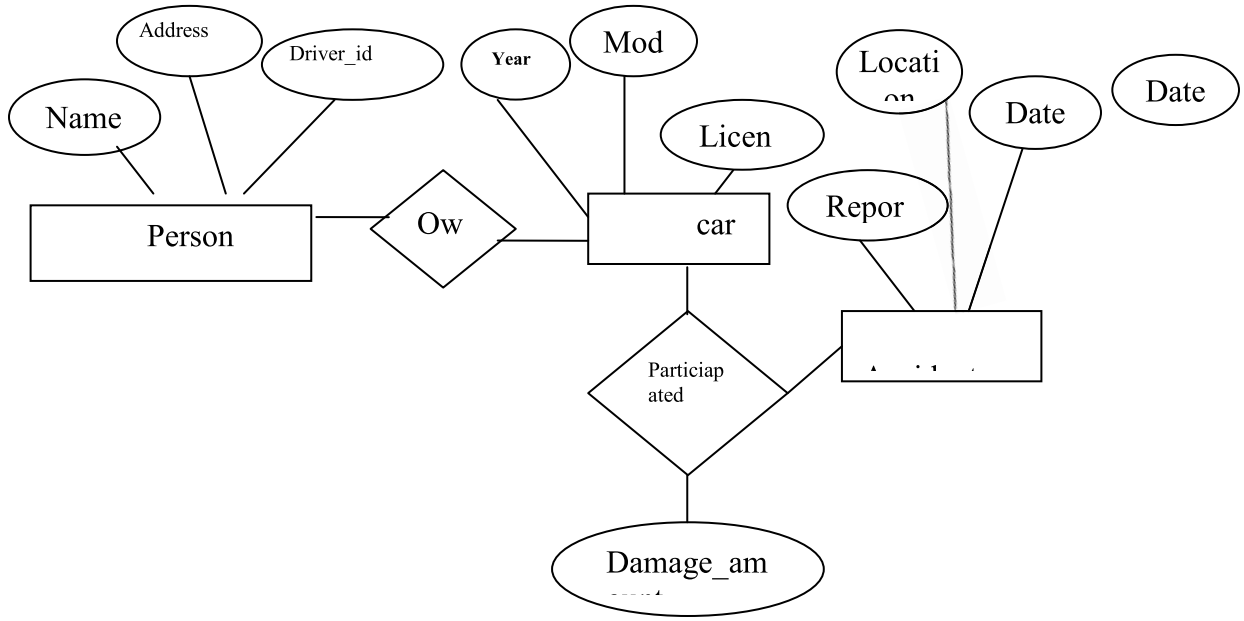
Q4. Discuss the naming conventions used for ER schema diagrams.

Q5. Construct an ER diagram for university registrar's office. The office maintains data about each class, including the instructor, the enrollment and the time and place of the class meetings. For each student class pair a grade is recorded. Determine the entities and relationships.

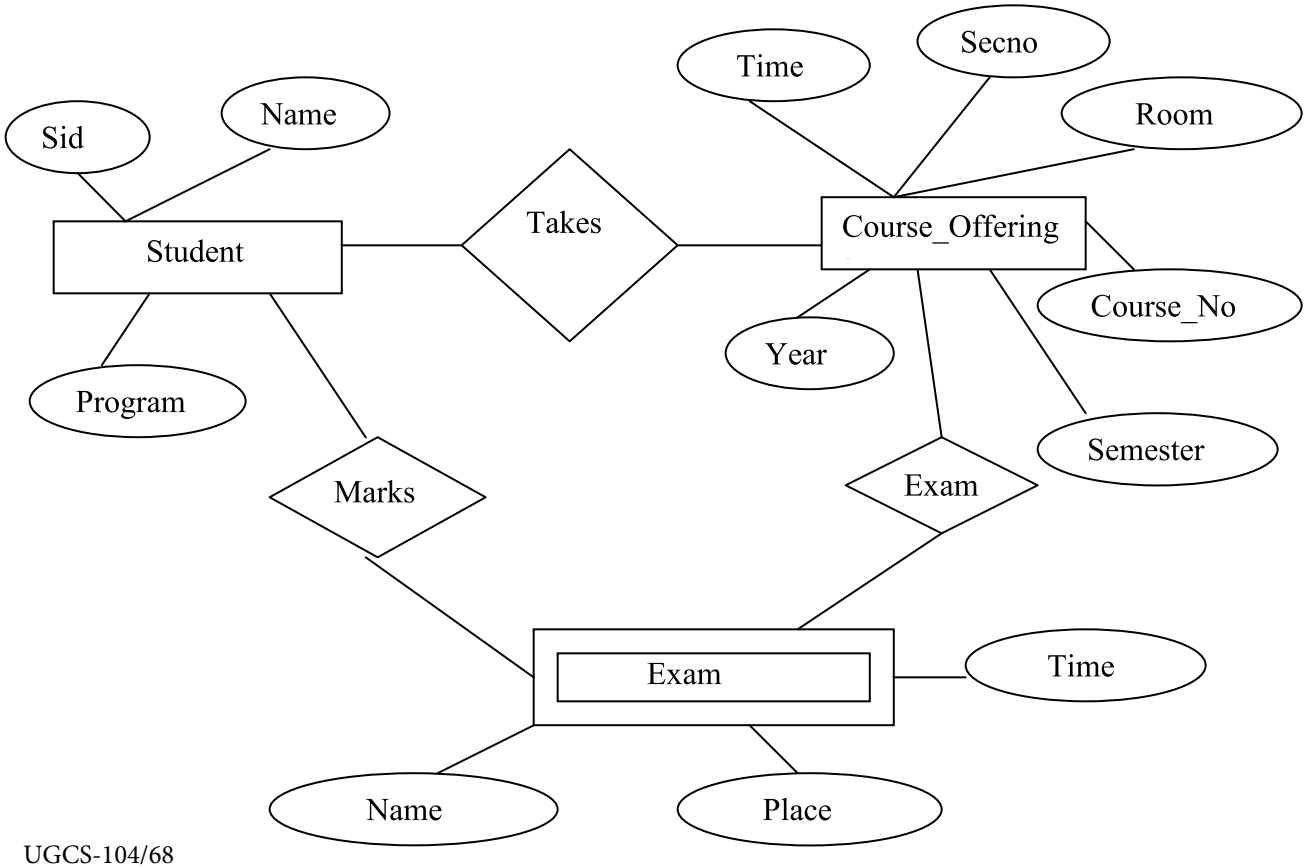
Q6. Draw an E-R diagram for a hospital with a set of patients and a set of medical doctors, with each patient a log of the various conducted tests is also associated.

3.11 E-R Examples

(1) 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.



(2) Construct an E-R diagram that used only a binary relationship between students amd course-offering. Make sure that only one relationship exists between a particular student and course-offering pair. Yet you can represent the marks that a student gets in different exams of a course offering.



3.12 SUMMERY

- The Entity-Relationship model is a high level data model based on a perception of a real world that consists of a collection of basic objects, called entities and of relationships among these objects.
- An entity can be define as a 'thing' or 'object' in the real world that is distinguishable from all other objects, whereas an entity set is a set of entities of the same type that share the same properties or attributes.
- Attributes are the descriptive properties possessed by each member of an entity set.
- The domain of an attribute is the collection of all possible values, and attribute can have.
- Mapping constraints express the number of entities to which another entity can be associated via a relationship set.
- A key is a value which can always be used to uniquely identify an object instance. It is of five types: super key, candidate key, primary key, composite key and foreign key.
- Generalization is the abstracting process of viewing set of objects as a single general class by concentrating on characteristics of lower level entities while ignoring there is different.
- Aggregation is a technique to express relationship among relationship.
- Specialization is the process of designing sub-groupings with an entity set.

3.13 OBJECTIVE TYPE QUESTIONS

Choose the correct or best alternative in the following:

Q1. Key to represent relationship between tables is called

(A) Primary key (B) Secondary Key
(C) Foreign Key (D) None of these

Ans: C

Q2. What is a relationship called when it is maintained between two entities?

(A) Unary (B) Binary
(C) Ternary (D) Quaternary

Ans: B

Q3. The RDBMS terminology for a row is

(A) tuple. (B) relation.
(C) attribute. (D) degree.

Ans: A

Q4. Which are the two ways in which entities can participate in a relationship?

(A) Passive and active **(B)** Total and partial
(C) Simple and Complex **(D)** All of the above

Ans: B

Q5. Which of the following is another name for weak entity?

(A) Child **(B)** Owner
(C) Dominant **(D)** All of the above
Ans: A

Q6. A primary key is combined with a foreign key creates

(A) Parent-Child relation ship between the tables that connect them.

B) Many to many relationship between the tables that connect them.

(C) Network model between the tables that connect them.
(D) None of the above.

Ans: A

Q7. In E-R Diagram derived attribute are represented by

(A) Ellipse **(B)** Dashed ellipse
(C) Rectangle **(D)** Triangle

Ans B

Q8. An instance of relational schema R (A, B, C) has distinct values of A including NULL
values. Which one of the following is true?

(A) A is a candidate key **(B)** A is not a candidate key
(C) A is a primary Key **(D)** Both (A) and (C)

Ans: B

Q9. A primary key if combined with a foreign key creates

(A) Parent-Child relationship between the tables that connect them.
(B) Many to many relationship between the tables that connect them.
(C) Network model between the tables that connect them.
(D) None of the above.
Ans: A

Q10. In E-R Diagram relationship type is represented by

(A) Ellipse **(B)** Dashed ellipse

(C) Rectangle **(D)** Diamond

Ans: D

3.14 SELECTED EXERCISES

1. Discuss briefly the convention for displaying an E-R diagram.
2. What is the difference between specialization and Generalization with respect to database?
3. Explain difference between a weak and a strong entity set.
4. Explain Primary key, Candidate key, super key and foreign key.
5. Explain the mapping constraints in detail.
6. Discuss the Relationship and its type.
7. Discuss the concept of aggregation.
8. Construct an E-R diagram for hospital management system.
9. Suppose you are given the following requirements for a simple database for the Indian Hockey League (IHL):

   - The IHL has many teams,
   - each team has a name, a city, a coach, a captain, and a set of players,
   - each player belongs to only one team,
   - each player has a name, a position (such as left wing or goalie), a skill level, and a set of injury records,
   - a team captain is also a player,
   - a game is played between two teams (referred to as host_team and guest_team) and has a date (such as July 25th, 2016) and a score (such as 3 to 1).

   Construct a clean and concise ER diagram for the IHL database. List your assumptions and clearly indicate the cardinality mappings as well as any role indicators in your ER diagram.

10. Construct an ER diagram for a college database. Statements are given below:

    - A college contains many departments
    - Each department can offer any number of courses
    - Many instructors can work in a department

- An instructor can work only in one department
- For each department there is a Head
- An instructor can be head of only one department
- Each instructor can take any number of courses
- A course can be taken by only one instructor
- A student can enroll for any number of courses
- Each course can have any number of students

11. An organization purchases items from a number of suppliers. Suppliers are identified by SUP-ID. It keeps track of the number of each item type purchased from each supplier. It also keeps a record of supplier's addresses. Supplied items are identified by ITEM-TYPE and have description (DESC). There may be more than one such addresses for each supplier and the price charged by each supplier for each item type is stored. Identify the entites and relationships for this organization and construct an E-R diagram. From the E-R diagram, write the scripts for creating a schema.
12. What is an entity? What is a relationship? Explain ER modeling with the help of database for a Student Management System.
13. Construct an ER diagram for university registrar's office. The office maintains data about each class, including the instructor, the enrollment and the time and place of the class meetings. For each student class pair a grade is recorded. Determine the entities and relationships.

Uttar Pradesh Rajarshi Tandon
Open University

UGCS-104

Database Management System
(DBMS)

BLOCK

2

Curriculum Design Committee

Dr. P. P. Dubey, **Coordinator**
Director, School of Agri. Sciences, UPRTOU, Allahabad
Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Allahabad
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg., MNNIT, Allahabad
Prof. P. K. Mishra **Member**
Dept. of Computer Science, Baranas Hindu University, Varanasi
Mr. Prateek Kesrwani **Member Secretary**
Academic Consultant-Computer Science
School of Science, UPRTOU, Allahabad

Course Design Committee

Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Prayagraj
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg., MNNIT, Allahabad, Prayagraj
Prof. P. K. Mishra **Member**
Dept. of Computer Science, Baranas Hindu University, Varanasi
Faculty Members, School of Sciences
Dr. Ashutosh Gupta, Director, Scool of Science, UPRTOU, Prayagraj
Dr. Shruti, Asst. Prof., (Statistics), Scool of Science, UPRTOU, Prayagraj
Ms. Marisha Asst. Prof., (Computer Science), Scool of Science, UPRTOU, Prayagraj
Mr. Manoj K Balwant Asst. Prof., (Computer Science), Scool of Science, UPRTOU, Prayagraj
Dr. Dinesh K Gupta Academic Consultant (Chemistry), Scool of Science, UPRTOU, Prayagraj

Course Preparation Committee

Mr. Sanjeev Gangwar* **Author**
Dept. of Computer Applications
VBS Purvanchal University, Jaunpur
Ms. Marisha[#] **Author**
Assistant Professor- Computer Science
School of Science, UPRTOU, Prayagraj
Dr. Ashutosh Gupta **Editor**
Director, School of Sciences,
UPRTOU, Prayagraj
Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Prayagraj
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg.,
MNNIT, Allahabad, Prayagraj
Prof. P. K. Mishra **Member**
Dept. of Computer Science
Baranas Hindu University, Varanasi
Dr. Dinesh K Gupta, **SLM Coordinator**
Academic Consultant- Chemistry School of Science, UPRTOU, Prayagraj

Note: Author's * -Block 1 and 2, # - Block-3

ISBN : 978-93-83328-29-1

UNIT-4

RELATIONAL MODEL

Structure

4.1 INTRODUCTION

The ER model is the generalization of earlier described models, hierarchical and network models. It was first proposed by CHEN in 1976 and since then so many modification have been introduced in the model.

Our goal is to provide information for the purpose of decision making in business. The database is one to provide such information**. The ER model** is normally expressed by a graphical representation called Entity Relationship Diagram (ER Diagram). E-R diagram serve the following purposes.

- They model the information needs of an organization.
- They identify entities and their relationships.
- They provide a starting point for data definition (data flow diagram).
- They provide an excellent source of documentation for application developers as

well as database and system administrators.

The various term used in ER model are Entities, Relationships and Attributes. An entity is an object that exists in real world and is distinguishable from other objects. The relational data model represents the database as a collection of relations. A relation can be seen as a table of values. There are six basic operations in relational algebra: selection, projection, cartesian product, set union, complement and rename. Some additional operators are also included in relational algebra such as join, outer join and division. Relational calculus is also used in relational algebra to retrieve the data from database. Relational calculus is divided into two categories, tuple and domain.

4.2 OBJECTIVES

After the end of this unit, you should be able to:

- Define terms related to entity relationship modeling, including entity, entity instance, attribute, relationship, cardinality, and foreign key.
- Describe the entity modeling process.
- Discuss how to draw an entity relationship diagram.
- Describe how to recognize entities, attributes, relationships, and cardinalities.
- Describe how to model supertype/subtype structures and unary relationships.
- Describe various constraints.

4.3 RELATIONAL DATABASE

Relational database is made up of two-dimensional table which is used to represent data in the form of rows and columns. The two-dimensional table in relational database is known as relation and represents a real world object. A relation describes the properties of the real world object, which it represents.

4.3.1 Entity: A collection of entities or relationships among entities. An entity is an object that exists and is distinguishable from other objects. An entity set is a set of entities of the same type that share the same properties. An entity is represented by a table in relational model. For example, employee, student etc. are entities. Entity instance refer to a particular occurrence of an entity in a table. For example, a particular employee in an organization represents an instance of the employee entity.

4.3.2 Attributes

An entity is represented by a set of attributes, that is, descriptive properties possessed by all members of an entity set. Attributes of an entity are used to describe the specific entity. For example, an employee entity is characterized by employee name, address etc. which are its attributes. Attributes can be classified into three categories, which are:

- Composite & simple attributes: composite attributes are the attributes which can be subdivided into smaller attributes. For example, the name of an employee can divided into first name, middle name, and last name. simple attributes are the attributes , which cannot be further subdivided into smaller attributes.
- Single valued & multivalued attributes: single valued attributes are the attributes, which have a single value for a particular entity. For example, employee name is a single valued attribute. Multivalued attributes are the attributes, which can have more than one value for a particular entity. For example, contact number of an employee is a multivalued attribute as a particular employee can have more than one contact number.
- Derived & stored attributes: derived attributes are the attributes whose value can be derived from the value of some other attributes. For example, the working years of an employee can be derived from the date of joining of the employee and present date. The attribute date of joining is said to be the stored attribute from which we derive the value of some other attribute.

4.4 RELATIONAL MODEL CONCEPTS

There are certain terms that are used in context of relational model. These are:

4.4.1 Domain: It is a set of atomic values. The values that cannot be divided into subcomponents are called atomic values. Generally, you specify a domain as a data type from which the values forming the domain are taken. For example, the employee_name attribute has domain name a to z i.e. it can accept between a to z only.

4.4.2 Tuple: In relational data model, a row of a table is termed as a tuple that gives complete information of an entity. For example, the row with employee name ‘Ravi’ in the Employee relation is a tuple.

4.4.3 Relationship: Relationship shows the association between two or more relations i.e. how entities are related to one other. For example in the Figure 4.1, Depositor is the relationship between the customer entity

and account entity. Customer entity has primary key Social_security and Account entity has primary key Account_number

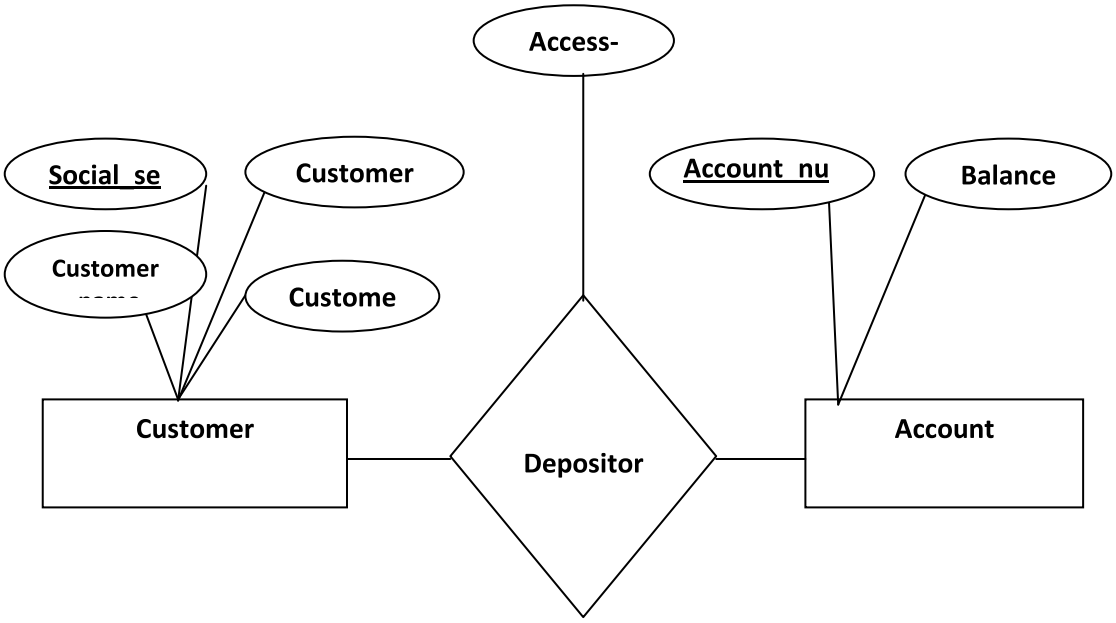


Fig. 4.1: Relationship set

4.4.4 Degree: Degree of a relationship is the number of relations participating in that relationship. For example Depositor relationship represents the relationship between two relations Customer and Account. Therefore the degree of Depositor relationship is two and is known as binary relationship.

4.5 TYPES OF RELATIONSHIPS

Mapping Cardinalities express the number of entities to which another entity can be associated via a relationship set. It is most useful in describing binary relationship sets. For a binary relationship set the mapping cardinality must be one of the following types:

- **One to one**
- **One to many**
- **Many to one**
- **Many to many**

4.5.1 One-To-One Relationship

A relationship between two entities is said to be 1:1 when a single instance of first entity is related only to another instance of the second entity. For example shown in Figure 4.2, Employee and Department are two entities. An Employee in an organization

belongs to a single department. Thus this relationship represents a 1:1 relationship:

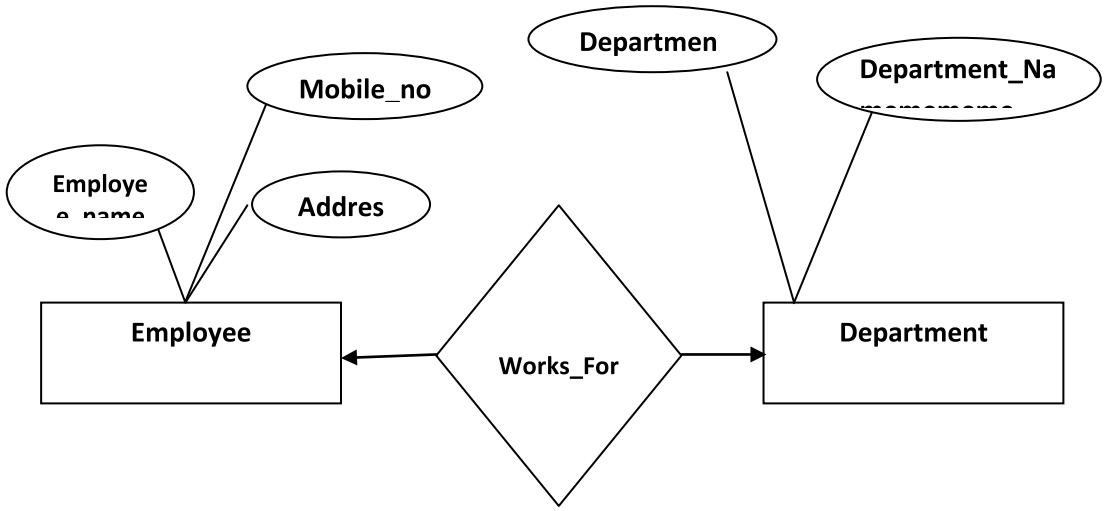


Fig. 4.2: One to One Relationship

4.5.2 One-To-Many Relationship

A relationship between two entities is said to be 1:N when a single instance of first entity is related to zero, one or more instances of the second entity but a single instance of second entity can be related to only a single instance of the first entity. For example, in the Figure 4.3, a loan is associated with at most one Customer via borrower; a customer is associated with several loans via borrower

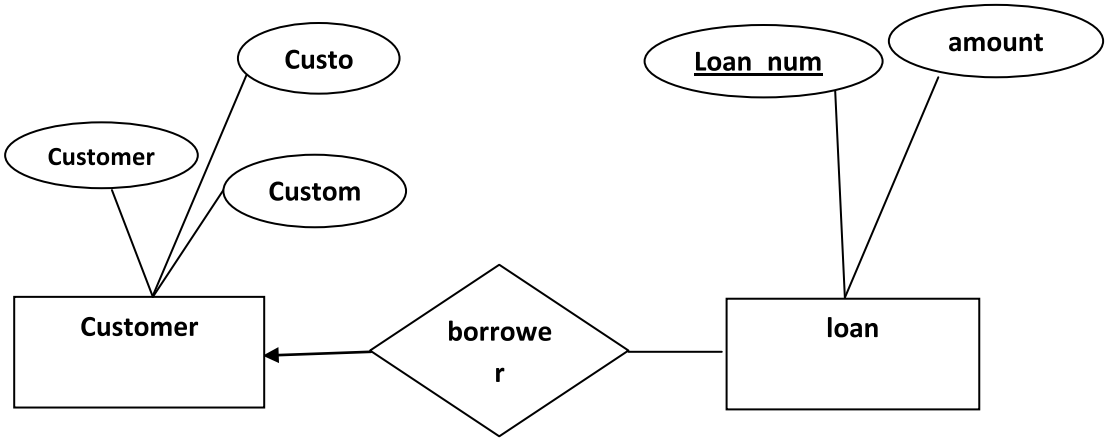


Fig. 4.3: One to Many Relationship

4.5.3 Many-To-Many Relationship

A relationship between two entities is said to be M:N when a single instance of first entity is related to zero, one or more instances of the

second entity and a single instance of second entity can also be related to zero, one or more instances of the first entity. For example, in the Figure 4.4, a Customer is associated with several loans via borrower and a loan entity is associated with several customers via borrower.

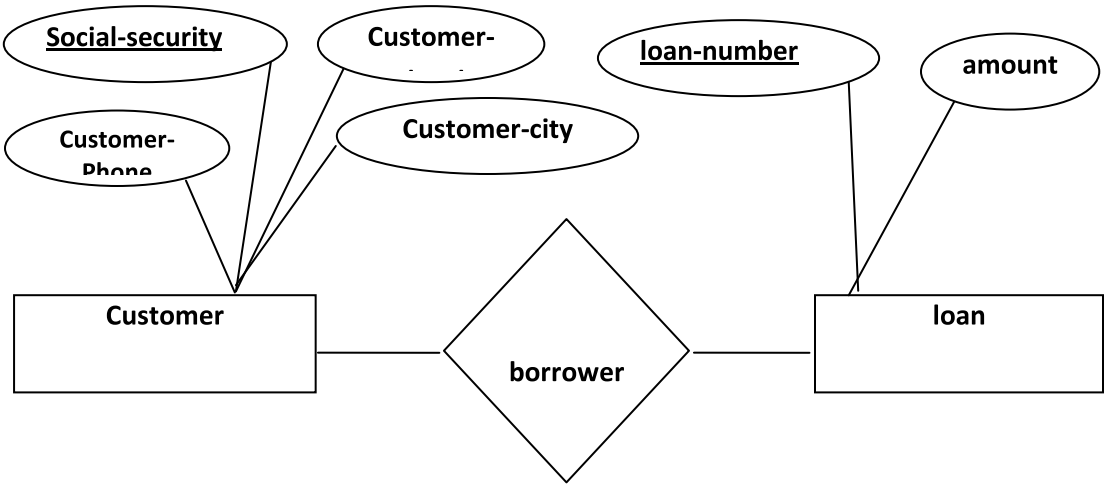


Fig. 4.4: Many to Many Relationships

4.6 Constraints in DBMS

Constraints define a condition, which need to be satisfied while storing data in a database. DBMS allows you to define and implement the constraints for a database object. For example, you can specify a constraint that each field in the employee_id column of the Employee table must contain a unique value. The database designers need to identify the constraints during database design.

Need of Constraints:

Constraints in the database provide a way to guarantee that:

- The values of individual columns are valid.
- In a table, rows have a valid primary key or unique key values.
- In a dependent table, rows have valid foreign key values that reference rows in a parent table.

4.7 Types of constraints in DBMS

4.7.1 Domain Constraints:

Domain Constraints specifies that what set of values an attribute can take. Value of each attribute X must be an atomic value from the domain of X. The data type associated with domains includes integer, character, string, date, time, currency etc. An attribute value must be available in the corresponding domain. Consider the example below –

RollNo	Name	Class	Age
11	Ankit	MCA	22
12	Srishti	BTech	18
13	Saurabh	MBA	A ⇦ Not allowed because Age is an integer attribute.

4.7.2 Tuple Uniqueness Constraints:

A relation is defined as a set of tuples. All tuples or all rows in a relation must be unique or distinct. Suppose if in a relation, tuple uniqueness constraint is applied, then all the rows of that table must be unique i.e. it does not contain the duplicate values. For example,

RollNo	Name	Class	Age
11	Ankit	MCA	22
12	Srishti	BTech	18
11	Ankit	MCA	22 ⇦ Not allowed because all rows must be unique.

4.7.3 Key Constraints

Keys are attributes or sets of attributes that uniquely identify an entity within its entity set. An Entity set E can have multiple keys out of which one key will be designated as the primary key. Primary key must have unique and not null values in the relational table. In an subclass hierarchy, only the root entity set has a key or primary key and that primary key must serve as the key for all entities in the hierarchy. For example,

RollNo	Name	Class	Age
11	Ankit	MCA	22
12	Srishti	BTech	18
11 ⇦ Not allowed as primary key value must be unique.	Saurabh	MBA	22

4.7.4 Single Value Constraints

Single value constraints refers that each attribute of an entity set has a single value. If the value of an attribute is missing in a tuple, then we cal fill it with a "null" value. The null value for a attribute will specify that either the value is not known or the value is not applicable. Consider the below example-

RollNo	Name	Class	Age	Driving Licence Number
11	Ankit	MCA	22	UP-2725
12	Srishti	BTech	18	UP-3871, UP- 6757 ⇧ Not allowed as a person does not have two driving licence
13	Saurabh	MBA	22	⇧ Allowed as a person may or may not have a driving licence.

4.7.5 Integrity Rule 1 (Entity Integrity Rule or Constraint)

The Integrity Rule 1 is also called Entity Integrity Rule or Constraint. This rule states that no attribute of primary key will contain a null value. If a relation has a null value in the primary key attribute, then uniqueness property of the primary key cannot be maintained. Consider the example below-

4.7.6 Integrity Rule 2 (Referential Integrity Rule or Constraint)

Referential integrity is a database concept that ensures that relationships between tables remain consistent. When one table has a foreign key to another table, the concept of referential integrity states that you may not add a record to the table that contains the foreign key unless there is a corresponding record in the linked table. It also includes the techniques known as cascading update and cascading delete, which ensure that changes made to the linked table are reflected in the primary table.

Foreign key constraint prevents conditions that violate any reference between the two database tables. A foreign key value refers to another table with the corresponding values of primary key in a database table. Consider the following example where you need to create a database table with a foreign key constraint.

CREATE TABLE Class

(student_id int, couese_name varchar(10), age int,

CONSTRAINT fk_student

FOREIGN KEY(student_id)

REFERENCES Student (STUD_ID))

The CREATE statement creates a Class database table with various columns such as student_id, course_name and age. The student_id defines a foreign key for the Class database and STUD_ID is the primary key for the Student database table. The student_id refers to the STUD_ID primary key of Student table.

4.7.7 General Constraints

General constraints are the arbitrary constraints that should hold in the database. Domain Constraints, Key Constraints, Tuple Uniqueness Constraints, Single Value Constraints, Integrity Rule 1 (Entity Integrity) and 2 (Referential Integrity Constraints) are considered to be a fundamental part of the relational data model. However, sometimes it is necessary to specify more general constraints like the CHECK Constraints or the Range Constraints etc.

Check constraints can ensure that only specific values are allowed in certain column. For example , if there is a need to allow only three values for the color like 'White', 'Red' and 'Green', then we can apply the check constraint. All other values like 'Black' etc would yield an error.

CarID	Name	Model	Color
A-2345	Maruti 800	2010	White
A-6754	Maruti 800	2012	Red
A-7865	Maruti 800	2011	Grey
A-5643	Maruti-800	2105	Black ⇧ Not allowed as CHECK constraint is applied

Range Constraints is implemented by BETWEEN and NOT BETWEEN. For example, if it is a requirement that student ages be within 16 to 25, then we can apply I the range constraints 0for it. The below example will explain Check Constraint and Range Constraint –

RollNo	Name	Class	Age
11	Ankit	MCA	22
12	Srishti	BTech	18
13	Saurabh	MBA	30 ⇧ Not allowed as the RANGE defined is in between 16 and 25

4.8 CONSTRAINTS IN SQL

The SQL CONSTRAINTS are an integrity which defines some conditions that restricts the column to remain true while inserting or updating or deleting data in the column. Constraints can be specified when the table created first with CREATE TABLE statement or at the time of modification of structure of an existing table with ALTER TABLE statement.

The SQL CONSTRAINTS are used to implement the rules of the table. If there is any violation of the constraints caused some action not performing properly on the table the action is aborted by the constraint.

Some CONSTRAINTS can be used along with the SQL CREATE TABLE statement.

The general structure of the SQL CONSTRAINT is defined as:

The CONSTRAINT keyword is followed by a constraint name followed by a column or a list of columns.

Syntax:

CREATE TABLE <table name>(

column1 data_type[(size)] constraint,

column2 data_type[(size)] constraint,

…….);

"CHECK YOUR PROGRESS 1"

1. What is a relation? What are its characteristics?
2. Define Constraints in database.
3. Define the terms: domain, tuple and degree.
4. Define derived and stored attributes.
5. Discuss the various features of relational data model in detail.

4.9 TYPES OF SQL CONSTRAINTS

The SQL provides following types of Constraints:

Constraint	Description
NOT NULL	This constraint confirms that a column cannot store NULL value.
UNIQUE	This constraint ensures that each rows for a column must have different value.
PRIMARY KEY	This constraint is a combination of a NOT NULL

	constraint and a UNIQUE constraint. This constraint ensures that the specific column or combination of two or more columns for a table have an unique identity which helps to find a particular record in a table more easily and quickly.
CHECK	A check constraint ensures that the value stored in a column meets a specific condition.
DEFAULT	This constraint provides a default value when specified none for this column.
FOREIGN KEY	A foreign key constraint is used to ensure the referential integrity of the data in one table to match values in another table.

4.9.1 Not Null

NOT NULL constraint makes sure that a column does not hold NULL value. When we don't provide value for a particular column while inserting a record into a table, it takes NULL value by default. By specifying NULL constraint, we can be sure that a particular column(s) cannot have NULL values.

Example:

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

4.9.2 Unique

UNIQUE Constraint enforces a column or set of columns to have unique values. If a column has a unique constraint, it means that particular column cannot have duplicate values in a table.

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL UNIQUE,
STU_AGE INT NOT NULL,
STU_ADDRESS VARCHAR (235) UNIQUE,
PRIMARY KEY (ROLL_NO)
);
```

4.9.3 Default

The DEFAULT constraint provides a default value to a column when there is no value provided while inserting a record into a table.

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

4.9.4 Check

This constraint is used for specifying range of values for a particular column of a table. When this constraint is being set on a column, it ensures that the specified column must have the value falling in the specified range.

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL CHECK(ROLL_NO >1000),
STU_NAME VARCHAR (35) NOT NULL,
STU_AGE INT NOT NULL,
EXAM_FEE INT DEFAULT 10000,
STU_ADDRESS VARCHAR (235),
PRIMARY KEY (ROLL_NO)
);
```

In the above example we have set the check constraint on ROLL_NO column of STUDENT table. Now, the ROLL_NO field must have the value greater than 1000.

4.9.5 Primary Key

Primary key uniquely identifies each record in a table. It must have unique values and cannot contain nulls. In the below example the ROLL_NO field is marked as primary key, that means the ROLL_NO field cannot have duplicate and null values.

```
CREATE TABLE STUDENT (
ROLL_NO INT NOT NULL,
```

STU_NAME VARCHAR (35) NOT NULL UNIQUE,

STU_AGE INT NOT NULL,

EXAM_FEE INT DEFAULT 10000,

STU_ADDRESS VARCHAR (235) UNIQUE,

PRIMARY KEY (ROLL_NO)

);

4.9.6 Foreign Key

A foreign key constraint specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table. We say this maintains the referential integrity between two related tables.

Say you have the product table that we have used several times already:

CREATE TABLE products (

product_no integer PRIMARY KEY,

name text, price numeric);

Let's also assume you have a table storing orders of those products. We want to ensure that the orders table only contains orders of products that actually exist. So we define a foreign key constraint in the orders table that references the products table:

CREATE TABLE orders (

order_id integer PRIMARY KEY,

product_no integer REFERENCES products (product_no),

quantity integer);

Now it is impossible to create orders with product_no entries that do not appear in the products table.

We say that in this situation the orders table is the referencing table and the products table is the referenced table. Similarly, there are referencing and referenced columns.

Example:

Consider the structure of the two tables as follows:

CUSTOMERS table:

```
CREATE TABLE CUSTOMERS (
ID INT     NOT NULL,
NAME VARCHAR (30)     NOT NULL,
AGE INT   NOT NULL,
ADDRESS CHAR (25),
SALARY DECIMAL(18,2)
PRIMARY KEY (ID)
);
```

ORDERS table:

```
CREATE TABLE CUSTOMERS (
ID INT     NOT NULL,
DATE    DATETIME,
CUSTOMER_ID INT references CUSTOMERS (ID),
AMOUNT DOUBLE,
PRIMARY KEY (ID)
);
```

If ORDERS table has already been created, and the foreign key has not yet been set, use the syntax for specifying a foreign key by altering a table.

```
ALTER TABLE ORDERS
ADD FIREIGN KEY (Customer_ID) REFERENCES CUSTOMERS (ID)
```

4.10 WEAK ENTITY SETS:

The entity set which does not have sufficient attributes to form a primary key is called as Weak entity set. An entity set that has a primary key is called as Strong entity set. Consider an entity set Payment which has three attributes: payment_number, payment_date and payment_amount. Although each payment entity is distinct but payment for different loans may share the same payment number. Thus, this entity set does not have a primary key and it is a weak entity set. Each weak set must be a part of one-to-many relationship set.

A member of a strong entity set is called dominant entity and member of weak entity set is called as subordinate entity. A weak entity set does not have a primary key but we need a means of distinguishing among all those entries in the entity set that depend on one particular strong entity set. The discriminator of a weak entity set is a set of attributes that allows this distinction be made. For example, payment_number acts as discriminator for payment entity set. It is also called as the Partial key of the entity set.

The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent plus the weak entity sets discriminator. In the above example {loan_number, payment_number} acts as primary key for payment entity set.

The relationship between weak entity and strong entity set is called as Identifying Relationship. In example, loan-payment is the identifying relationship for payment entity. A weak entity set is represented by doubly outlined box and corresponding identifying relation by a doubly outlined diamond as shown in Figure 4.5. Here double lines indicate total participation of weak entity in strong entity set. It means that every payment must be related via loan-payment to some account. The arrow from loan-payment to loan indicates that each payment is for a single loan. The discriminator of a weak entity set is underlined with dashed lines rather than solid line

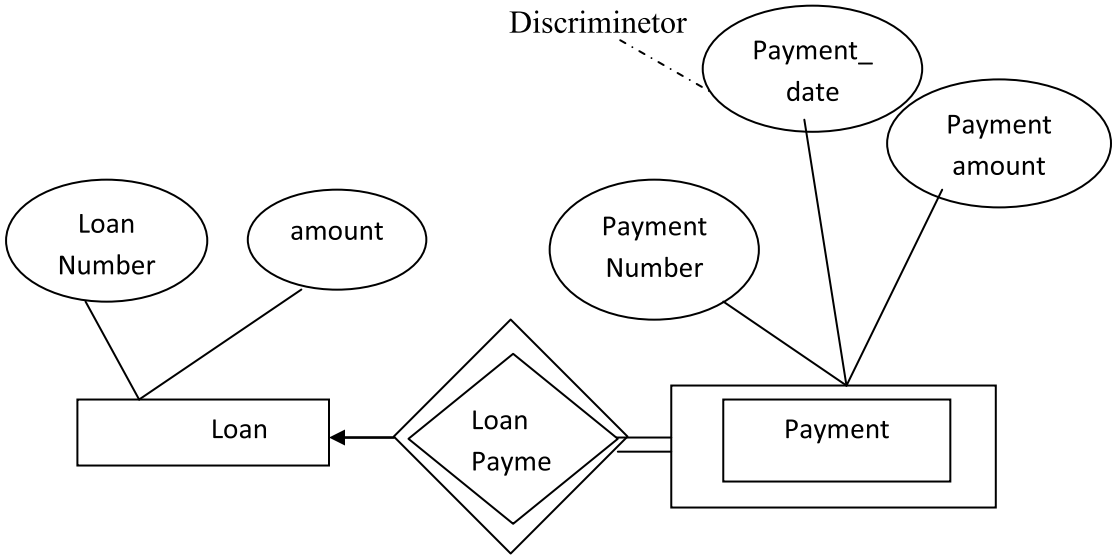


Fig. 4.5: Relation between strong and weak entity set

The tabular comparison between Strong Entity Set and Weak Entity Set is as follows:

S.No	Strong Entity Set	Weak Entity Set
1	It has its own primary key.	It does not have sufficient attributes to form a primary key on its own.
2	It is represented by a rectangle.	It is represented by a double rectangle.
3	It contains a primary key represented by an underline.	It contains a partial key or discriminetor represented by a dashed underline.
4	The member of strong entity set is called as dominant entity set.	The member of weak entity set is called as subordinate entity set.
5	The relationship between two strong entity set is represented by a diamond symbol.	The relationship between one strong and a weak entity set is represented by a double diamond sign known as identifying relationship.
6	The line connecting strong entity set with the relationship is single.	The line connecting weak entity set with the identifying relationship is double.
7	Total participation in the relationship may or may not exist.	Total participation in the identifying relationship always exist.

4.11 ASSERTIONS

An assertion is a predicate expressing a condition that we wish the database to always satisfy. For example, consider the statements:

(a) The sum of all loan amounts for each branch must be less than the sum of all balances at the branch.

(b) Each loan has at least one customer who maintains an account with a minimum balance of Rs. 1000.000

The SQL-92 standard provides the following form for the assertion declaration:

create assertion <assertion-name> check <predicate>

The two previous constraints would be:

```
create assertion sum-constraint check
   (not exists (select * from branch
      where ( select sum(amount) from loan
            where loan.branch-name = branch.branch-name)
         >= (select sum(amount) from account
            where loan.branch-name = branch.branch-name)))
```

create assertion balance check (not exists (select * from loan where not exists (select *

```
                from   borrow, depositor, account
                where  loan.loan-number = borrower.loan
                and borrower.customer-name =
                      depositor.customer-name
                and depositor.account-number =
                      account.account-number
                and account.balance = 1000)))
```

4.12 RELATIONAL CALCULUS:

The relational calculus is a non procedural query language whereas relational algebra is a procedural query language.

In non-procedural query language, the user is concerned with the details of how to obtain the end results. Whereas in procedural query language we define each step in order to obtain the end result. In relational calculus a query is expressed as a formula consisting variables. There is no mechanism to specify how a formula should be evaluated.

Relational calculus are of two types

(1) Tuple Relational Calculus

(2) Domain Relational Calculus

1.12.1 Tuple Relational Calculus

The tuple relational calculus is based on specifying a number of tuple variables. The simple query based of tuple relational calculus is given by the following equation:

{ t| COND (t)}

where t is a tuple variable and COND (t) is a conditional expression.

The result of a query in the tuple relational calculus is all the tuples which satisfy condition that may specify in the query. Consider the relation Student listed in Table 4.1

Name	SSN	Phone No	City	Age	GPA
Amit	546-65	9876754634	Agra	19	5.8
Vijay	456-78	Null	Delhi	18	4.9
Saroj	435-67	0656567656	Mumbai	22	3.29
Himanshu	768-34	9656467646	Bareilly	23	7.8
Santosh	564-78	9876574367	Varanasi	18	5.7

Table 4.1: The Student relation

In this relation if you need to find all the students whose ages are less than 20, you can write the following query

{t| Student(s) and s.age<20}

As a result three tuples will be retrieved from the Student relation as listed in table 4.2

Name	SSN	Phone No	City	Age	GPA
Amit	546-65	9876754634	Agra	19	5.8
Vijay	456-78	Null	Delhi	18	4.9
Santosh	564-78	9876574367	Varanasi	18	5.7

4.12.2 The Domain Relational Calculus

The domain relational calculus is based on specifying single values from domain attributes. The simple query based of domain relational calculus is given by the following equation:

$\{x_1,x_2,.....x_n \mid COND(x_1,x_2,.....x_n,x_{n+1}, x_{n+2}, x_{n+m})\}$

Here $x_1,x_2,...x_n, x_{n+1}, x_{n+2},..... x_{n+m}$ are domain variables that range over attributes, whereas COND is a condition or formula of the domain relational calculus.

The atoms of formula in domain relational calculus are

(i) An atom of the form relation $R(x_1,x_2,.....x_n)$ where degree of the relation is n and name of the relation is a domain variable. Each $x_{i\ is}$ a domain variable in which i ranges between 1<=i<=n. This atom specifies a list of values of $(x_1,x_2,.....x_{n)}$ which are tuple in the relation R.

(ii) An atom of the form x_i op x_j where op is one of the comparison operator and x_i and x_j are domain variables.

(iii) An atom of the form x_i op c or c op x_j where op is one of the comparison operator and x_i and x_j are domain variables, and c is a constant value.

Some examples of queries specified in the domain calculus follow. We will use lowercase letters l,m,n,…….,x,y,z for domain variables.

(1) List the names of managers who has at least one dependency.

{s,q | (∃ t) (∃ j) (∃ l) (EMPLOYEES (qestuvwxyz) AND DEPARTMENT (hijk) AND DEPENDENT (lmnop) AND t=j AND l=t))

(2) Retrieve the name and address of all employees who work for the 'Research' department.

{q, s, v| (∃ z) (∃ l) (∃ m) (EMPLOYEE(qrstuvwxyz) AND DEPARTMENT (lmno) AND l='Research' AND m=z)}

4.12.3 Domain versus Tuple Calculus

In tuple calculus, the range variable ranges over the tuple instead of relation. In domain calculus the range variables range over the domain instead of tuple. Range variable is a variable that helps in relational calculus to retrieve data in the form of tuples and domain from database.

"CHECK YOUR PROGRESS 2"

1. Explain relational algebra.
2. Differentiate between tuple and domain calculus,
3. Differentiate between procedural and non-procedural language.
4. Define Strong and Weak entity set.
5. Explain Entity Integrity constraint with example.

4.13 RELATIONAL CALCULUS VERSUS RELATIONAL ALGEBRA

Relational algebra is a procedural query languages that allow you to retrieve data from a database by specifying sequence of operations. Whereas relational calculus is a nonprocedural query language in which

you can write declarative expressions to specify the request for retrieving data from a database.

Relational algebra contains a set of explicit operators such as join, union and intersection. These explicit operators tell the database how to construct a desired relation from the given relations. Relational calculus provides a notation for retrieving a desired relation instead of using explicit operators. Relational calculus is divided into two categories tuple and domain.

The distinction between relational algebra and relational calculus is superficial because relational calculus and relational algebra are logically equivalent. Each query of relational algebra has an equivalent query in relational calculus. Similarly each query of relational calculus has equivalent query in relational algebra.

4.14 SUMMERY

- Entity- An entity is an object that exists and is distinguishable from other objects. An entity set is a set of entities of the same type that share the same properties.
- Attributes- An entity is represented by a set of attributes, that is, descriptive properties possessed by all members of an entity set.
- Relationship sets - A relationship is an association among several entities.
- Degree of a Relationship Set-Refers to number of entity sets that participate in a relationship set.
- Mapping Cardinalities Express the number of entities to which another entity can be associated via a relationship set.
- An integrity constraint is the condition that can be applied on a database schema to restrict the data according to the need. It is of two types: Entity integrity constraints and referential integrity constraints.
- Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.
- An assertion is a predicate expressing a condition that we wish the database to always satisfy.
- Query language is a language through which we communicate with the database. It is of two types: Relational algebra & relational Calculus.

4.15 OBJECTIVE TYPE QUESTIONS:

Q 1. Every time attribute A appears, it is matched with the same value of attribute B, but not the same value of attribute C. Therefore, it is true that:

(A) A ? B. (B) A ? C.

(C) A ? (B,C). (D) (B,C) ? A.

ANS: A

Q 2. The different classes of relations created by the technique for preventing modification anomalies are called:

(A) normal forms. (B) referential integrity constraints.

(C) functional dependencies. (D) None of the above is correct.

ANS: A

Q 3. A relation is in this form if it is in BCNF and has no multivalued dependencies:

(A) second normal form. (B) third normal form.

(C) fourth normal form. (D) domain/key normal form.

ANS: C

Q 4. Row is synonymous with the term:

(A) record. (B) relation.

(C) column. (D) field.

ANS: A

Q 5. The primary key is selected from the:

(A) composite keys. (B) determinants.

(C) candidate keys. (D) foreign keys.

ANS: C

Q 6. Which of the following is a group of one or more attributes that uniquely identifies a row?

(A) Key (B) Determinant

(C) Tuple (D) Relation

ANS: A

Q 7. When the values in one or more attributes being used as a foreign key must exist in another set of one or more attributes in another table, we have created a(n):

(A) transitive dependency. (B) insertion anomaly.

(C) referential integrity constraint. (D) normal form.

ANS:C

Q 8. A relation is considered a:

(A) Column. (B) one-dimensional table.

(C) two-dimensional table. (D) three-dimensional table.

ANS C

Q 9. In the relational model, relationships between relations or tables are created by using:

(A) composite keys. (B) determinants.

(C) candidate keys. (D) foreign keys.

ANS: D

Q 10. A functional dependency is a relationship between or among:

(A) tables. (B) rows.

(C) relations. (D) attributes.

ANS: D

4.16 SELECTED EXERCISE

Q1: Given relational schema:

Frequent (D, P)

Serves (P, B)

Likes (D, B)

Attributes: P (pub), B (beer), D (drinker)

Write the SQL statements for the following query

(a) The pubs that serve a beer that Jefferson likes.

(b) Drinkers that frequent at least one pub that serves “Bud” or “Becks”.

(c) Drinkers that frequent only pubs that serve some beer they like

(d) Drinkers that frequent only pubs that serve no beer they like.

Q2: What is Relational Algebra? Explain with Example.

Q3: What is Entity Set Relationship & describe roles & design issues.

Q4: Describe the Relational Calculus with Example.

Q5: Define Assertion & types of assertion.

Q6: What is Referential Integrity?

Q7: What do you mean by Weak Entity Sets?

Q8. Define relational algebra, Tuple & Domain relational calculus.

Q9. Consider the following database.

Employee (employee-name, street, city)

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

Company (company-name, city)

Manager (employee-name, manager-name)

Give an expression in the relational algebra, the tuple relational calculus, and the domain relational calculus, for the following query.

Find the names of all employees who work for estate bank.

Q10. Discuss the various features of relational data model in detail.

UNIT-5

STRUCTURE QUERY LANGUAGE

Structure

5.1 INTRODUCTION

The objective of a database is to store data in such a manner that it can be easily accessed and altered by the user who is using a database. To store and retrieve data from a database, DBMS uses a database language. The database languages allow a database administrator to retrieve, update and remove data from the database. The language which a database administrator can use to maintain database are SQL (structured query language). SQL is a standard, interactive and programming language for accessing, querying, managing and updating data in RDBMS. Using SQL you can also create databases and various database objects such as table, columns and views. In addition SQL is useful not only for changing the configuration of the server of a database but also for modifying the

database or its session settings. Now, with technology advancements, SQL also supports object-relational database management systems.

Programming language SQL (PL/SQL) is a development tool that extends the features provided in the SQL database language. PL/SQL allows you to provide flow control and logic design to unstructured SQL command blocks.

5.2 OBJECTIVES

At the end of this unit, you should be able to:

- Introducing the SQL objects needed for database management activities.
- Describing how SQL Server Enterprise Manager is used to create a database table.
- Describing the CREATE TABLE statement for creating a database table and use of SELECT statement to retrieve data from databases.
- Explaining how you can manipulate data stored in a database using SQL statements.
- Describing character sets, literals and data types in SQL.

5.3 STRUCTURED QUERY LANGUAGE

SQL (structured Query Language) is a computer language for storing, manipulating and retrieving data stored in relational database. SQL is the standard language for Relation Database System. All relational database management systems like MySQL, MS Access, and Oracle, Sybase, Informix, postgres and SQL Server uses SQL as standard database language.

5.4 NEED OF SQL

SQL is a standard language for accessing and manipulating databases. SQL help us to retrieve data from database according to user requirement. It is a standard language for relational database system. We can describe SQL as follows:

- Allow users to access data in relational database management systems.
- Allow users to describe the data.
- Allow users to define the data in database and manipulate that data.
- Allow to embed within other languages using SQL modules, libraries & pre- compilers.
- Allow users to create and drop/delete databases and tables.

- Allow users to create view, stored procedure, functions in a database.
- Allow users to set permissions on tables, procedures, and views.

5.5 CHARACTERISTICS OF SQL

SQL is a standard interactive and programming language for querying and modifying data and managing databases. Although SQL is both an ANSI and an ISO standard, many database products support SQL with proprietary extensions to the standard language. The core of SQL is formed by a command language that allows you to retrieve, insert, update, and delete data, and perform management and administrative functions. SQL also includes a call-level interface (SQL/CLI) for accessing and managing data and database remotely. Some important characteristics of SQL are

(i) SQL enables end user and system persons to deal with a number of database management systems where it is available.

(ii) Applications written in SQL can be easily ported across systems. Such porting could be required when the underlying DBMS needs to upgraded because of change in transaction volumes or when a system developed in one environment is to be used on another.

(iii) SQL as a language is independent of the way it is implanted internally. A query returns the same result regardless of whether optimizing has been done with indexes or not. This is because SQL specifies what is required and not how it is to be done.

(iv) The language while being simple and easy to learn can cope with complex situations.

(v) The results to be expected are well defined in SQL.

5.5.1 SQL statements

SQL statements can be executed on any computer (mainframe or personal computer) and under any operating system. The SQL statement (or command) accepts one or more relations as input and returns a single relation as output. SQL follows the relational calculus style. The syntax of SQL statements are also similar to relational calculus operators.

It also has features for:

(1) Defining views on database.

(2) Specifying, security and authorization.

(3) Defining integrity constraints.

(4) Specifying transaction controls.

5.6 Advantages of SQL

Below given are the advantages of Structured Query Language (SQL):

(1) **Portable:** SQL is run in programs in mainframes, PCs, laptops, servers and even mobile phones. It runs in local systems, intranet and internet. Databases using SQL can be moved from one device to another without any problems.

(2) **Used with any DBMS system with any vendor:** SQL is used by all the vendors who develop DBMS.

(3) **SQL Standard:** First standard for SQL was put up in 1986 by ANSI (American National Standards Institute) and ISO (International Standards Organization). It was later expanded in 1989 and in 1992 and 1999.

(4) **Used for relational databases:** SQL is widely used for relational databases.

(5) **Easy to learn and understand:** SQL mainly consists of English statements and it is very easy to learn and understand a SQL query.

(6) **Interactive language:** SQL can be used to communicate with the databases and get answers to complex questions in seconds.

(7) **Both as programming language and interactive language:** SQL can do both the jobs of being a programming as well as an interactive language at the same time.

(8) **Complete language for a database:** SQL is used to create databases, manage security of a database. It can also be used for updating, retrieving and sharing data with users.

(9) **Multiple data views:** By use of SQL, different views of structure and content of a database can be provided for different users.

(10) **Dynamic database language:** By the use of SQL database structure can be changed in a dynamic fashion even when the contents of the database are accessed by users at the same time.

(11) **Supports object based programming:** SQL supports the latest object based programming and is highly flexible.

(12) **Supports enterprise applications:** SQL is the database language which is used by businesses and enterprises throughout the globe. For an enterprise application it is a perfect language for a database.

(13) Integrates with Java: SQL integrates with Java by using an API known as JDBC (Java Database Connectivity).

(14) Used in internet: SQL is used in three tiered Internet architecture. The architecture includes a client, application server and a database. The architecture includes a client, application server and a database. In the Oracle client/server architecture, the database application and the database are separated into two parts: a front-end or client portion, and a back-end or server portion. The client executes the database application that accesses database information and interacts with a user through the keyboard, screen, and pointing device such as a mouse. The server executes the Oracle software and handles the functions required for concurrent, shared data access to an Oracle database.

The client and server are located on different computers; these computers are connected via a network as shown in Figure 5.1.

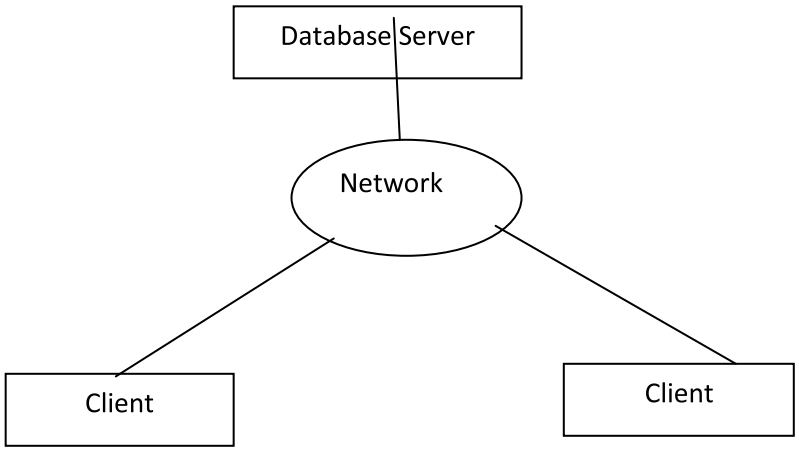


Fig. 5.1: Client/server architecture

5.7 SQL Process

When you are executing an SQL command from any RDBMS, the system determines the best way to carry out your request and SQL engine figures out how to interpret the task. There are various components included in the process. These components are Query Dispatcher, Optimization engines, Classic Query Engine and SQL query engine etc. Classic query engine handles all non-SQL queries but SQL query engine won't handle

logical files. Following is a simple diagram (Figure 5.2) showing SQL Architecture

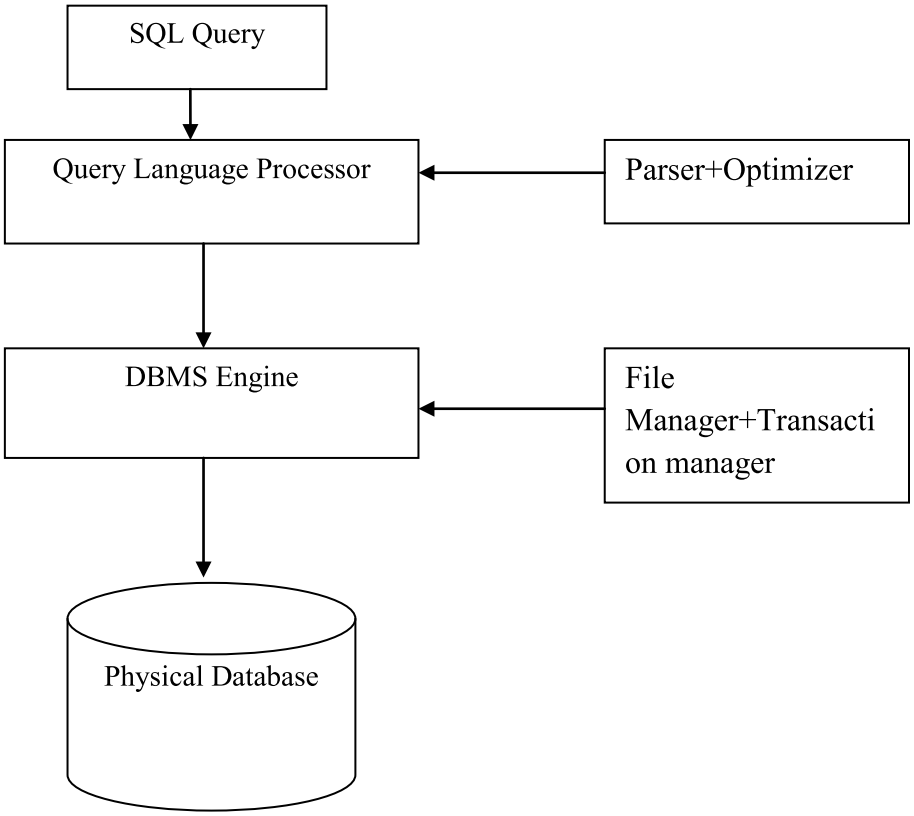


Fig. 5.2: SQL Process

5.8 SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE, and DROP. These commands can be classified into groups based on their nature:

5.8.1 DDL - Data Definition Language

Command	Description
CREATE	Creates a new table, a view of a table, or other object in database
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other object in the database.

(a) Creating a Table

A row refers to the horizontal part of the table which contains one or more columns. A column refers to the vertical part of the table which contains one or more rows of data of one type.

The CREATE statement is used to create and manage database objects. For example you can create a Customers database table that consists of information details of customers of an organization. To create a Customer table, execute the following SQL statement**.**

CREATE TABLE Customer;

The above command creates the Customer database on DBMS.

Alternatively the following statement can also be used to create a customer details. table that contains detail of the customers of the organization such as cust_firstname, cust_lastname and cust_id.

CREATE TABLE cust_details(cust_firstname char(20) not null, cust_lastname char(20) not null, cust_id int not null);

The above statement creates a table cust_details in the current database. The various attributes of the cust_details are cust_firstname, cust_lastname and cust_id and column of the cust_details table.

(b) How to modify table

Once a table is created it's structure is not necessarily fixed in stone. In time requirements change and the structure of the database is likely to evolve to match your wishes. SQL can be used to change the structure of a table, so, for example, if we need to add a new field to our Customer table to tell us if the customer has phone no, then we can execute an SQL ALTER TABLE command as shown below:

ALTER TABLE Customer ADD COLUMN Phone_no int;

To delete a column the ADD keyword is replaced with DROP, so to delete the field we have just added the SQL is:

ALTER TABLE Customer DROP COLUMN Phone_no;

(c) How to delete table

If you have already executed the original CREATE TABLE command your database will already contain a table called Customer, so let's get rid of that using the DROP command:

DROP TABLE Customer;

5.8.2 DML - Data Manipulation Language

Command	Description
INSERT INTO	inserts new data into a database table
UPDATE	updates data in a database table
DELETE	deletes data from a database table

(a) How to Insert Data

The command to add new records to a table is:

INSERT INTO target [(field1 [, field2 [, ...]])]

VALUES (value1 [, value2 [, ...]);

So, to add a customer record for user amit, we would issue the following INSERT query:

INSERT INTO Customer (cust_firstname, cust_lastname, cust_id)

VALUES ("sanjeev", "gangwar", 9);

(b) How to Update Data

The INSERT command is used to add records to a table, but what if you need to make an amendment to a particular record? In this case the SQL command to perform updates is the UPDATE command, with syntax:

UPDATE table

SET newvalue

WHERE criteria;

For example, let's assume that we want to change customer last name gangwar to patel. Our SQL statement would then be:

UPDATE Customer

SET cust_lastname="patel"

WHERE cust_id=9;

(c) How to Delete Data

SQL provides a simple command to delete complete records. The syntax of the command is:

DELETE [table.*]

FROM table

WHERE criteria;

Let's assume we have a user record for Sanjeev gangwar, (with cust_id 9), which we want to remove from our User we could issue the following query:

DELETE *

FROM Customer

WHERE cust_id=9;

This query will delete an entire record of customer sanjeev gangwar.

5.8.3 DCL - Data Control Language:

Command	Description
GRANT	Gives a privilege to user
REVOKE	Takes back privileges granted from user

(a) The GRANT statement is used to grant the privileges on the database objects to specific users. Normally the GRANT statement is used by owner of the table or view to give other users access to the data. The GRANT statement includes list of the privileges to be granted, name of the table to which privileges apply and user id to which privileges are granted.

Example

(1) Give user ABC full access to employee table:

GRANT Select, Insert, Delete, update on employee to ABC

(2) Let user ABC only read the employee table. Update, delete and insert are not allowed.

GRANT Select on employee to ABC

(3) Give all users select access to employee table:

GRANT Select on employee to public

Note that GRANT statement in the above example grants access to all present and future authorized users. This eliminates the need for you to explicitly grant privileges to new users as they are authorized.

(b) **REVOKE STATEMENT:** In most SQL based databases, the privileges that you have granted with the GRANT statement can be taken away with the REVOKE statement. The structure of the REVOKE statement is much similar to that of the GRANT statement. A REVOKE statement may take away all or some of the privileges granted to a user id. E.g.

Revoke Select, Insert on employee from ABC

5.8.4 DQL - Data Query Language:

Command	Description
SELECT	Retrieves certain records from one or more tables

The SELECT statement is used in conjunction with the FROM keyword which specifies the name of a table from where the stored data is to be retrieved. The following syntax determines the basic SQL SELECT statement:`

SELECT Column_name

From Table_name

let's assume that we want to retrieve customer first name from Customer table. Our SQL statement would then be

SELECT cust_firstname

FROM Customer;

This SELECT statement retrieves records from a single column in a database table.

5.9 SQL Queries

SQL is followed by unique set of rules and guidelines called syntax. This unit gives you a quick start with SQL by listing all the basic SQL Syntax. All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and the entire statements end with a semicolon (;).

Important point to be noted is that SQL is **case insensitive** which means SELECT and select have same meaning in SQL statements.

(1) SQL SELECT Statement:

SELECT column1, column2…column FROM table_name;

(2) SQL DISTINCT Clause:

SELECT DISTINCT column1, column2…column FROM table_name;

(3) SQL WHERE Clause:

SELECT column1, column2....columnN FROM table_name WHERE CONDITION;

(4) SQL AND/OR Clause:

ELECT column1, column2....columnN FROM table_name WHERE CONDITION-1

{AND|OR} CONDITION-2;

(5) SQL IN Clause:

SELECT column1, column2....columnN FROM table_name

WHERE column_name IN (val-1, val-2,...val-N);

(6) SQL BETWEEN Clause:

SELECT column1, column2....columnN FROM table_name

WHERE column_name BETWEEN val-1 AND val-2;

(7) SQL LIKE Clause:

SELECT column1, column2....columnN FROM table_name

WHERE column_name LIKE {PATTERN};

(8) SQL ORDER BY Clause:

SELECT column1, column2....columnN FROM table_name

HERE CONDITION ORDER BY column_name {ASC|DESC};

(9) SQL GROUP BY Clause:

SELECT SUM (column_name) FROM table_name WHERE CONDITION

GROUP BY column_name;

(10) SQL COUNT Clause:

SELECT COUNT (column_name) FROM table_name WHERE CONDITION;

(11) QL HAVING Clause:

SELECT SUM (column_name) FROM table_name WHERE CONDITION

GROUP BY column_name HAVING (arithematic function condition);

(12) SQL DROP TABLE Statement:

(13) QL CREATE TABLE Statement:

CREATE TABLE table_name (Column1 datatype, Column2 datatype, Column3

datatype, ... ColumnN datatype, PRIMARY KEY (one or more columns));

DROP TABLE table_name;

(14) SQL CREATE INDEX Statement:

CREATE UNIQUE INDEX index_name ON table_name (column1,

column2,...columnN);

(15) SQL DROP INDEX Statement:

ALTER TABLE table_name DROP INDEX index_name;

(16) SQL DESC Statement:

DESC table_name;

(17) SQL TRUNCATE TABLE Statement:

TRUNCATE TABLE table_name;

(18) SQL ALTER TABLE Statement:

ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_ype};

(19) SQL ALTER TABLE Statement (Rename) :

ALTER TABLE table_name RENAME TO new_table_name;

(20) SQL INSERT INTO Statement:

INSERT INTO table_name (column1, column2....columnN)

VALUES (value1, value2....valueN);

(21) SQL UPDATE Statement:

UPDATE table_name SET column1 = value1, column2 = value2....columnN=valueN

[WHERE CONDITION];

(22) SQL DELETE Statement:

DELETE FROM table_name WHERE {CONDITION};

(23) SQL CREATE DATABASE Statement:

CREATE DATABASE database_name;

(24) SQL DROP DATABASE Statement:

DROP DATABASE database_name;

(25) SQL USE Statement:

USE DATABASE database_name;

(26) SQL COMMIT Statement:

COMMIT;

(27) SQL ROLLBACK Statement:

ROLLBACK;

5.10 SQL DATA TYPES:

SQL data type is an attribute that specifies type of data of any object. Each column, variable and expression has related data type in SQL.

You would use these data types while creating your tables. You would choose a particular data type for a table column based on your

requirement. Following are the various categories of data types that are supported by SQL:

- Numeric
- Character
- Date and time
- Binary
- Boolean

5.10.1 Numeric Data Types

Numeric data types are used to store numeric data in a database table. Table 5.1 lists the numeric data types and their description.

Data types	Description
INTEGER or INT(size) SMALLINT(size) BIGINT(size)	Defines integer data types only. The size in the parenthesis determines the numbers of digits.
DECIMAL or DEC(size,d) NUMERIC(size,d)	Defines the decimal number. The size in parenthesis determines the maximum numbers of digits and d determines maximum number of digits to the right of the decimal.

Table 5.1: The Numeric Data Types in SQL

5.10.2 Character Data Types

Character data types are the data types that represent sentences, symbols or a combination of both of them. Table 5.2 lists the character data types.

Data types	Description
CHARACTER or CHAR(SIZE)	Defines a fixed length string that can be letters, numbers, and special characters. The fixed size is specified in parenthesis.
CHARACTER VARYING or VARCHAR(SIZE)	Defines a variable length string. The maximum size is specified in parenthesis.

Table 5.2: The Character Data Types in SQL

5.10.3 Date and Time Data Types

The date data type enables you to define the date for a column value. Table 5.3 lists the date data type in different formats:

Data Type	Description
Date(yyyymmdd)	2015-07-25
Date(mmm yyyy)	Jan 2016
Date(mm-yy)	07-2014

Table 5.3: The Date Data Types in SQL

The above table describes the date data types for specifying the dates in the columns of a database table. The date and time data types are datetime and smalldatetime.

5.10.4 Binary Data Types:

Binary data type is similar to hexadecimal data and comprising characters that range from 0-9 and A-F, in groups of two characters each. You need to specify 0x before the binary value. Table 5.4 lists the two binary data types and their description:

DATA TYPE	Description
BINARY[(n)]	Stores up to 8000 bytes of fixed length binary data. The maximum byte length may be specified in parenthesis.
VARBINARY[(n)]	Stores up to 8000 bytes of variable length binary data. n specifies the maximum byte length of a binary variable. By variable-length, it is meant that binary data may contain less than n bytes and the actual length of the data entered gives the storage size.

You should use the varbinary data type instead of the binary data type, when you expect null values or a variation in data size.

5.10.5 Boolean Data Types

The BOOLEAN type is used for representing values that can be either true or false. Unless prohibited by a NOT NULL constraint, a BOOLEAN can be NULL or unknown.

5.11 SQL Operators

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparisons and arithmetic operations. Operators are used to specify conditions in an SQL statement and to serve as conjunctions for multiple conditions in a statement.

A SELECT statement uses operators to filter the data records that you want to retrieve from the database tables. The various operators used in the SELECT statement are:

- Arithmetic operators
- Comparison operators
- Logical operators
- Operators used to negate conditions

5.11.1 SQL Arithmetic Operators:

The arithmetic operators allow you to perform mathematical functions such as addition and multiplication on data stored in the database tables. Let us assume that variable ‘a’ holds 10 and variable ‘b’ holds 20 then:

Operator	Description	Example
+	Addition - Adds values on either side of the operator	a + b will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	a - b will give -10
*	Multiplication - Multiplies values on either side of the operator	a * b will give 200
/	Division - Divides left hand operand by right hand operand	b / a will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	b % a will give 0

5.11.2 SQL Comparison Operators

The comparison operators allow you to compare values of columns in a database table with a specified value. Let us assume variable ‘a’ holds 10 and variable ‘b’ holds 20 then:

Operator	Description	Example
=	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a = b) is not true.
!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.

<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.
!<	Checks if the value of left operand is not less than the value of right operand, if yes then condition becomes true.	(a !< b) is false.
!>	Checks if the value of left operand is not greater than the value of right operand, if yes then condition becomes true.	(a !> b) is true.

5.11.3 SQL Logical Operators

Here is a list of all the logical operators available in SQL.

Operator	Description
ALL	The ALL operator is used to compare a value to all values in another value set.
AND	The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause.
ANY	The ANY operator is used to compare a value to any applicable value in the list according to the condition.
BETWEEN	The BETWEEN operator is used to search for values that are within a set of values, given the minimum value and the maximum value.
EXISTS	The EXISTS operator is used to search for the presence of a row in a specified table that meets certain criteria.
IN	The IN operator is used to compare a value to a list of literal values that have been specified.
LIKE	The LIKE operator is used to compare a value to similar values using wildcard operators.
NOT	The NOT operator reverses the meaning of the logical operator with which it is used. Eg. NOT EXISTS, NOT BETWEEN, NOT IN etc. **This is negate operator.**
OR	The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.
IS NULL	The NULL operator is used to compare a value with a NULL value.
UNIQUE	The UNIQUE operator searches every row of a specified table for uniqueness (no duplicates).

5.11.4 SQL EXPRESSION

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. SQL Expression are like formulas and they are written in query language. You can also use them to query the database for specific set of data.

Syntax:

Consider the basic syntax of the SELECT statement as follows:

SELECT column1,column2…columnN FROM table_name where [CONDITION][EXPRESSION];

There are different types of SQL expression, which are mentioned below:

(a) SQL Boolean Expressions

SQL Boolean Expressions fetch the data on the basis of matching single value.

Following is the syntax of SQL Boolean expression:

SELECT column1, column2…columnN FROM table_name WHERE [SINGLE VALUE

MATCHING EXPRESSION];

Consider CUSTOMERS table that has following records:

SQL>SELECT * FROM CUSTOMERS;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	33	Delhi	2000.00
2	Ram	32	Lucknow	1500.00
3	Shyam	25	Bareilly	2000.00
4	Komal	27	MP	6500.00

In the above query * refers to selection of all rows.

Here is a simple example showing usage of SQL Boolean Expressions:

SQL> SELECT * FROM CUSTOMERS WHERE SALARY = 2000;

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	33	Delhi	2000.00

(b) SQL - Numeric Expression

This expression is used to perform any mathematical operation in any query. Following is the syntax:

SELECT numerical expression AS OPERATION_NAME

[FROM table name WHERE CONDITION] ;

Here numerical expression is used for mathematical expression or any formula. Following is a simple examples showing usage of SQL Numeric Expressions:

SQL> SELECT (12 + 6) AS ADDITION

ADDITION 18

There are several built-in functions like avg (), sum (), count () etc. to perform what is known as aggregate data calculations against a table or a specific table column.

SQL> SELECT COUNT (*) AS "RECORDS" FROM CUSTOMERS;

RECORDS 6

(c) SQL - Date Expressions

Date Expressions return current system date and time values:

SQL> SELECT CURRENT_TIMESTAMP;

Current_Timestamp

2013-07-18 05:30:23

5.11.5 CREATE DATABASE

The SQL **CREATE DATABASE** statement is used to create new SQL database.

Basic syntax of CREATE DATABASE statement is as follows:

>CREATE DATABASE DatabaseName;

Database name should be unique within the RDBMS.

Example:

To create new database <testDB>, then CREATE DATABASE statement would

be as follows:

SQL> CREATE DATABASE testDB;

Make sure you have admin previledge before creating any database. Once a database is

created, you can check it in the list of databases as follows:

SQL> SHOW DATABASES;

5.11.6 DROP

The SQL **DROP DATABASE** statement is used to drop any existing database in SQL schema.

The syntax of DROP DATABASE statement is :

DROP DATABASE DatabaseName;

Example:

To delete an existing database <tryDB>, then DROP DATABASE statement would be as follows:

SQL> DROP DATABASE tryDB;,

5.11.7 USE

The SQL **USE** statement is used to select any existing database in SQL schema.

The syntax of USE statement is as follows:

USE DatabaseName;

Example:

You can check available databases as follows:

SQL> SHOW DATABASES;

information_schema

AMROOD

TUTORIALSPOINT

mysql

orig

test

tryDB

Now if you want to work with AMROOD database then you can execute USE AMROOD SQL command and start working with AMROOD database:

5.11.8 CREATE TABLE

The SQL **CREATE TABLE** statement is used to create a new table.

The syntax of CREATE TABLE statement

CREATE TABLE table_name(column1 datatype, column2 datatype, column3 datatype,

..... columnN datatype, PRIMARY KEY(one or more columns));

CREATE TABLE is the keyword telling the database system what you want to do. The unique name or identifier for the table follows the CREATE TABLE statement.

Then in brackets comes the list defining each column in the table and what sort of data type it is. The syntax becomes clearer with an example below.

Example:

Following is an example which creates a CUSTOMERS table with ID as primary key and NOT NULL are the constraints showing that these fileds can not be NULL while creating records in this table:

SQL> CREATE TABLE CUSTOMERS (ID INT NOT NULL, NAME VARCHAR (20)

NOT NULL, AGE INT NOT NULL, ADDRESS CHAR (25), SALARY

DECIMAL (18, 2), PRIMARY KEY (ID));

You can verify when your table is created successfully by looking at the message displayed by the SQL server otherwise you can use **DESC** command as follows:

SQL> DESC CUSTOMERS

5.11.9 INSERT INTO

The SQL **INSERT INTO** statement is used to add new rows of data to a table in the database.

Syntax:

There are two basic syntax of INSERT INTO statement is follows:

(i) INSERT INTO TABLE_NAME (column1, column2, column3,...columnN)]

VALUES (value1, value2, value3,…….valueN);

Here column1, column2,...columnN are the names of the columns in the table into which you want to insert data.

(ii) INSERT INTO TABLE_NAME VALUES (value1, value2, value3,…..,valueN);

You may not need to specify the column(s) name in the SQL query if you are adding values for all the columns of the table. But make sure the order of the values is in the same order as the columns in the table. The SQL INSERT INTO syntax would be as follows:

Example:

Following statements would create four records in CUSTOMERS table:

INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (1, 'Ramesh', 32, 'Ahmedabad', 2000.00);

INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (2, 'Khilan', 25, 'Delhi', 1500.00);

INSERT INTO CUSTOMERS

VALUES (3, 'kaushik', 23, 'Kota', 2000.00);

INSERT INTO CUSTOMERS

VALUES (4, 'Chaitali', 25, 'Mumbai', 6500.00);

5.11.10 WHERE

The SQL **WHERE** clause is used to specify a condition while fetching the data from single table or joining with multiple table.

If the given condition is satisfied then only it returns specific value from the table. You would use WHERE clause to filter the records and fetching only necessary records.

The WHERE clause not only used in SELECT statement, but it is also used in UPDATE, DELETE statement etc. which we would examine in subsequent sections.

Syntax:

The syntax of SELECT statement with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table_name

WHERE [condition]

You can specify a condition using comparision or logical operators like >, <, =, LIKE, NOT etc. Below examples would make this concept clear.

Example:

Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	KOTA	2000.00
4	CHAITALI	25	MUMBAI	6500.00

Following is an example which would fetch ID, Name and Salary fields from the CUSTOMERS table where salary is greater than 2000:

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY >2000;

Output of the above query is

4	CHAITALI	6500

5.11.11 AND & OR

The SQL **AND** and **OR** operators are used to combine multiple conditions to narrow data in an SQL statement. These two operators are called conjunctive operators.

These operators provide a means to make multiple comparisons with different operators in the same SQL statement.

(a) The AND Operator:

The **AND** operator allows the existence of multiple conditions in an SQL statement's WHERE clause.

Syntax:

The syntax of AND operator with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table_name

WHERE [condition1] AND [condition2]...AND [conditionN];

You can combine N number of conditions using AND operator. For an action to be taken by the SQL statement, whether it be a transaction or query, all conditions separated by the AND must be TRUE.

(b) The OR Operator

The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause.

Syntax:

The syntax of OR operator with WHERE clause is as follows:

SELECT column1, column2, columnN

FROM table_name

WHERE [condition1] OR [condition2]...OR [conditionN]

You can combine N number of conditions using OR operator. For an action to be taken by the SQL statement, whether it be a transaction or query, only any ONE of the conditions separated by the OR must be TRUE.

5.11.12 UPDATE

The SQL **UPDATE** query is used to modify the existing records in a table.

You can use WHERE clause with UPDATE query to update selected rows otherwise all the rows would be effected.

Syntax:

The syntax of UPDATE query with WHERE clause is:

UPDATE table_name

SET column1 = value1, column2 = value2...., columnN = valueN

WHERE [condition];

You can combine N number of conditions using AND or OR operators.

5.11.13 DELETE

The SQL **DELETE** Query is used to delete the existing records from a table.

You can use WHERE clause with DELETE query to delete selected rows, otherwise all the records would be deleted.

Syntax:

The syntax of DELETE query with WHERE clause is as follows:

DELETE FROM table_name

WHERE [condition];

You can combine N number of conditions using AND or OR operators.

5.11.14 LIKE

The SQL **LIKE** clause is used to compare a value to similar values using wildcard operators. There are two wildcards used in conjunction with the LIKE operator:

- The percent sign (%)
- The underscore (_)

The percent sign represents zero, one, or multiple characters. The underscore represents a single number or character. The symbols can be used in combinations.

Syntax:

The syntax of % and _ is as follows:

SELECT FROM table_name

WHERE column LIKE 'sssssss%'

or

SELECT FROM table_name

WHERE column LIKE '%aaaaa%'

or

SELECT FROM table_name

WHERE column LIKE 'aaaaa_'

or

SELECT FROM table_name

WHERE column LIKE '_aaaa'

or

SELECT FROM table_name

WHERE column LIKE '_aaaa_'

You can combine N number of conditions using AND or OR operators. Here aaaa could be any numeric or string value.

Example:

Here are number of examples showing WHERE part having different LIKE clause with '%' and '_' operators:

Statement	**Description**
WHERE SALARY LIKE '200%'	Finds any values that start with 200
WHERE SALARY LIKE '%200%'	Finds any values that have 200 in any position
WHERE SALARY LIKE '_00%'	Finds any values that have 00 in the second and third positions
WHERE SALARY LIKE '2_%_%'	Finds any values that start with 2 and at least 3 characters in length
WHERE SALARY LIKE '%2'	Finds any values that end with 2
WHERE SALARY LIKE	Finds any values that have a 2 in the second

'_2%3'	position and end with a 3
WHERE SALARY LIKE '2___3'	Finds any values in a five-digit number that start with 2 and end with 3

5.11.15 TOP

The SQL **TOP** clause is used to fetch a TOP N number or X percent records from a table.

Note: All the databases do not support TOP clause. For example MySQL supports **LIMIT** clause to fetch limited number of records and Oracle uses **ROWNUM** to fetch limited number of records.

Syntax:

The syntax of TOP clause with SELECT statement would be as follows:

SELECT TOP number|percent column_name(s)

FROM table_name

WHERE [condition]

Example:

Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	KOTA	3000.00
4	CHAITALI	25	Pune	6500.00

Following is an example on SQL server which would fetch top 3 records from CUSTOMERS table:

SQL> SELECT TOP 3 * FROM CUSTOMERS;

This would produce following result:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	KOTA	3000.00

5.11.16 GROUP BY

The SQL **GROUP BY** clause is used in collaboration with the SELECT statement to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and precedes the ORDER BY clause.

Syntax:

The syntax of GROUP BY clause is given below. The GROUP BY clause must follow the conditions in the WHERE clause and must precede the ORDER BY clause if one is used.

SELECT column1, column2

FROM table_name

WHERE [conditions]

GROUP BY column1, column2

ORDER BY column1, column2

Example:

Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	KOTA	3000.00
4	CHAITALI	25	Pune	6500.00
5	RAMESH	26	BAREILLY	1500.00

If you want to know the total amount of salary on each customer, then GROUP BY query would be as follows:

SQL> SELECT NAME, SUM (SALARY) FROM CUSTOMERS

GROUP BY NAME;

This would produce following result:

NAME	SUM(SALARY)
RAMESH	3500.00
KHILAN	1500.00
KAUSHIK	3000.00
CHAITALI	6500.00

The SUM function returns the grand total of the records contained in a specified column of the database.

SQL> SELECT SUM (SALARY) FROM CUSTOMERS;

This would produce following result:

SUM(SALARY)
14500.00

5.11.17 DISTINCT

The SQL **DISTINCT** keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records.

There may be a situation when you have multiple duplicate records in a table. While fetching such records, it makes more sense to fetch only unique records instead of fetching duplicate records.

Syntax:

The syntax of DISTINCT keyword to eliminate duplicate records is as follows:

SELECT DISTINCT column1, column2….. columnN

FROM table_name

WHERE [condition]

Example:

Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	KOTA	3000.00
4	CHAITALI	25	Pune	6500.00
5	VIJAY	30	Chennai	2000.00

To get the distinct column values, we use the following SQL command:

SQL> SELECT DISTINCT SALARY FROM CUSTOMERS ORDER BY SALARY

SALARY
1500.00
2000.00
3000.00
6500.00

5.11.18 ORDER BY

The SQL **ORDER BY** clause is used to sort the data in ascending or descending order, based on one or more columns. Some database sorts query results in ascending order by default.

Syntax:

The syntax of ORDER BY clause which would be used to sort result in ascending or descending order is as follows:

SELECT column-list

FROM table_name

[WHERE condition]

[ORDER BY column1, column2, .. columnN] [ASC | DESC];

You can use more than one column in the ORDER BY clause. Make sure whatever column you are using to sort, that column should be in column-list.

Example:

Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	KOTA	3000.00
4	CHAITALI	25	Pune	6500.00

SQL> SELECT ID, NAME, SALARY FROM CUSTOMERS WHERE SALARY > 1500 ORDER BY SALARY;

The output of above SQL statement is:

1	RAMESH	2000
3	KAUSHIK	3000
4	CHAITALI	6500

5.12 Constraints

Constraints are the rules enforced on data columns on table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database. Contraints could be column level or table level. Column level constraints are applied only to one column where as table level constraints are applied to the whole table.

Following are commonly used constraints available in SQL

- NOT NULL Constraint: Ensures that a column cannot have NULL value.
- DEFAULT Constraint: Provides a default value for a column when none is specified.
- UNIQUE Constraint: Ensures that all values in a column are different.
- PRIMARY Key: Uniquely identified each rows/records in a database table.
- FOREIGN Key: Uniquely identified a rows/records in any another database table.
- CHECK Constraint: The CHECK constraint ensures that all values in a column satisfy certain conditions.
- INDEX: Use to create and retrieve data from the database very quickly.

Constraints can be specified when a table is created with the CREATE TABLE statement or you can use ALTER TABLE statment to create constraints even after the table is created.

5.12.1 Dropping Constraints

Any constraint that you have defined can be dropped using the ALTER TABLE command with the DROP CONSTRAINT option. For example, to drop the primary key constraint in the EMPLOYEES table, you can use the following command:

ALTER TABLE EMPLOYEES DROP CONSTRAINT EMPLOYEES_PK;

where EMPLOYEES_PK is a primary key for table EMPLOYEES.

Some implementations may provide shortcuts for dropping certain constraints. For example, to drop the primary key constraint for a table in Oracle, you can use the following command:

ALTER TABLE EMPLOYEES DROP PRIMARY KEY;

Some implementations allow you to disable constraints. Instead of permanently dropping a constraint from the database, you may want to temporarily disable the constraint, and then enable it later.

5.12.2 Integrity Constraints

Integrity constraints are used to ensure accuracy and consistency of data in a relational database. Data integrity is handled in a relational database through the concept of referential integrity.

There are many types of integrity constraints that play a role in referential integrity (RI). These constraints include Primary Key, Foreign Key, Unique Constraints and other constraints mentioned above.

5.13 SQL Join Types

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. There are different types of joins available in SQL

5.13.1 INNER JOIN: It returns rows when there is a match in both tables. The most frequently used and important of the joins is the **INNER JOIN**. It is also referred 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.

Syntax:

The basic syntax of **INNER JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
INNER JOIN table2
ON table1.common_field = table2.common_field;
```

Example

Consider the following two tables; (a) CUSTOMERS Table 5.1

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 5.1 CUSTOMERS

(b) ORDERS Table 5.2

ORDERID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08	3	3000
100	2009-10-08	3	1500
101	2009-11-20	2	1560
103	2008-05-20	4	2060

Table 5.2 ORDERS

Now, let us join these two tables using INNER JOIN as follows:

```
SELECT ID, NAME, AMOUNT, DATE
FORM CUSTOMERS
INNER JOIN ORDERS
ON CUSTOMERS.ID= ORDERS.CUSTOMER_ID;
```

This would produce the following result:

ID	NAME	AMOUNT	DATE
3	Kaushik	3000	2009-10-08
3	Kaushik	1500	2009-10-08
2	Khilan	1560	2009-11-20
4	Chaitali	2060	2008-05-20

5.13.2 LEFT 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 right table, the join will still return a row in the result, but with NULL in each column from 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 of join predicate.

Syntax:

The basic syntax of **LEFT JOIN** is as follows:

```
SELECT table1.column1, table2.column2...
FROM table1
LEFT JOIN table2
ON table1.common_filed = table2.common_field;
```

here given condition could be any given expression based on your requirement.

Let us consider the CUSTOMERS AND ORDERS table describe in Table 5.1 and Table 5.2

Now, let us join these two tables using LEFT JOIN as follows:

SELECT ID, NAME, AMOUNT, DATE

FROM CUSTOMERS

LEFT JOIN ORDERS

ON CUSTOMERS.ID=ORDERS.CUSTOMERS_ID;

This would produce the following result:

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20
3	Kaushik	3000	2009-10-08
3	Kaushik	1500	2009-10-08
4	Chaitali	2060	2008-05-20
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

5.13.3 RIGHT 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 left table, the join will still return a row in the result.but with NULL in each column from 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 of join predicate.

Syntax:

The basic syntax of **RIGHT JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

RIGHT JOIN table2

ON table1.common_filed = table2.common_field;

Let us consider the CUSTOMERS AND ORDERS table describe in table 5.1 and table 5.2.

Now, let us join these two tables using RIGHT JOIN as follows:

SELECT ID, NAME, AMOUNT, DATE

FROM CURTOMERS

RIGHT JOIN ORDERS

ON CUSTOMER.ID=ORDERS.CUSTOMER_ID

This would produce the following result

ID	NAME	AMOUNT	DATE
3	Kaushik	3000	2009-10-08
3	Kaushik	1500	2009-10-08
2	Khilan	1560	2009-11-20
4	Chaitali	2060	2008-05-20

5.13.4 FULL JOIN

The SQL **FULL JOIN** combines the results of both left and right outer joins. It returns rows when there is a match in one of the tables.

The joined table will contain all records from both tables, and fill NULLs for missing matches on either side.

Syntax:

The basic syntax of **FULL JOIN** is as follows:

SELECT table1.column1, table2.column2...

FROM table1

FULL JOIN table2

ON table1.common_filed = table2.common_field;

here given condition could be any given expression based on your requirement.

Let us consider the CUSTOMERS AND ORDERS table describe in Table 5.1 and Table 5.2.

Now, let us join these two tables using FULL JOIN as follows

SELECT ID, NAME, AMOUNT, DATE

FROM CURSOMERS

FULL JOIN ORDERS

ON CUSTOMERS.ID=ORDERS.CUSTOMER_ID;

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20
3	Kaushik	3000	2009-10-08
3	Kaushik	1500	2009-10-08
4	Chaitali	2060	2008-05-20
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	Kaushik	3000	2009-10-08
3	Kaushik	1500	2009-10-08
2	Khilan	1560	2009-11-20
4	Chaitali	2060	2008-05-20

If your Database does not support FULL JOIN like MySQL does not support FULL JOIN, then you can use **UNION ALL** clause to combine two JOINS as follows:

```
SQL> SELECT ID, NAME, AMOUNT, DATE
    FROM CUSTOMERS
    LEFT JOIN ORDERS
    ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION ALL
    SELECT ID, NAME, AMOUNT, DATE
    FROM CUSTOMERS
    RIGHT JOIN ORDERS
    ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
```

5.13.5 SELF JOIN

The SQL **SELF JOIN** is used to join a table to itself, as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax:

The basic syntax of **SELF JOIN** is as follows:

```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

here WHERE clause could be any given expression based on your requirement.

Consider the CUSTOMERS table as shown in Table 5.3

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000
2	KHILAN	25	DELHI	1500
3	KAUSHIK	23	KOTA	2000
5	HARDIK	27	BHOPAL	8500
6	KOMAL	22	MP	4500
7	SANJAY	24	INDORE	10000

Table 5.3: CUSTOMERS

Now, let us join this table using SELF JOIN as follows:

```
SELECT a.ID, b.NAME, a.SALARY
FROM CUSTOMERS a, CUSTOMERS b
```

WHERE a.SALARY < b.SALARY;

This would produce the following result:

ID	NAME	SALARY
2	RAMESH	1500
2	KAUSHIK	1500
1	CHAITALI	2000
2	CHAITALI	1500
3	CHAITALI	2000
6	CHAITALI	4500
1	HARDIK	2000
2	HARDIK	1500
3	HARDIK	2000
4	HARDIK	6500
6	HARDIK	4500
1	KOMAL	2000
2	KOMAL	1500
3	KOMAL	2000
1	SANJAY	2000
2	SANJAY	1500
3	SANJAY	2000
4	SANJAY	6500
5	SANJAY	8500
6	SANJAY	4500

5.14 UNION

The SQL **UNION** clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order but they do not have to be the same length.

Syntax:

The basic syntax of **UNION** is as follows:

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

UNION

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

here given condition could be any given expression based on your requirement.

Example:

Suppose the first table is

ID	Name
1	Ashish
2	Amit

And the second table is

ID	Name
2	Amit
3	Vijay

```
SELECT * from first

UNION

SELECT * from secomd
```

The result table will look like,

ID	Name
1	Ashish
2	Amit
3	Vijay

5.14.1 UNION ALL CLAUSE

The UNION ALL operator is used to combine the results of two SELECT statements including duplicate rows.

The same rules that apply to UNION apply to the UNION ALL operator.

Syntax:

The basic syntax of **UNION ALL** is as follows:

```
SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]

UNION ALL

SELECT column1 [, column2 ]

FROM table1 [, table2 ]

[WHERE condition]
```

here given condition could be any given expression based on your requirement.

Example:

Suppose the first table is

ID	Name
1	Ashish
2	Amit

And the second table is

ID	Name
2	Amit
3	Vijay

```
SELECT * from first
UNION ALL
SELECT * from second
```

The result table will look like,

ID	Name
1	Ashish
2	Amit
2	Amit
3	Vijay

There are two other clauses (i.e operators) which are very similar to UNION clause, they are

- INTERSECT CLAUSE
- EXCEPT CLAUSE

5.14.2 INTERSECT CLAUSE

The SQL **INTERSECT** clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. It means INTERSECT returns only common rows returned by the two SELECT statements.

Just as with the UNION operator, the same rules apply when using the INTERSECT operator. MySQL does not support INTERSECT operator.

Syntax:

The syntax of **INTERSECT** IS

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

INTERSECT

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

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

Example:

The **First** table is

ID	Name
1	Ashish
2	Amit

And the second table is

ID	Name
2	Amit
3	Vijay

Intersect query will be,
SELECT * from first
INTERSECT
SELECT * from second

The result table will look like

ID	Name
2	Amit

5.14.3 EXCEPT CLAUSE

The SQL **EXCEPT** clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement. It means EXCEPT returns only rows which are not available in second SELECT statement.

Just as with the UNION operator, the same rules apply when using the EXCEPT operator. MySQL does not support EXCEPT operator.

Syntax:

The syntax of **EXCEPT**

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

EXCEPT

SELECT column1 [, column2]

FROM table1 [, table2]

[WHERE condition]

here given condition could be any given expression based on your requirement.

5.15 NULL

The SQL **NULL** is the term used to represent a missing value. A NULL in a table is a value in a field that appears to be blank.

A field with a NULL has no value. It is very important to understand that a NULL value is different than a zero value or a field that contains spaces.

Syntax:

The basic syntax of **NULL** while creating a table:

```
SQL> CREATE TABLE CUSTOMERS (
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE INT               NOT NULL,
   ADDRESS CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

here **NOT NULL** signifies that column should always accept an explicit value of the given data type. There are two columns like ADDRESS and SALARY where we did not use NOT NULL which means these columns could be NULL.

A field with a NULL value is one that has been left blank during record creation.

Example:

The NULL value can cause problems when selecting data, however, because when comparing an unknown value to any other value, the result is always unknown and not included in the final results.

You must use the **IS NULL** or **IS NOT NULL** operators in order to check for a NULL value.

Consider the following table, CUSTOMERS that have following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	NULL	NULL
4	CHAITALI	25	NULL	NULL

Now following is the usage of **IS NOT NULL** operator:

```
SQL> SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
WHERE SALARY IS NOT NULL;
```

This would produce following result:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00

Now following is the usage of **IS NULL** operator:

```
SQL> SELECT ID, NAME, AGE, ADDRESS, SALARY
FROM CUSTOMERS
WHERE SALARY IS NULL;
```

This would produce following result:

ID	NAME	AGE	ADDRESS	SALARY
3	KAUSHIK	23	NULL	NULL
4	CHAITALI	25	NULL	NULL

5.16 TRUNCATE TABLE

The SQL **TRUNCATE TABLE** command is used to delete complete data from an existing table.
You can also use DROP TABLE command to delete complete table but it would remove complete table structure form the database and you would need to re-create this table once again if you wish to store some data.

Syntax:
The syntax of **TRUNCATE TABLE**

```
TRUNCATE TABLE  table_name;
```

Example:

Consider CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

Following is the example to truncate:

```
SQL > TRUNCATE TABLE CUSTOMERS;
```

Now CUSTOMERS table is truncated and following would be output from SELECT statement:

```
SQL> SELECT * FROM CUSTOMERS;
Empty set (0.00 sec)
```

"CHECK YOUR PROGRESS 1"

1. What is DDL?
2 What is DML?
3. What are the advantages of SQL?
4. Define join.
5. Explain group by, having clause of SQL with example.
6. Give syntax of UPDATE command. Demonstrate with suitable example.

5.17 Views

A view is a "virtual table" in the database whose contents are defined by a query. The tables of a database define the structure and organization of its data. However, SQL also lets you look at the stored data in other ways by defining alternative views of the data. A view is a SQL query that is permanently stored in the database and assigned a name. The results of the stored query are "visible" through the view, and SQL lets you access these query results as if they were, in fact, a "real" table in the database

Views are an important part of SQL, for several reasons:.

- Views let you tailor the appearance of a database so that different users see it from different perspectives.
- Views let you restrict access to data, allowing different users to see only certain rows or certain columns of a table.
- Views simplify database access by presenting the structure of the stored data in the way that is most natural for each user.

5.17.1 Advantages of VIEW

Views provide a variety of benefits and can be useful in many different types of databases. In a personal computer database, views are usually a convenience, defined to simplify database requests. In a production database installation, views play a central role in defining the structure of the database for its users and enforcing its security. Views provide these major benefits:

Security: Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data.

Query simplicity: A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.

Structural simplicity: Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.

Insulation from change: A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.

Data integrity: If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets specified integrity constraints.

5.17.2 Disadvantages of VIEW

While views provide substantial advantages, there are also two major disadvantages to using a view instead of a real table:

Performance: Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query, then even a simple query against the view becomes a complicated join, and it may take a long time to complete

Update restrictions: When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying source tables. This is possible for simple views, but more complex views cannot be updated; they are "read-only."

5.17.3 Creating a VIEW

The CREATE VIEW statement is used to create a view. The statement assigns a name to the view and specifies the query that defines the view. To create the view successfully, you must have permission to access all of the tables referenced in the query.

The CREATE VIEW statement can optionally assign a name to each column in the newly created view. If a list of column names is specified, it must have the same number of items as the number of columns produced by the query. Note that only the column names are specified; the data type, length, and other characteristics of each column are derived from the definition of the columns in the source tables. If the list of column names is omitted from the CREATE VIEW statement, each column in the view takes the name of the corresponding column in the query. The list of column names must be specified if the query includes calculated columns or if it produces two columns with identical names.

The CREATE VIEW syntax is as follows:

CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE [condition];

You can include multiple tables in your SELECT statement in very similar way as you use them in normal SQL SELECT query.

Example:

Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAMESH	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

Now, following is the example to create a view from CUSTOMERS table. This view would be used to have customer name and age from CUSTOMERS table:

```
SQL > CREATE VIEW CUSTOMERS_VIEW AS
SELECT name, age
FROM CUSTOMERS;
```

Now you can query CUSTOMERS_VIEW in similar way as you query an actual table. Following is the example:

SQL > SELECT * FROM CUSTOMERS_VIEW;

This would produce following result:

NAME	AGE
RAMESH	32
KHILAN	25
KAUSHIK	23
CHAITALI	25

5.17.4 Updating a View

A view can be updated under certain conditions:

- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.

- The SELECT clause may not contain an ORDER BY clause.
- The FROM clause may not contain multiple tables.
- The WHERE clause may not contain subqueries.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.
- All NOT NULL columns from the base table must be included in the view in order for the INSERT query to function.

So if a view satisfies all the above mentioned rules then you can update a view. Following is an example to update the age of Ram.

```
SQL > UPDATE CUSTOMERS_VIEW
SET AGE = 32
WHERE name='Ram';
```

This would ultimately update the base table CUSTOMERS and same would reflect in the view itself. Now try to query base table, and SELECT statement would produce following result:

ID	NAME	AGE	ADDRESS	SALARY
1	RAM	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

5.17.5 Dropping Views

Obviously, when you have a view, you need a way to drop the view if it is no longer needed. The syntax is very simple as given below:

```
DROP VIEW view_name;
```

Following is an example to drop CUSTOMERS_VIEW from CUSTOMERS table:

```
DROP VIEW CUSTOMERS_VIEW;
```

5.18 NESTED QUERY

A Subquery or Inner query or Nested query is a query within another SQL query, and embedded within the WHERE clause. A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved. Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN etc.

There are few rules that subqueries must follow:

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators, such as the IN operator.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery; however, the BETWEEN can be used within the subquery.

Subqueries with the SELECT Statement:

Subqueries are most frequently used with the SELECT statement. The syntax is as follows:

```
SELECT column_name [, column_name ]
FROM   table1 [, table2 ]
WHERE  column_name OPERATOR
    (SELECT column_name [, column_name ]
    FROM table1 [, table2 ]
    [WHERE])
```

Example:

Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAM	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

Now let us check following sub-query with SELECT statement:

```
SQL> SELECT *
   FROM CUSTOMERS
   WHERE ID IN (SELECT ID
            FROM CUSTOMERS
            WHERE SALARY > 2000) ;
```

This would produce following result:

ID	NAME	AGE	ADDRESS	SALARY
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

5.19 SQL Aggregate Functions

SQL aggregate functions return a single value, calculated from values in a column. Some of the useful aggregate functions are:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum
- Consider the query "Find the **average** account balance at the Perryridge branch."We write this query as follows:

SELECT AVG (balance)

FROM account

where branch-name = 'Perryridge';

The result of this query is a relation with a single attribute, containing a single tuple with a numerical value corresponding to the average balance at the Perryridge branch.

- Consider the query "Find the **minimum** salary offered to a employee."We write this query as follows:

SELECT MIN (salary)

FROM employee;

The result of this query is a relation with a single attribute, containing a single tuple with a numerical value corresponding to the minimum salary offered to an employee.

- Consider the query "Find the **maximum** salary offered to a employee."We write this query as follows:

SELECT MAX (salary)

FROM employee;

The result of this query is a relation with a single attribute, containing a single tuple with a numerical value corresponding to the maximum salary offered to an employee.

- To find the number of tuples in the customer relation, we write

  SELECT **COUNT** (*)

  FROM customer;

The result of this query is a relation with a single attribute, containing a single tuple with a numerical value corresponding to the total number of customers present in the customer table.

- To find the total salary issued to the employees we write the query:

  SELECT SUM (salary)

  FROM employee;

The result of this query is a relation with a single attribute, containing a single tuple with a numerical value corresponding to the addition of the salaries offered to all the employees.

5.19.1 SQL FIRST () Function

SQL SELECT FIRST () function returns the first value of selected column.

Syntax:

SELECT FIRST (Column_name) FROM table_name;

OR

SELECT FIRST (Column_name) AS First_Name FROM table_name;

Example: Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAM	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

Write a query to display a first name from table 'CUSTOMERS'.

SELECT FIRST (Name) AS First_name form CUSTOMERS;

The result is shown in the following table.

First_name
Ram

5.19.2 SQL LAST () Function

SQL SELECT LAST() function returns the last value of selected column.

Syntax:

SELECT LAST (Column_name) FROM table_name;

OR

SELECT LAST (Column_name) AS Last_Name FROM table_name;

Example: Consider the CUSTOMERS table having following records:

ID	NAME	AGE	ADDRESS	SALARY
1	RAM	32	AHMEDABAD	2000.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

Write a query to display a last name from table 'CUSTOMERS'.

SELECT LAST (Name) AS Last_name form CUSTOMERS;

The result is shown in the following table.
Last_name
CHAITALI

5.19.3 MINUS (Difference)

Minus returns the rows from the first query that were not present in the second. For example:

TABLE: (A)

ID	NAME	AGE	ADDRESS	SALARY
1	RAM	32	AHMEDABAD	2250.00
2	KHILAN	25	DELHI	1500.00
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

TABLE: (B)

ID	NAME	AGE	ADDRESS	SALARY
3	KAUSHIK	23	MUMBAI	3500.00
4	CHAITALI	25	PUNE	4500.00

```
SELECT * A
MINUS
SELECT * B;
```

The output of this query is:

ID	NAME	AGE	ADDRESS	SALARY
1	RAM	32	AHMEDABAD	2250.00
2	KHILAN	25	DELHI	1500.00

5.20 Cursors

A database cursor is similar to the cursor on a word processor screen. As you press the Down Arrow key, the cursor scrolls down through the text one line at a time. Pressing the Up Arrow key scrolls your cursor up one line at a time. Hitting other keys such as Page Up and Page Down results in a leap of several lines in either direction. Database cursors operate in the same way. Database cursors enable you to select a group of data, scroll through the group of records (often called a recordset), and examine each individual line of data as the cursor points to it. You can use a combination of local variables and a cursor to individually examine each record and perform any external operation needed before moving on to the next record.

One other common use of cursors is to save a query's results for later use. A cursor's result set is created from the result set of a **SELECT** query. If your application or procedure requires the repeated use of a set of records, it is faster to create a cursor once and reuse it several times than to repeatedly query the database. (And you have the added advantage of being able to scroll through the query's result set with a cursor.)

Follow these steps to create, use, and close a database cursor:

1. Create the cursor.
2. Open the cursor for use within the procedure or application.
3. Fetch a record's data one row at a time until you have reached the end of the cursor's records.
4. Close the cursor when you are finished with it.
5. Deallocate the cursor to completely discard it.

5.20.1 Creating a Cursor

To create a cursor using SQL, issue the following syntax:

>declare cursor_name cursor

>for select_statement

>[for {read only | update [of column_name_list]}]

The Oracle7 SQL syntax used to create a cursor looks like this:

SYNTAX:

>DECLARE cursor_name CURSOR

>FOR {SELECT command | statement_name | block_name}

By executing the **DECLARE cursor_name CURSOR** statement, you have defined the cursor result set that will be used for all your cursor operations. A cursor has two important parts: the cursor result set and the cursor position.

The following statement creates a cursor based on the **ARTISTS** table:

INPUT:

> create Artists_Cursor cursor

> for select * from ARTISTS

> go

ANALYSIS:

You now have a simple cursor object named **Artists_Cursor** that contains all the records in the **ARTISTS** table. But first you must open the cursor.

5.20.2 Opening a Cursor

The simple command to open a cursor for use is

SYNTAX:

>open cursor_name

Executing the following statement opens **Artists_Cursor** for use:

> open Artists_Cursor

> go

Now you can use the cursor to scroll through the result set.

5.20.3 Scrolling a Cursor

To scroll through the cursor's result set, Transact-SQL provides the following **FETCH** command.

SYNTAX:

>fetch cursor_name [into fetch_target_list]

Oracle SQL provides the following syntax:

>FETCH cursor_name {INTO : host_variable

>[[INDICATOR] : indicator_variable]

>[, : host_variable

>[[INDICATOR] : indicator_variable]]...

>| USING DESCRIPTOR descriptor }

Each time the **FETCH** command is executed, the cursor pointer advances through the result set one row at a time. If desired, data from each row can be fetched into the **fetch_target_list** variables.

NOTE: Transact-SQL enables the programmer to advance more than one row at a time by using the following command: **set cursor rows** number **for cursor_name**. This command cannot be used with the **INTO** clause,

It is useful only to jump forward a known number of rows instead of repeatedly executing the

FETCH statement.

The following statements fetch the data from the **Artists_Cursor** result set and return the data to the program variables:

INPUT:

> declare @name char(30)

> declare @homebase char(40)

> declare @style char(20)

> declare @artist_id int

> fetch Artists_Cursor into @name, @homebase, @style, @artist_id

> print @name

> print @homebase

> print @style

> print char(@artist_id)

> go

5.20.4 Testing a Cursor's Status

Transact-SQL enables you to check the status of the cursor at any time through the maintenance of two global variables: @@sqlstatus and @@rowcount.

The @@sqlstatus variable returns status information concerning the last executed FETCH statement. (The Transact-SQL documentation states that no command other than the FETCH statement can modify the @@sqlstatus variable.) This variable contains one of three values. The following table appears in the Transact-SQL reference manuals:

Status Meaning

- Successful completion of the FETCH statement.
- The FETCH statement resulted in an error.
- There is no more data in the result set.

The **@@rowcount** variable contains the number of rows returned from the cursor's result set up to the previous fetch. You can use this number to determine the number of records in a cursor's result set.

The following code extends the statements executed during the discussion of the **FETCH** statement. You now use the **WHILE** loop with the **@@sqlstatus** variable to scroll the cursor:

INPUT:

```
> declare @name char(30)
> declare @homebase char(40)
> declare @style char(20)
> declare @artist_id int
> fetch Artists_Cursor into @name, @homebase, @style, @artist_id
> while (@@sqlstatus = 0)
> begin
> print @name
> print @homebase
> print @style
> print char(@artist_id)
> fetch Artists_Cursor into @name, @homebase, @style, @artist_id
> end
> go
```

ANALYSIS:

Now you have a fully functioning cursor! The only step left is to close the cursor.

5.20.5 Closing a Cursor

Closing a cursor is a very simple matter. The statement to close a cursor is as follows:

SYNTAX:

```
>close cursor_name
```

This cursor still exists; however, it must be reopened. Closing a cursor essentially closes out its result set, not its entire existence. When you are completely finished with a cursor, the DEALLOCATE command frees the memory associated with a cursor and frees the cursor name for reuse. The DEALLOCATE statement syntax is as follows:

SYNTAX:

>deallocate cursor cursor_name

The following example illustrates the complete process of creating a cursor, using it, and then closing it, using SQL.

Example

INPUT:

> declare @name char(30)

> declare @homebase char(40)

> declare @style char(20)

> declare @artist_id int

> create Artists_Cursor cursor

> for select * from ARTISTS

> open Artists_Cursor

> fetch Artists_Cursor into @name, @homebase, @style, @artist_id

> while (@@sqlstatus = 0)

> begin

> print @name

> print @homebase

> print @style

> print char(@artist_id)

> fetch Artists_Cursor into @name, @homebase, @style, @artist_id

> end

> close Artists_Cursor

> deallocate cursor Artists_Cursor

> go

NOTE: The following is sample data only.

OUTPUT:

Soul Asylum Minneapolis Rock 1

Maurice Ravel France Classical 2

Dave Matthews Band Charlottesville Rock 3

Vince Gill Nashville Country 4

Oingo Boingo Los Angeles Pop 5

Crowded House New Zealand Pop 6

Mary Chapin-Carpenter Nashville Country 7

Edward MacDowell U.S.A. Classical 8

5.20.6 The Scope of Cursors

Unlike tables, indexes, and other objects such as triggers and stored procedures, cursors do not exist as database objects after they are created. Instead, cursors have a limited scope of use.

5.21 TRIGGERS

Triggers are stored programs, which are automatically executed or fired when some events occur. Triggers are, in fact, written to be executed in response to any of the following events:

- A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).
- A database definition (DDL) statement (CREATES, ALTER, or DROP).

Triggers could be defined on the table, view, schema, or database with which the event is associated.

5.21.1 Advantages of Triggers

Triggers can be written for the following purposes:

- Generating some derived column values automatically
- Enforcing referential integrity
- Event logging and storing information on table access
- Auditing
- Synchronous replication of tables
- Imposing security authorizations
- Preventing invalid transactions

5.21.2 Creating Triggers

The syntax for creating a trigger is:

CREATE [OR REPLACE] TRIGGER trigger_name

{ BEFORE |AFTER | IBSTEAD OF }

{INSERT [OR] | UPDATE [0R] | DELETE}

[OF col_name]

ON table_name

[REFERENCING OLD AS o AS NEW AS N]

[FOR EACH ROW]

WHEN (condition)

DECLARE

Declaration-statements

BEGIN

Executable-statements

EXCEPTION

Exception-handling-statements

END;

Where,

- CREATE [OR REPLACE] TRIGGER trigger_name: Creates or replaces an existing trigger with the *trigger_name*.
- {BEFORE | AFTER | INSTEAD OF}: This specifies when the trigger would be executed. The INSTEAD OF clause is used for creating trigger on a view.
- {INSERT [OR] | UPDATE [OR] | DELETE}: This specifies the DML operation.
- [OF col_name]: This specifies the column name that would be updated.
- [ON table_name]: This specifies the name of the table associated with the trigger.
- [REFERENCING OLD AS o NEW AS n]: This allows you to refer new and old values for various DML statements, like INSERT, UPDATE, and DELETE.
- [FOR EACH ROW]: This specifies a row level trigger, i.e., the trigger would be executed for each row being affected. Otherwise the trigger will execute just once when the SQL statement is executed, which is called a table level trigger.
- WHEN (condition): This provides a condition for rows for which the trigger would fire. This clause is valid only for row level triggers.

Example: Let us consider the customers table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

The following program creates a **row level** trigger for the customers table that would fire for INSERT or UPDATE or DELETE operations performed on the CUSTOMERS table. This trigger will display the salary difference between the old values and new values:

```
CREATE OR REPLACE TRIGGER display_salary_changes

BEFORE DELETE OR INSERT OR UPDATE ON customers

FOR EACH ROW

WHEN (NEW.ID>0)

DECLARE

sal_diff number;

BEGIN

sal_diff := :New.salary - :OLD.salary:

dbms_output.put_line('Old salary: ' || : OLD.salary);

dbms_output.put_line('New salary: ' || : NEW.salary);

dbms_output.put_line(' Salary difference: ' || sal_diff);

END;/
```

When the above code is executed at SQL prompt, it produces the following result:

Trigger created.

Here following two points are important and should be noted carefully:

- OLD and NEW references are not available for table level triggers, rather you can use them for record level triggers.
- If you want to query the table in the same trigger, then you should use the AFTER keyword, because triggers can query the table or change it again only after the initial changes are applied and the table is back in a consistent state.
- Above trigger has been written in such a way that it will fire before any DELETE or INSERT or UPDATE operation on the table, but you can write your trigger on a single or multiple operations, for

example BEFORE DELETE, which will fire whenever a record will be deleted using DELETE operation on the table.

5.21.3 Triggering a Trigger

Let us perform some DML operations on the CUSTOMERS table. Here is one INSERT statement, which will create a new record in the table:

INSERT INTO CUSTOMERS (ID, NAME, AGE, ADDRESS, SALARY)

VALUES (7, 'KRITI', 22, 'HP', 7500.00);

When a record is created in CUSTOMERS table, above create trigger **display_salary_changes** will be fired and it will display the following result:

Old salary;

New salary: 7500

Salary difference:

Because this is a new record so old salary is not available and above result is coming as null. Now, let us perform one more DML operation on the CUSTOMERS table. Here is one UPDATE statement, which will update an existing record in the table:

UPDATE customers

SET salary= salary + 500

WHERE id=2;

When a record is updated in CUSTOMERS table, above create trigger **display_salary_changes** will be fired and it will display the following result:

Old salary: 1500

New salary: 2000

Salary difference: 500

"CHECK YOUR PROGRESS 2"

1 What is Cursor?
2. List advantages of triggers.
3. Explain aggregate function.
4. Define view.
5. What are the SQL constructs to modify the structure of tables, views and to destroy the tables and views?
6. What are the various characteristics of SQL? Discuss five aggregate functions with a suitable example.
7. What do you mean by Query and sub-query? Discuss cursors in SQL also.
8. What are triggers? How to create triggers?

5.22 SUMMERY

- Structured Query Language (SQL) is a comprehensive language for controlling and interacting with a database management system. It is non-procedural in nature. It can be also define as a query language that allows access to data residing in database management system.
- A table is uniquely identified by its name and consists of rows that contain the stored information and column that describes an attributes of the tuples.
- Aggregate functions are statistical function such as COUNT, MIN, MAX, AVG etc. they are used to compute to describes an attributes of the tuples.
- A view is a virtual table where data is not stored physically but give the convenient method to retrieve and manipulate users to make simple queries to retrieve the results from complicated queries. It is of two types: simple view and complex view.
- Assertion is a method of expressing a condition that we want database to satisfy always.
- Triggers are stored programs, which are automatically executed or fired when some events occur.
- The cursor is a database object that allows you to manipulate data contained within the work area assigned to execute an SQL statement.

5.23 OBJECTIVE TYPE QUESTIONS

Q1. The statement in SQL which allows changing the definition of a table is

(A) Alter. **(B)** Update.

(C) Create **(D)** select.

Ans: **A**

Q2. Relational Algebra is.

(A) Data Definition Language. **(B)** Meta Language

(C) Procedural query Language **(D)** None of the above

Ans: C

Q3. Which of the following is correct?

(A) a SQL query automatically eliminates duplicates.

(B) SQL permits attribute names to be repeated in the same relation.

(C) a SQL query will not work if there are no indexes on the relations

(D) None of these

Ans: D

Q4. Which of the following is a valid SQL type?

(A) CHARACTER **(B)** NUMERIC

(C) FLOAT **(D)** All of the above

Ans: D

Q5. Which of the following is an advantage of view?

(A) Data security **(B)** Derived columns

(C) Hiding of complex queries **(D)** All of the above

Ans: D

Q6. Which of the following is a legal expression in SQL?

(A) SELECT NULL FROM EMPLOYEE;

(B) SELECT NAME FROM EMPLOYEE;

(C) SELECT NAME FROM EMPLOYEE WHERE SALARY = NULL;

(D) None of the above

Ans: B

Q7. The result of the UNION operation between R1 and R2 is a relation that includes.

(A) all the tuples of R1

(B) all the tuples of R2

(C) all the tuples of R1 and R2

(D) all the tuples of R1 and R2 which have common columns

Ans: D

Q8. A file manipulation command that extracts some of the records from a file is called

(A) SELECT **(B)** PROJECT

(C) JOIN **(D)** PRODUCT

Ans: A

Q9. A table joined with itself is called

(A) Join **(B)** Self Join

(C) Outer Join **(D)** Equi Join

Ans: B

Q10. The _____ operator is used to compare a value to a list of literals values that have been

specified.

(A) BETWEEN **(B)** ANY

(C) IN **(D)** ALL

Ans: A

5.24 SELECTED EXERCISE

(1) What is the Differences between the Table and View? What are the various advantage of using views. Explain the simple and complex views with suitable example?

(2) What are the types of Joins in SQL?

(3) Explain the cursors with example?

(4) Define a NULL value? How do you retrieve null values from the database?

(5) How do you use views during the application development?

(6) Database consists of the following tables:

STUDENT(S#, Sname)

COURSE(C#,Title,Teacher_name)

RESULT(S#,C#,Marks)

Write queries in SQL

(i) List of students who appear in all courses tough by a teacher named Dr. Sharma & marks scored more than 60 Marks.

(iii) The subject titles in which there are maximum failure

(7) Consider the following table:

Employee (Emp_Name, Dept_Name, Salary)

Write SQL statements for the following:

(i) Find the employee name who is getting lowest salary.

(ii) Find the department name which has highest average salary.

(iii) Find all the department where more than 60 employees are working.

(iv) Find all employees whose salary is higher than the average salary of their department.

(8) (a) Write SQL statement to create these tables:

Suppliers(sid: integer, sname: string, address: string)

Parts(pid: integer, pname: string, color: string)

Catalog(sid: integer, pid: integer, cost: real)

(b) Solve the following queries:

(i) Find the names of suppliers who supply some red part.

(ii) Find the most expansive parts supplied by suppliers named Mahesh.

(iii) Find the parts supplied by every supplier at less than Rs. 200.

(iv) Find the supplier names of the suppliers who supply a red part that costs less than Rs. 100.

(9) (a) Schema defined for Employee Management System is:

Employee: EmpID, Name, Address, Department, Designation, Salary

Department: DeptID, Name, HeadID

Create and insert data for the above schema.

(b) Write SQL queries for the following and show the results:

(i) Retrieve the details of employee who gets the maximum salary.

(ii) List names of all employees who earn more than Rs. 1, 00,000 in a year.

(iii) Give the name of the employee who heads the department where employee with EmpID 3 works.

(10) Consider Following Schema

Employee (ENO, ENAME, Department, Designation, DOJ, Salary, Dept_Location)

Solve the following query

(i) List the employees having Designation as "Manager" and .Dept_Location as "Mumbai"

(ii) Set the salary as Rs.50,000/- having Designation as "Project Leader"

(iii) List ENO, ENAME, Salary of employees having Salary between Rs. 20,000/- to Rs.30,000/-

(iv) List Ename of employees having 2nd alphabet in the name as "A".

(11) Given the relational schema:

ENROLL(S#, C#, Section), S# is student number.

TEACH (Prof, C#, Section), C# is course number.

ADVICE (Prof, S#), Prof is Thesis advisor of S#.

PRE-REQ(C#, pre-C#), pre-C# is prerequisite course.

GRADE (S#, C#, grade, year)

STUDENT (S#, Sname), Sname is student name.

Give queries expressed in SQL

(i) List of students taking courses with Ajay or Vijay.

(ii) List all students taking at least one course that their advisor teaches.

(iii) List those professors who teach more than one section of the same course.

(iv) List all students number and course number.

(v) List the student number and course number who got grade A

(12) Consider following Relational Algebra schema

STUDENT (RNO, Name, DOB, Percentage, DNO)

DEPARTMENT (DNO, DNAME, HEAD)

Solve the following query using SQL

i. Find Student's name and course from Computer Department

ii. Get the Student's name who has percentage greater than 70.

(13) For the following relational database write the expressions in SQL.

Branch Schema (branch name, Branchcity, Assets)

Customer schema (customername, customerstreet, customercity)

Loan schema (Branchname, loannumber, Amount)

Borrower schema (customername, loannumber)

Account schema (Branchname, Accountnumber, balance)

Depositor schema (customername, Accountnumber)

i. Find the names of all branches in Loan Schema?

ii. Find all customers having loan, account or both at bank?

iii. Display customer names in alphabetical order who have a loan at the Perryridge branch?

iv. Find set of all customers who have an account at the bank?

UNIT- 6

DATABASE DESIGN

Structure

6.1 INTRODUCTION

Database Normalization, or data normalization, is a technique to organize the contents of the table for transactional database and for the data warehouses. Normalization is a part of the successful database design; without normalization, database systems may be inaccurate, slow, and inefficient, and they might not produce the data we expect.

The primary reason for the normalizing databases is that normalization is the potent weapon against the possible corruption of databases stemming from what are called insertion, deletion & update anomalies.

Normalization can be viewed as a series of steps one after another, to deal with ways in which tables can be too complicated. These steps are known as First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF).

6.2 OBJECTIVES

After the end of this unit, you should be able to:

- Describe the various types of anomalies.
- Describe the normalization and the various concepts such as primary key and functional dependencies used in normalization.
- Explaining the first, second and third normal forms applied on a database in normalization technique.
- Explaining transitive dependency, decomposition, attribute preservation and multivalued dependency.
- Describing the fourth and fifth normal forms with join dependency.

6.3 DATABASE ANOMALIES

The goal of designing a database schema is to minimize the storage space, which is occupied by data stored on the hard drive. Database anomalies are the errors in data contained in the database that reduces the performance of database management system (DBMS). The database anomalies also affect the performance of DBMS by increasing the size of data files. The following types of database anomalies can increase the size of data files:

- **Insertion anomalies:** These occur when it becomes difficult to insert data in the database. You cannot insert data having null values in a table, which has a primary key constraint. So when you have a record that contains values for all the columns apart from the

primary key column, you cannot insert that record into the table. This restricts the ability of inserting the records into the database.

- **Deletion anomalies:** When any particular relation in the database is affected by the deletion of a particular record, deletion anomalies occur. For example, in a database, a table contains the record of students. The subject column of the table contains the information about the student, which the students have opted. Now if you delete all the records for the multimedia subject, then you can lose the information about the students who are studying only multimedia.
- **Modification anomalies:** These occur when a database user changes the value of a data item and the value of that data item does not change in other tables.

6.4 NORMALIZATION

Normalization is the process of eliminating redundancy of data in a database. A relational table in a database is said to be in a normal form if it satisfies certain constraints. The normalization process involves various levels of normal forms that allow you to separate data into multiple related tables. The common examples of normal forms are : first normal form(INF), second normal form(2NF), third normal form(3NF), fourth normal form(4NF) and fifth normal form(5NF). The goals of normalization are:

- Elimination of redundant data storage.
- Ensuring that only related data is stored in a table.

Therefore, normalization helps you to remove data redundancy and update inconsistencies when data is inserted, deleted or modified in a database. A normalized relational database provides several benefits:

- Provide better overall database organization and data consistency with a database.
- Allows you to create tables that can be easily joined with other tables with related information.
- Helps to reduce redundant data across the table.
- Prevents data loss by assigning primary and foreign keys in a table.
- Helps to reduce modification anomalies such as deletion, insertion and update anomalies.

6.5 NORMALIZATION TERMINOLOGY

Normalization terminology consists of various concepts frequently used in normalization such as primary key and functional dependency.

6.5.1 PRIMARY KEY

The primary key of a relational table uniquely identifies each row in a table. A primary key is either a column in a table that is unique such as identification number and social security number. Primary key is a set of single column or multiple columns from a table. For example, consider the student record database that contains tables related to student's information. The table STUDENTS consists of various attributes such as student_id, first_name, last_name and student_stream.

student_id	first_name	last_name	student_stream
S01	Sanjeev	Gangwar	Computers
S02	Amit	Yadav	Electronics
S03	Sanjay	Rawar	Electrical

Table 6.1: the STUDENTS table

A unique student_id number of a student is a primary key in the STUDENTS Table 6.1. You cannot make the first or last name of a student a primary key because two or more students may have the same first name and stream

6.5.2 FUNCTIONAL DEPENDENCY

A functional dependency is a constraint between two sets of attributes from the database. Functional dependency is represented by X-->Y between two attributes, X and Y in a table. The functional dependency X-->Y implies that Y is functionally dependent on X. Table 6.2 lists the various attributes in the EMPLOYEE table.

employee_id	employee_name	employee_dept
E01	Sanjeev	Gangwar
E02	Amit	Yadav
E03	Sanjay	Rawar

Table 6.2: the EMPLOYEE Table

In Table 6.2 the various attributes of the EMPLOYEE are employee_id, employee_name and employee_dept. You can state that:

employee_id→ employee_name

the above representation states that the employee_name attribute is functionally dependent on the employee_id which implies that the name of an employee can be uniquely identified from the id of the employee. However you cannot uniquely identify the employee_id from the employee_name column because more than one employee can have the

same name. However each employee has different value in the employee_id column.

Functional dependency is a type of constraint based on keys such as primary key or foreign key. For a relation table R, a column Y is functionally dependent on a column X of the same table if each value of the column X is associated with only one value of the column Y at a given time. All the columns in the relational table R should be functionally dependent on X if the column X is a primary key.

If the columns X and Y of a relation R are functionally dependent, the functional dependency can be represented as:

$$R.x \rightarrow R.y$$

For example consider the following functional dependency in a table

$$employee\_id \rightarrow salary$$

the column employee_id functionally determines the salary column because the salary of each employee is unique and remains the same for an employee, each time the name of the employee appears in the table.

A functional dependency represented by $X \rightarrow Y$ between two sets of attributes X and Y that are subsets of R, is called as trivial functional dependency if Y is a subset of X. For example employee_id→project is a trivial functional dependency.

6.6 INFERENCE RULES AND ATTRIBUTE CLOSURE

Inference rule for functional dependencies define the new dependencies, which can exist between two relations. The inference rules help deduce these new dependencies from a given set of functional dependencies F. The set of dependencies which cannot be specified is called the closure of F and is denoted by F+. Following are the six inference rules of functional dependencies F.

- **1R1 (reflexive rule):** if $X \subseteq Y$ then $X \rightarrow Y$: this rule states that if $X \subseteq Y$ and two tuples t1 and t2 exist in a relation instance r of relation R such that t1[X]=t2[X]. Now t1[X] =t2[X] because $X \subseteq Y$. this implies that $X \rightarrow Y$ holds true in relation instance r of relation R.
- **1R2(augmentation rule):** $\{X \rightarrow Y\} \models XZ \rightarrow YZ$: this rule states that if $X \rightarrow Y$ holds true in a relation instance r of R but $XZ \rightarrow YZ$ does not exist, then tuple t1 and t2 must exist in relation R.
- **IR3(transitive rule):** $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$
- **IR4 (decomposition or projective rule):** this rule states that if $X \rightarrow YZ$ holds true then $X \rightarrow Y$ and $X \rightarrow Z$ also hold true.

- **1R5 (union or additive rule):** this rule states that if X→Y and X→Z hold true then in the relation R, X→YZ also hold true.
- **1R6 (pseudo transitive rule):** this rule states that if X→Y and WY→Z hold true, then WX→Y also holds true.

Attribute Closure

To compute the closure J^+ of a given set J of functional dependencies, you can apply the inference rules until they stop producing new functional dependencies. You can test whether a set of attributes, J is a super key or not, by finding the set of attributes, which are functionally determined by J. You can use the following algorithm to compute the closure $J^{+:}$

result :=J

while (changes to result) do

for each functional dependency B→C in F do

begin

if B ⊆result;

then result := result ∪ C

end

the above code assumes that J is a set of attributes and you can call the set of attributes determined by a set F of functional dependencies. The closure of J under F is denoted by J^+.

Let us discuss this algorithm with an example;

Assume a relation schema R = (A, B, C) with the set of functional dependencies F = {A → B, B → C}. Now, we can find the attribute closure of attribute A as follows;

Step 1: We start with the attribute in question as the initial result. Hence, result = A.

Step 2: Take the first FD A → B. Its left hand side (i.e, A) is in the result, hence the right hand side can be included with the result. This lead to result = AB.

Step 3: Take the second FD B → C. Its left hand side (i.e, B) is in the result (or subset of result), hence the right hand side can be included with the result. Now, result = ABC.

We have no more attributes. Hence the algorithm exits. As the result, J^+ includes all the attributes of relation R. Now we would say J^+ is ABC. And, A is one of the keys of the relation R.

6.7 FIRST NORMAL FORM

If data in the table has an identifying key and does not include repeating groups of data, the table is said to be in 1NF. To reduce data redundancy by using first normal form you need to

- Remove the duplicate columns from a table.
- Create a separate table for related data and identify the primary key in the table.

According to the first normal form, a table should be atomic which implies that no duplicate data exists within the same row of a table. For example consider the items in Table 6.3.

Order_No	Item 1	Item1_Qty	Item1_Price	Item 2	Item2_Qty	Item2_Price
011	IT90	322	36$	IT91	564	45$

Table 6.3: The Items Table

In Table 6.3 the information provided is redundant. The multiple values of same type are stored in multiple columns such as quantity and price of two items are stored in different columns.

The requirement of first normal form is:

- Eliminate the multivalve fields from the table.
- Each column in the table must be atomic.
- Each column in the table must have a key such as primary or foreign key.
- Remove the repeating information from the table.

6.7.1 Eliminate the Multivalued Columns from the Table

The first requirement to apply 1NF to a table is to ensure that the table does not contain mutivalued columns. For example; consider a Books table with attributes book_name, book_author, book_ISBNno, book_price, book_publisher and book_category. Table 6.4 lists the various attributes in the Books table.

book_author	book_ISBNno	book_price	book_publisher	book_category
John Wilkins	8790478	35	ABC	Sales
Chris Burton	8790388	25	PQR	Accounts
Ken Wilkins	8790689	77	ABC	Sales

Table 6.4: The Books Table

In the table, since a book can have more than one author and also a book can be included in different categories, therefore such columns that consist of multivalued elements should be removed from the table. Therefore the Books table should contain book_ISBNno, book_price, and book_publisher.

Table 6.5 lists the various attributes of the Books table after the multivalued elements are removed.

book_ISBNno	book_price	book_publisher
8790478	35	ABC
8790388	25	PQR
8790689	77	ABC

Table 6.5: The Books Table after the multivalued elements are removed

Table 6.6 lists the Book category table.

book_ISBNno	book_category
8790478	Sales
8790388	Accounts
8790689	Sales

Table 6.6: The Books category table

6.7.2 Each Column in the Table is Atomic

You need to ensure that each column in a table that is to be normalized is atomic. For example, the author table can be divided into two columns which are first name and last name to make the table atomic.

Table 6.7 lists the various attributes in the author table.

book_ISBNno	First_name	Last_name
8790478	John Wilkins	Wilkins
8790388	Chris Burton	Burton
8790689	Ken Wilkins	Wilkins

Table 6.7: The Author Table

6.7.3 Each Column Must Have a Key

You can determine if each column in a table contains unique value by verifying the keys in table. The various key that can exist in a table are:

- **Super key:** refers to one or more than one column that identifies a unique row within a table,
- **Candidate key:** refers to the super key consisting of minimum number of columns required for identifying a unique row in a table.
- **Primary key:** refers to the candidate key required to uniquely identify a row in a table.
- **Alternate key:** refers to the candidate key which is not selected as a primary key.
- **Foreign key:** refers to one or more than one column in a table that matches a candidate key in the same or different table. A row in a table may be linked to a row in another table by using a foreign key.

In the Books table, the super keys are Book_author and book_ISBNno. The super keys for the author table are the combination of first_name and last_name.

The primary key for the Books table is Book_ISBNno and the primary keys for the auther table are first_name and last_name to ensure that each row in the author table is unique you can add the Author_city and Author_zipcode columns in the primary key field.

Table 6.8 lists the various attributes in the author table.

Book_author	book_ISBNno	Author_address	Author_phoneno
John Wilkins	8790478	Houston	0067675467
Chris Burton	8790388	New York	0087658945
Ken Wilkins	8790689	Denver	0067345275

Table 6.8: The Author Table

6.7.4 Eliminate the Repeating values

To make a table compliant with 1NF you need to eliminate the repeated values from the table. For example, in the Books table the publisher column can contain same values for different books. Therefore, to remove the repeating values you can make a separate table, publisher with attributes Publisher_id and Publisher_name. The publisher_id can be identified as the primary key for the publisher table.

Table 6.9 lists the various attributes in the Publisher table.

Publisher_id	Publisher_name
P0240	ABC
P0240	PQR
P0240	ABC

Table 6.9: The Publisher Table

6.8 PARTIAL DEPENDENCY

In a table, a primary key consists of one or more than one column to uniquely identify each row in the table. Partial dependency occurs when a row of a table is uniquely identified by one column that constitutes a primary key without requiring the entire primary key to uniquely identify the row. For example consider a table 'Stocks' with attributes cust_id, stock and stock_price.

Table 6.10 lists the various attributes in the Stock table.

cust_id	stock	Stock_price
C012	stk1	15
C013	stk2	10
C014	stk3	20

Table 6.10: **The Stocks Table**

In the above table, suppose cust_id and stock are identified as the primary key for the Stocks table. However the column stock_price is partially dependent on the primary key because only the stock column determines the stock_price. Also the values in the stock_price column do not need the cust_id column to uniquely identify the price of the stocks. Therefore you need to make a separate table for the stock_price where the stock column is the primary key. In the new table, partial dependency is eliminated because the stock_price column depends on the entire primary key.

Partial dependencies can only occur when more than one field constitutes the primary key. If there is only one field in the primary identifier then partial dependencies cannot occur.

6.9 SECOND NORMAL FORM

A table is in 2NF if the table satisfies all the conditions of first normal form and does not consist of any column that depends on only one part of the identified primary key. The 2NF is based on the concept of full dependency.

To apply 2NF to a table you need to:

- Ensure that the table confirms to INF.
- Create a separate table for sets of values that are applicable to multiple records.
- Relate these tables with a foreign key.

Table 6.11 lists the various attributes in the Emp_project table

Emp_id	Proj_no	Proj_hrs	Emp_name	Proj_name	Proj_loc
1	P34	10	Ravi	Sales	Delhi
2	P45	04	Kisan	Account	Noida

Table 6.11: **The Emp_project Table**

The above table Emp_project confirms to 1NF since it does not contain repeated values and Emp_id and Proj_no are identified as the primary keys for the table. However you need not confuse that the table is in 2NF because all the columns of the table depend on only a part of the primary key, which comprises Emp_id and Proj_no identified for the table. For example the column Emp_name is dependent on only the Emp_id and does not depend on the Proj_no part of the primary key. Similarly the Proj_name column is dependent on only the Proj_no column and not on the Emp_id primary key.

Therefore to apply 2NF to the Emp_project table you need to make a separate table for columns thet depend on only a part of the primary key. The new table should contain columns that are dependent on the entire primary key identified for the table. The table formed after applying 2NF to the Emp_project table are emp_proj and emp table and proj table. Table 6.12 lists the various attributes in the emp_proj table.

Emp_id	Proj_no	Proj_hrs
1	P34	10
2	P45	04

Table 6.12: The emp_project

Table 6.13 lists the various attributes in the emp table

Emp_id	Emp_name
1	Ravi
2	Kishan

Table 6.13: **The emp table**

Table 6.14 lists the various attributes in the proj table

Proj_no	Proj_name	Proj_loc
P34	Sales	Delhi
P45	Account	Noida

Table 6.14: The proj table

"CHECK YOUR PROGRESS 1"

1. How do you apply 2NF to a relational table?
2. Define the term partial dependency.
3. How do you apply 2NF to a relational table?
4. Define inference rule.
5. What are the advantages of normalized relations over the un normalized relations?

6.10 TRANSITIVE DEPENDENCY

A transitive dependency occurs when a non-key column is uniquely identified by values in another non-key column of a table. A non-key column of a table refers to the column that is not identified as a key such as candidate key or primary key. For example consider a SUPPLIER table with attributes supplier_id, supplier_status and supplier_address. The functional dependencies that exist in the SUPPLIER table help to understand the concept of a transitive dependency.

Table 6.15 lists the various attributes in the SUPPLIER table.

supplier_id	supplier_status	supplier_address
S01	10	Mumbai
S02	20	Pune
S03	30	Lucknow

Table 6.15: the SUPPLIER table

In the following table, the following functional dependencies hold:

supplier_id → supplier_status

supplier_id → supplier_addree

supplier_address → supplier_status

in the SUPPLIER table, the non-key column supplier_status is identified by both the primary key supplier_id and non-key columns supplier_address. Therefore transitive dependency exists in the above table. To eliminate transitive dependency, you need to apply 3NF to the table.

6.11 THIRD NORMAL FORM

If a table satisfies the requirements of 2NF and the functional dependence of the non-key columns is only on the primary key, it is in

3NF. The third normal form is based on the concept of transitive dependency. A functional dependency A → B, in a relation, R is a transitive dependency if the following conditions are satisfied:

- A column or set of columns, C, exists in the table that is neither the candidate key of R nor the subset of any key of R.
- The functional dependencies A → C and C →B hold in the table.

For example consider a Subject table with attributes such as Subject_no and Chapter_name. Table 6.16 lists the various attributes in the Subject table.

Subject_no	Chapter_name	Instructor	Department
CS001	Entity relationship	Dillep Gupta	Computer
EC001	Digital Electronics	Ravi Kumar	Electronics

Table 6.16: The Subject table

In the above table, Subject_no is the only candidate key. Therefore the following functional dependency exists for the Subject table.

Subject_no → Chapter_name

Subject_no → Instructor

Instructor → Department

From the above functional dependencies you can say that Subject_no → Department and therefore the above table is in 2NF. However the table is not in 3NF since Department is not directly dependent on Subject_no. In the Subject table the Department column is determined by another non-key column, Instructor. Therefore to apply 3NF to the Subject table, you need to decompose the table into two tables subject_inst table and instructor table.

Table 6.17 lists the various attributes in the subject_inst table.

Subject_no	Subject_name	Instructor
CS001	Entity relationship	Dillep Gupta
EC001	Digital Electronics	Ravi Kumar

Table 6.17: The Subject_inst table

Table 6.18 lists the various attributes in the instructor table.

Instructor	Department
Dillep Gupta	Computer
Ravi Kumar	Electronics

Table 6.18: The instructor table

6.12 BOYCE-CODD NORMAL FORM

Boyce-Codd normal form is stricter than third normal form. In BCNF the relation which is in BCNF is also present in third normal form (3NF) but the relation in 3NF need not necessarily be present in BCNF. In 3NF anomalies can occur, if a relation has two or more candidate keys. In the situation of overlapping candidate keys, 3NF is unable to stop occurrence of anomalies. This provides a base for BCNF, which is based on the concept of determinant. A determinant refers to an attribute on which some other attribute is functionally dependent. The following code shows the relation and determinants:

R (a, b, c, d)
a, c→b,d
a, d→b

In the above code the first determinant states that you can change the primary key of relation R from a,b to a,c. After applying this change, you can still determine the non-key attributes present in relation R. The second determinant indicates that a,d determines b, but as a,d do not determine all the non-key attributes of R, it cannot be considered as the primary key of R. This implies that the first determinant is a candidate key but the second determinant is not a candidate key. Hence this relation is not in BCNF but is in 3NF.

To be in BCNF every determinant of the relation has to be a candidate key. The definition of BCNF specifies that a relation schema R is in BCNF if a non trivial functional dependency

X →A holds in R, then X is a super-key of R.

6.13 DECOMPOSITION

The relational database design algorithm starts with a single universal relation schema R= {A_1, A_2, A_3… A_n) which includes all the attributes of a database. The database designers specify the set, F of functional dependencies which holds true for all the attributes of R. This set, F of functional dependencies is also provided to the design algorithms. With the help of functional dependencies, these algorithms decompose the universal relation schema, R into a set of relation schemas, D= {R_1, R_2… R_m) which turns out to be the relational database schema. In this case D is referred as a decomposition of R. The properties of decomposition are:

- **Attribute preservation:** It involves preserving all the attributes of the relation, which is being decomposed by the design algorithms. While decomposing a relation, you need to make sure that each

attribute in R exists in at least one relation schema R_i while decomposition the relation.

- **Lossless-join decomposition:** It ensures that the join remains in the same relation, as it was before the decomposition of the relation. The decomposition of the relation R into several relations R_1, R_2… R_n is called lossless-join decomposition. If the relation R is the natural join of the relation R_1, R_2… R_n. to test whether a given decomposition is a lossless-join for a given set F of functional dependencies, you need to decompose the relation R into R_1 and R_2. If the decomposition of the relation R is lossless-join then one of the following conditions has to be true:

  (R_1 intersection R_2) →($R_1 - R_2$)

  (R_2 intersection R_1) →($R_2 - R_1$)

- **Dependency preservation:** It states that of each functional dependency X → Y, specified in F, either directly appears in one of the relation schemas R_i in the decomposition D or is inferred from the dependencies that appear in the relation R_i . the need of dependency preservation arises because each dependency in F represents a constraint on the database. When decomposition does not preserve the dependency, then some dependency can be lost in decomposition. You can check for a lost dependency by creating a join of two or more relations in decomposition to get a relation, which includes all the left and right-hand side attributes of the lost dependency. Then, check whether or not the dependency is preserved on the result of join.

Example:

Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

Functional dependencies in the table above:
emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate key: {emp_id, emp_dept}

The table is not in BCNF as neither emp_id nor emp_dept alone are keys.

To make the table comply with BCNF we can break the table in three tables like this:

emp_nationality table:

emp_id	emp_nationality
1001	Austrian
1002	American

emp_dept table:

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

emp_dept_mapping table:

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

Functional dependencies:

emp_id -> emp_nationality
emp_dept -> {dept_type, dept_no_of_emp}

Candidate keys:

For first table: emp_id
For second table: emp_dept
For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.

6.14 MULTI-VALUED DEPENDENCY

An entity in E-R model can have multi-value attributes. A multi-value attribute is one that does not have a single value but a collection of values. If you have to store such an entity in one relation, you should repeat all the information excepting the multi-value-attribute value. In this way the same instance of the entity will have many tuples. The situation becomes much inferior if the entity's multivalued attributes exceed one. The multi-valued dependency (MVD) gives a solution to the problem of more than one multi-valued attributes.

MVD: Let R(X, Y, Z) be a relation. The multi-valued dependency X→→ Y is said to hold for relation R if for a given set of value for attribute X, there is a set of zero or more associated values for the set of attribute Y. The Y values depend only on X values and have no dependence on the set of attributes Z.

Suppose a Students table, which has Stud_name, Qualifications and Languages as attributes. In the relation a student can have more than one qualification (Stud_id →→ Qualifications) and know more than one language (Stud_id→→ Languages). This relation shows duplication of data inconsistency. You can decompose the Students relation into two relations having attributes Stud_id, Qualifications and Stud_id, Languages.

In this example if there is dependency between Qualifications and Languages attributes then Student relation would not have MVD and cannot be decomposed into two relations.

6.15 FOURTH NORMAL FORM

The fourth and fifth normal forms depend on multivalued facts and composite keys. The multivalued facts correspond to many-to-many relationship and many-to-one relationship. In the fourth and fifth normal forms, you need to minimize the number of columns in a composite key.

In the fourth normal form, you must ensure that a record satisfies the third normal from and does not contain two or more independent multivalued facts about an entity. For example, consider a table R with attributes employees, skills and languages. Since in the table, an employee may have various skills and may know various languages, therefore the table has two many-to-many relationships. Under fourth normal form, the two many-to-many relationships are not represented in a single row and you need to split R into two tables. Therefore, the table R is split into a table with attributes employees and skill and another table with attributes employees and language.

If each table contains only one multi-valued dependency per key attribute, the relation is in fourth normal form.

For example, consider the following instructor table as shown in Table 6.19 which lists the various attributes of instructor table

MID	Database	Instructor
1	Access	Manoj
8	Access	Ravi
1	Access	Sachin
1	DB2	Manoj
1	DB2	Sachin
8	Oracle	Ravi

Table 6.19: The Fourth Normal Form of Instructor Table

The redundancy of data is obvious: there are multiple values of Instructors and Database for each MID. This illustrates a multi-valued dependency. Table 6.20 and 6.21 shows the fourth normal form of instructor table.

MID	Database
1	Access
8	Access
1	DB2
8	Oracle

Table 6.20: MID_DATA Table

MID	Instructor
1	Manoj
1	Sachin
8	Ravi

Table 6.20: MID_instructor Table

6.16 JOIN DEPENDENCY AND FIFTH NORMAL FORM

The fifth normal form (5NF) is based on join dependency. Join dependency implies that after a table is decomposed into three or more tables, the tables can be joined to from the original table. A table is in 5NF if the table cannot have lossless decomposition into smaller tables. A lossless decomposition implies that after a relational table is decomposed into smaller tables. The joining of the tables results in exactly the same relational table which was decomposed. If an instance is added to a table, which is not in 5NF, it results in spurious results when the tables are decomposed and then rejoined.
Table 6.21 lists the various attributes of instructor –MID-Location table.

Instructor	MID	Location
Smith	1	New York
Smith	2	Chicago
Jones	1	Chicago

Table 6.21: The Instructor-MID-Location Table

If you were to add the MID-2 to New York, you would be faced with adding a line to the table for each instructor located in New York. If Jones were certified for MID-2 and could travel to New York, you would have to add two lines to reflect this.

Table 6.22 shows the instructor-MID-Location table and its decomposition into fifth normal form.

Instructor-Seminar Table

Instructor	MID
Smith	1
Smith	2
Smith	1

Seminar-Location Table

MID	Location
1	New York
1	Chicago
2	Chicago

Instructor-Location Table

Instructor	Location
Smith	New York
Smith	Chicago
Jones	Chicago

Table 6.22: The fifth normal form of instructor-MID-Location Table

6.17 FUNCTIONAL DEPENDENCY

Functional dependency is represented by X→Y between two attributes, X and Y in a table. The functional dependency X→Y implies that the Y is functionally dependent on X. For example, consider the following EMPLOYEE table:

Table 6.23 lists the various attributes in the EMPLOYEE table.

Employee_id	Employee_name	Employee_dept
K067263	John	Sales
K067264	Chris	Accounts
K067265	Ken	sales

Table 6.23: The EMPLOYEE table

In table 6.23, the various attributes of the EMPLOYEE are Employee_id, Employee_name and Employee_dept. You can state that:

Employee_id →Employee_name

The above representation states that the Employee_name attribute is functionally dependent on the Employee_id which implies that the name of an employee can be uniquely identified from id of the employee. However you cannot uniquely identify the Employee_id from the Employee_name column because more than one employee can have the same name. However each employee has different value in the Employee_id column.

Functional dependencies are a type of constraints based on keys, for a relation table R, a column Y is said to be functionally dependent on a

column X of the same table if each value of the column X is associated with only one value of the column Y at a given time. All the columns in the relational table R should be functionally dependent on X if column X is a primary key.

If the columns X and Y are functionally dependent, the functional dependency can be expressed as:

$$R.x \rightarrow R.y$$

For example consider the following functional dependency in a table.

$$\text{Employee} \rightarrow \text{Salary}$$

The column 'Employee' functionally determines the 'Salary' column because the salary of each employee is unique and remains same for an employee, each time the name of the employee appears in the table.

A functional dependency represented by X → Y between two sets of attributes X and Y are the subsets of R, is called trivial functional dependency if Y is a subset of X.

For example, Employee Project → Project is a trivial functional dependency.

A functional dependency represented by X → Y between two sets of attributes X and Y which are subsets of R, is called a non-trivial functional dependency if at least on of the attributes of Y is not among the attributes of X.

For example Employee → Salary is a non-trivial dependency.

Full functional dependencies are applied to the tables, which have composite keys. Full functional dependency implies that when a primary key in a table is made up of two or more columns, the other columns are identified by the entire key.

"CHECK YOUR PROGRESS 2"

1. What is a primary key in a relational table?
2. Explain the concept of functional dependency with an example
   Explain join dependency and fifth normal form.
3. What is transitive dependency?
4. Describe the term MVD in the context of relational database management system by giving an example.
5. What is dependency Preservation property for decomposition? Explain why it is important.

6.18 SUMMERY

- Normalization is the technique to organize the contents of the table for transactional databases and data warehouses. Normalization eliminates two factors: redundancy and in consistent dependency.
- An attribute in a relational table is said to be functionality dependent on another attribute in the table if it can take one value for a given value of the attribute upon which it is functionally dependent.
- The reflexivity rule, Augmentation rule and the Transitivity rule are known as "RAT Axioms" or "Armstrong Axioms" in honour of the person who was first proposed it.
- The various types of normal forms are: First normal form, Second normal form, Third normal form, BCNF form, Fourth normal form, Fifth normal form.
- Multi-valued dependency occurs when two or more independent multi-valued facts about the same attribute occur within the same relation. More generally, it is denoted by

$$X \twoheadrightarrow Y$$

6.19 Objective TYPE Questions

Q1: Every time attribute A appears, it is matched with the same value of attribute B, but not the same value of attribute C. Therefore, it is true that:

(a) $A \rightarrow B$ (b) $A \rightarrow C$

(c) $A \rightarrow (B, C)$ (d) $(B, C) \rightarrow A$

ANS: A

Q2: The different classes of relations created by the technique for preventing modification anomalies are called:

(a) Normal forms. (b) Referential integrity constraints

(c) Functional dependencies. (d) None

ANS: A

Q3: A relation is in this form if it is in BCNF and has no multi-valued dependencies:

(a) Second normal form. (b) third normal form

(b) fourth normal form (d) domain/key normal form

ANS: C

Q4: Row is synonymous with the term:

(a) Record (b) relation

(c) column (d) field

Ans: A

Q5: The primary key is selected from the:

(a) Composite keys. (b) Determinants

(c) Candidate keys (d) foreign keys

Ans C

Q6: Which of the following is a group of one or more attributes that uniquely identifies a row?

(a) Key (b) Determinant

(c) Tuple (d) relation

Ans: A

Q7: When the values in one or more attributes being used as a foreign key must exist in another set of one or more attributes in another table, we have created a (n):

(a) Transitive dependency (b) insertion anomaly

(c) Referential integrity constraint (d) normal form

Ans:C

Q8: A relation is considered a:

(a) Column. (b) 1 d table

(c) 2 d table (d) 3 d table

Ans: C

Q9: In the relational model, relationships between relations or tables are created by using:

(a) Composite keys (b) determinants

(c) candidate keys (d) foreign keys

Ans: D

Q10: A functional dependency is a relationship between or among:

(a) Tables (b) rows

(c) Relations (d) attributes

Ans: D

6.20 SELECTED EXERCISE

1. Define the following Models:

(i) Conceptual Data Models

(ii) Logical Data Models

(iii) Physical Data Models

2. What is the Normalization?

3. Discuss kinds of normalization with example.

4. Define 1NF with example

5. Define 2NF and 3NF with example

6. What is BCNF? Explain with example?

7. Differentiate between 4NF and MVD.

8. Differentiate between 5NF and Join dependency.

9. Explain Following:

(i) Inclusion Dependency

(ii) Boyce-Codd Normal Form

10. What is join dependency? Explain with example?

11. What do you mean by decomposition of a relation?

12. Discuss on the various ways in which we can arrive at a good database design.

13. Define BCNF. How does BCNF differ from 3NF? Explain with an example.

BLOCK

3

Curriculum Design Committee

Dr. P. P. Dubey, **Coordinator**
Director, School of Agri. Sciences, UPRTOU, Allahabad
Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Allahabad
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg., MNNIT, Allahabad
Prof. P. K. Mishra **Member**
Dept. of Computer Science, Baranas Hindu University, Varanasi
Mr. Prateek Kesrwani **Member Secretary**
Academic Consultant-Computer Science
School of Science, UPRTOU, Allahabad

Course Design Committee

Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Prayagraj
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg., MNNIT, Allahabad, Prayagraj
Prof. P. K. Mishra **Member**
Dept. of Computer Science, Baranas Hindu University, Varanasi
Faculty Members, School of Sciences
Dr. Ashutosh Gupta, Director, Scool of Science, UPRTOU, Prayagraj
Dr. Shruti, Asst. Prof., (Statistics), Scool of Science, UPRTOU, Prayagraj
Ms. Marisha Asst. Prof., (Computer Science), Scool of Science, UPRTOU, Prayagraj
Mr. Manoj K Balwant Asst. Prof., (Computer Science), Scool of Science, UPRTOU, Prayagraj
Dr. Dinesh K Gupta Academic Consultant (Chemistry), Scool of Science, UPRTOU, Prayagraj

Course Preparation Committee

Mr. Sanjeev Gangwar* **Author**
Dept. of Computer Applications
VBS Purvanchal University, Jaunpur
Ms. Marisha[#] **Author**
Assistant Professor- Computer Science
School of Science, UPRTOU, Prayagraj
Dr. Ashutosh Gupta **Editor**
Director, School of Sciences,
UPRTOU, Prayagraj
Prof. U. N. Tiwari **Member**
Dept. of Computer Science and Engg.,
Indian Inst. Of Information Science and Tech., Prayagraj
Prof. R.S. Yadav, **Member**
Dept. of Computer Science and Engg.,
MNNIT, Allahabad, Prayagraj
Prof. P. K. Mishra **Member**
Dept. of Computer Science
Baranas Hindu University, Varanasi
Dr. Dinesh K Gupta, **SLM Coordinator**
Academic Consultant- Chemistry School of Science, UPRTOU, Prayagraj

Note: Author's * -Block 1 and 2, # - Block-3

ISBN : 978-93-83328-29-1

UNIT -7: FILE ORGANIZATION

Introduction, file organization, sequential file organization, index-sequential file organization, direct file organization, multi key file organization.

UNIT -8: TRANSACTION PROCESSING CONCEPT

Transaction System, testing of serializability, serializability of schedules, conflict and view serializable schedule, recoverability, recovery from transaction failure, log based recovery, checkpoints, deadlock handling, concept of concurrency.

BLOCK INTRODUCTION

Storage of database and its efficient access when needed are important aspects of any practical database system. In the first unit of this block, we shall primarily discuss the techniques of storing databases permanently over physical storage devices and the techniques used for accessing it later. In particular, we shall discuss the physical placement of database records on disk, various types of file organizations such as heap file organization, sequential, direct, indexed and multi-key file organizations. We shall also learn the advantages and disadvantages of each of them. In the next unit, we shall discuss transaction processing concepts. We will start with the concept of concurrency and why it is needed. Then, we will discuss transactions, its definition, the states of transaction, their properties and then we will define the concept of schedules. We will discuss about serializability of schedules and serializability test. Thereafter, we will move on to discuss the various types of failures that might occur in a database system and their remedies. In the end we will briefly discuss the concept of deadlock and how the database system handles deadlock. After learning this block, you would have gained most of the knowledge required for working with real database systems and be able to design and manage small database applications.

UNIT-7

FILE ORGANIZATION

Structure

7.1 INTRODUCTION

So far we have studied the relational database model, database integrity and normalization techniques. For a typical database system, the number of tables and the records in each table is very large. This large amount of data is to be stored for future use. In most cases, the databases are stored for a long period of time and the information in the databases are repeatedly accessed and modified during this period. Most databases are permanently stored on magnetic disks (secondary storage) mainly because:

1. Databases are generally very large and cannot fit entirely in the main memory
2. Primary storage is volatile and hence its contents will be lost if the power is turned off.
3. The cost of storage is much less in case of disk storage as compared to primary storage.

There are various ways of storing large amounts of data on the disk each having its own advantage and disadvantage. Physical database design involves choosing a particular database design technique out of several available techniques, that best suits the given application requirement. **File organization** is mainly concerned with the organization of data into records, physical placement of records on the disk and access structures for retrieval of the records whenever needed.

In most database applications, only a part of the database is required at any point of time for processing. Whenever a portion of the database is needed, it has to be located on the disk, copied to the primary memory for processing, and written back to the disk if any changes are made in the data. The records should be stored on the disk in such a manner as to make locating and retrieving the records efficient. It is therefore, necessary for database designers, administrators and implementers to know various file organization techniques, their implementation, their advantages and dis-advantages for design, implementation and operation of Database Management Systems.

In this unit, we will discuss file organization in database systems. We will discuss how different types of records are placed physically on the disk. We will also discuss various access structures which can be defined on different types of file organizations for providing faster retrieval of records. These provide alternative ways of accessing records in addition to the one based on physical placement of records.

OBJECTIVE

After learning this unit you should be able to:

1. Describe how databases are stored physically on the disk
2. Describe the various types of file organizations and discuss their advantages and disadvantages
3. Differentiate between primary and secondary file organizations (auxiliary access structures)
4. Define multi-key file organization

7.2 PLACING FILE RECORDS ON DISK

The magnetic disk is a random access storage device. The smallest unit of data stored on a disk is a **bit**. Data is stored on the disk into concentric circles known as **tracks**. The number of tracks range from a few hundred to a few thousand and each track can store tens to 150 Kbytes. The track is further divided into smaller units known as **sectors**. Though most disks have their tracks divided into sectors, it is not necessary. The division of tracks into sectors is hard-coded on the disk surface and cannot be changed.

The tracks are divided into equal sized disk blocks (or pages). The size of disk blocks is set by the operating system during formatting (or initialization) and cannot be changed dynamically. Data is transferred from disk to the main memory in units of disk blocks. Blocks sizes typically range from 512 to 8192 bytes. As we have just discussed, the database is stored as files on a disk. A file is a logical sequence of records. The records of the file are mapped onto the disk blocks. Although the size of the disk blocks is fixed and cannot be changed dynamically, the size of the records may vary. Typically, in a relational database, the tuples of different relations are of different sizes. We employ different methods for storing different types of records. Based on whether the size of each record of a file is same or not, the files in a database system can be of two types:

1. Fixed Length Records and
2. Variable Length Records

If different records in a file are of fixed length then the file is said to be of fixed length otherwise it is said to be of variable length. Figure 7.1 below gives an example of fixed length records. Here, every record has same fields and the length of each field is fixed. The system can identify the starting position of each field relative to the starting position of the record.

Name	Employee ID	Salary	Hire Date	Department
Ram	1234	xxx	xxx	Computer

Figure 7.1: Records of fixed length

It is possible to have variable lengths of records in a file if different fields of a record are of variable lengths or some records can have repeating or missing values. Nevertheless, we can store records of variable lengths as fixed length records if we know the maximum possible length of the variable length fields. We can then reserve this maximum amount of space for every record. This method however can cause a large amount of space to be wasted if the number of records is large. Storage and access is much easier for fixed length records than for variable length records. This is because if the length of each record and the length of each field of the

record is fixed, the system can access each record by calculating the fixed length offset from the starting position of the record.

For variable length records, where fields of a record can be of varying length, some special character is used as separator (the separator should not appear in any of the field values) for the field values. For example, if in the above case, we allow name of the employee to be of variable length then we would require a separator character to mark the end of the field name. This is shown in the Figure 7.2 below:

Name	Employee ID	Salary	Hire Date	Department
Ram	▮ 1234	xxx	xxx	Computer

▮: Separator Character

Figure 7.2: Records of Variable length

For optional fields the records can be stored as a sequence of name-value pairs as <field-name, field-value> or we may use some integer coding for each field and store the fields as <field-code, field-value>. In such cases, we need separator characters to separate the field name from the field value and for separating one field from another field. A separator character is also needed to mark the end of a record.

A repeating field in a record is a field that can have multiple values. For storing repeating fields in a variable length record, we need a separator character for separating the multiple values of the repeating field and another one to mark the end of the field. Lastly, a separator character will be required to mark the end of the record. The processing of files of variable lengths is usually a part of the file system and is therefore not of concern for programmers. In other words, the programmer need not worry about whether the file contains fixed length records or variable length records and she can work with the variable length records file in the same manner as with the fixed length records file. Understandably, therefore the storage and processing of variable length records files is more complex than that of fixed length records where the starting position of each field are known and fixed.

Check Your Progress

Q. Discuss how optional fields and repeating fields of a variable length record are stored.

7.2.1 Allocation of Files on Disk Blocks

There are different ways in which files can be allocated on disk blocks. Several ways of allocating disk blocks are: **Contiguous Allocation, Linked Allocation and Indexed allocation**.

In **contiguous allocation**, the records of files are stored contiguously one after the other on the disk blocks. This makes reading the file easier.

However, insertion and deletion of records becomes difficult as it would involve shifting of records. Thus, while reading of file is easier in contiguous allocation, the expansion of file becomes difficult.

In **linked allocation**, file can be stored on several non-contiguous disk blocks and each disk block contains a pointer to the next block. This makes it easy to expand the file; however reading the files takes longer time than for contiguous allocation. In some cases, the file is stored on clusters of disk blocks and the clusters are linked.

The third type of storage used for allocation of files is **indexed allocation**. Here, one or two blocks act as index blocks. The index blocks store the address of the disk blocks that actually store the data. The index can be single level or multi level depending on the amount of data to be stored. In multi level indexing structures, there are more than one index blocks and one index block contains the pointer to other index blocks which in turn contains the pointer to actual disk blocks. It is also common to use a combination of the three techniques.

7.2.2 File Headers

A file header also known as a file descriptor contains information about a file that is required by the system to access the file. It contains the address of the disk blocks that store the file and also the record format description of the file. The record format description includes field length and order of fields for fixed length records and field type codes, separator characters and record type codes for variable length records. For searching a particular record in a file, one or more disk blocks are copied to the buffers in the main memory. The desired records are then searched in the buffers.

If the address of the disk block that contains the record is unknown then we must do a linear search over all the blocks until the desired record is found or the search ends unsuccessfully after searching all the blocks. Each block is copied into the main memory and searched for the desired record using the information in the file header. This can be very time consuming if the size of the file is very large. Therefore, the goal of a good file organization is to locate the desired record within a minimal number of block transfers.

Organization of Records in a File

There are a number of operations that can be performed on a file. Typical file operations include scan, insert, delete, find or locate, modify etc. Depending on the structure and purpose of the database system, there can be operations that we expect to perform more frequently on a particular file rather than others. Some files can be more static in nature i.e. update operations may be very rare on them while others can be dynamic meaning they are updated more frequently than others. A successful file organization is one which can execute as efficiently as possible those operations which we expect to perform more frequently on a file. For example, if a user wants to search the details of an employee

based on employee ID then the file organization should be such that it facilitates finding a record based on its Employee ID value.

Different applications on the database may require different arrangement of the records of the data. This might give rise to conflicts in terms of the most efficient arrangement of the records. In many cases, there exists no such organization that allows all the operations to be done efficiently. In such cases, the designer must choose a compromise between different operations by taking into account the relative importance and mix of different operations.

File organization can be broadly divided into two different types. Primary file organization and Secondary file organization. **Primary file organization** describes how the records are actually placed on the disk and therefore, how they can be accessed. Primary organization of files can be of different types. A **heap** file places the records randomly on the disk following no particular ordering. **Sequential** file places the records sequentially one after the other based on the value of some particular data field (attribute). In a **Hashed** file, a hash function determines the position of a record on the disk based on a particular data field. Another primary file organization is B-tree which uses a tree structure to store the data. **Secondary file organization or auxiliary access structure** describes a number of access structures on the files either based on the order in which records are placed or independently. It makes efficient access of the records possible on fields other than the primary field. We will discuss both Primary and Secondary file organization techniques here.

Check Your Progress

Q. Discuss the goals of a successful file organization.

7.3 HEAP FILE ORGANIZATION

This is the simplest and most basic type of file organization. This is applied to files of unordered records. Here, the records are placed in no particular order and new records are inserted at the end of the file. Therefore, it is known as a heap file organization. In this type of file organization, insertion of new records is very efficient as the new records can simply be place after the last record. Searching a record in the file would however amount to linearly searching the record block by block which is quite expensive. If a file has n blocks, then on an average $n/2$ blocks need to be searched to find a particular record. If a record is not present or if there are multiple possible matches then we must do a linear search over all the n blocks.

Deletion of a particular record requires finding the block containing the record to be deleted, copying the disk block into the memory buffer, deleting the record and writing back the disk block. Another approach

could be to use a deletion marker with every record. A record will be considered as deleted if the deletion marker is set to a particular value, a different value of the marker will indicate a valid (not deleted) record. Both of these techniques require **reorganization** of the file to reclaim the unused space of the deleted records. During reorganization, all the records are accessed sequentially and are clubbed together by removing the unused space. Another possibility is to use the space of deleted records to insert new records. This however requires keeping track of the empty locations.

Advantages:

1. Simplest type of file organization
2. Insertion is very efficient

Disadvantages:

1. Retrieval is inefficient and requires a linear search
2. Deletion results into unused space
3. Requires frequent reorganization of files if delete operations are frequent

7.4 SEQUENTIAL FILE ORGANIZATION

In sequential file organization the records are placed sequentially one after the other based on the values of one of the fields known as the **ordering field**. If the ordering field is same as the key field of the file then the ordering field is known as the ordering key. Figure 7.3 shows an example of an ordered sequential file based on the employee ID. Sequential file organization allows records to be read in sorted order which can be used for display purposes and in certain types of query processing algorithms.

Employee ID	Name	Salary	Hire Date	Department
12301	Ram	xxx	xxx	Computer
12302	Aman	xxx	xxx	Mathematics
12303	Akash	xxx	xxx	Humanities
12304	Srishti	xxx	xxx	Arts

Figure 7.3: An Ordered sequential file

In sequential file organization reading the records and searching for a particular record is very efficient if it is based on the ordering key field. For finding the next record no additional disk block access is required until the current record is not at the end of a block. However, the sequential file organization provides no improvement over heap file organization if the searching over the records is based on some field other than the ordering field.

Insertion and deletion are expensive operations to perform on an ordered file as the sequence of the records must be maintained after

insertion and deletion. For insertion we first need to find the correct location of the record to be inserted based on the ordering field and then space must be created to insert the record at that location by shifting the following records. This process is very time consuming if the file size is large. On an average half the number of disk blocks that contain the file will have to be re-written. Similarly for deletion, all the records following the record to be deleted have to be shifted up. This process however can be made more efficient by using the deletion marker and periodic reorganization of file.

The process of insertion in a sequential file organization can also be made less severe by using some techniques. Maintaining some unused space in each block is one possibility where new records can be inserted in this space. In this case, the shifting of records will be limited by the size of the block. However, once all the space gets used up, the original problem would appear again. Another technique which is commonly used is to keep an overflow file. The **overflow file** is an unordered file where the new records can be inserted at the end of the file. The original file is known as the main file or the master file. In such a system, the overflow file is periodically sorted and merged with the main file. Although this technique reduces the time and overhead of insertion, it makes searching more difficult. The searching algorithm needs to perform a linear search over the overflow file if, the search item is not found in the master file. In applications where most up-to-date information is not required, the overflow file is ignored while searching. Ordered files are much less frequently used in common database applications. They are more common when an additional access path also known as the primary-index is used. This type of organization is known as index-sequential file organization. We will discuss this type of file-organization in detail in subsequent sections.

It is possible that the ordering field is not the key field for the file. In this case, the file organization is known as **clustering** file organization and the file is known as **clustered file**. In clustering file organization, as the ordering field is not the key field there can be multiple records for each value of the ordering field. In this case, all the records that match a particular value of the ordering field are stored sequentially followed by the records for next higher value of the ordering field.

Advantages:

1. Searching of records is very efficient (ignoring overflow file)
2. It is suitable for batch systems (large volume of data that need to be processed periodically)

Disadvantages:

1. Requires frequent reorganization of file. File operations such as insertion, modification and deletion all require reorganization of records.

2. The method is too slow to handle most of the real life applications and hence it is rarely used in database applications.

7.5 DIRECT FILE ORGANIZATION

Direct file organization implies providing direct access to the records of the data file. For providing direct access to the records the concept of hashing is used. Therefore, this type of file organization is also known as hashed file organization. It provides very fast access to the records. In hashed file organization, a hash function is defined which takes the value of a particular field called hash field as input and provides the disk block containing the particular record as the output. Using hashed file organization a record can be accessed directly for equality condition on the hash field. If the hash field is also a key field then it is known as hash key. The search for the particular record within the block retrieved using the hash function can be performed internally in the main memory buffer which is much faster than disk search. Hashing can be of two types Internal hashing and External Hashing. While internal hashing is used for internal files, external hashing is used for disk files.

7.5.1 Internal Hashing

Internal hashing is used for internal files and is typically implemented using a **hash table**. The address of a particular record or the location where a record is to be stored is determined using the hash function. The **hash function** performs the key to address translation, also known as **mapping**. Suppose we have an array of records and *M* address spaces that correspond to it as shown in Figure 7.4. In this case, we have to choose a hashing function such that it takes the hash field value as input and gives an integer value between *0* and *M-1*. There are many different hashing functions which can be used to this effect. One common hash function is h(*K*) = *K* Mod *M*. This function returns the remainder of the hash field value K after division by the total address space M. This value will always lie between 0 and M-1 and hence it is a valid hash function. The value returned by this function is then used for storing the record. Other commonly used techniques for generating hash functions are mid square method, folding, algebraic coding, digit analysis etc.

A situation that commonly arises with hashing is known as **collision**. Collision occurs if the address, to which the hash field of a record being inserted is mapped, already contains another record. In this situation, the new record must be inserted at some other location. The technique by which we find the new location where the colliding record will be inserted is known as **collision resolution**. Some of the methods commonly used for collision resolution are open addressing, chaining and multiple hashing. Each collision resolution has its own algorithm for insertion, deletion and retrieval of records. This increases the complexity of the file organization method. A good hashing function should be such that it distributes the records uniformly over the address space so as to minimize collision while not leaving many unused locations.

Advantages:

1. It provides the fastest possible way of retrieving record given the value of the hash field.
2. Insertion of records is very fast.

Disadvantages:

1. Search condition must be an equality condition on the hash field.
2. It is a complex file organization method.
3. Performance of the hashing technique depends on the choice of hash function.
4. Hashing function needs to be carefully chosen so as to ensure that not only the collisions are minimal but also there isn't too much unused space.

7.6 INDEXED FILE ORGANIZATION

So far we have discussed various types of primary file organization techniques. As discussed earlier primary file organization refers to the way files are stored physically on the disk. We can also define additional access structures on the file. Access structures also known as indexes can provide speedy retrieval for certain search conditions. The access structures provide alternative ways of accessing the records irrespective of their physical placement on the disk, based on the indexing field. Indexing field is the field on which the access structures are defined. Any field in the data file can be used as an indexing field and thus multiple indexes can be defined on the same file. To find a record using the index structure, the index is accessed first, which points to the disk block(s) where the record is located.

Indexes

The index used in a database storage application can be understood as similar to the one which is used in a textbook. The index which is given at the end of the textbook lists the important terms used in the book in the alphabetical order. We can search the index of a book to find out the page number corresponding to a given word which gives us the location where the term is used in the textbook. This helps in locating the word easily in the text. Indexes can be categorized based on different criteria.

Indexes can be of many different types. We will discuss here the two most prevalent types of indexes namely:

1. Single-level indexes and
2. Multilevel indexes

Single-level Indexes

For a database application, a file contains a number of records with certain record structure consisting of a number of fields. Usually an index access structure is on a single field of the file. This field is also called as the **indexing field** or indexing attribute. The index then stores two values namely, the value of the index field and a pointer to all the disk blocks that contain the records corresponding to the value of the index field. The file that stores the indexes for the database file is called the **index file**. The values of the index are ordered and therefore, a binary search can be performed on the index file for efficient retrieval. We will discuss here, only about ordered indexes. The ordered indexes can be of several types:

1. **Primary Index**: A primary index is defined on the ordering key field of the ordered file. It is an ordered file with two fields. One of the field is same as the primary key and the other field contains the address of the block that contains record with that primary key value. The index file has that many entries as the number of disk blocks in the data file. Thus, it is a sparse index as the entries are not maintained for each value of the index key field.

2. **Clustering Index**: If the physical ordering of the records of the file is not on a key field then another type of indexing called clustering index can be used. A clustering index again is an ordered file consisting of two fields. One of the fields is same as the clustering index while the other field is a block pointer pointing to the address of the block containing record with that value of the clustering field. The clustering index has one entry for each distinct value of the clustering field and the corresponding block pointer contains the block address of the first block of the data file that contains a record with same field value. As you know that a file can have at most one physical ordering, one can at most define one primary index or one clustering index on the file but not both.

3. **Secondary Index**: A secondary index is the one which is defined on a non-ordering field of a file. It provides an additional way of accessing a file for which a primary index is already defined. A file can have multiple secondary indexes in addition to its primary access method.

Multilevel Indexes

Multilevel indexing scheme uses multiple files for storing the indexing information of a file. The indexing schemes discussed thus far have only one ordered index file. The problem with keeping only one index file is that the searching over the index file itself becomes enough time consuming if the file size increases beyond certain limit. Typically for large database files, the index file itself can be very large extending over multiple disk blocks and the binary search over the disk blocks for searching a particular index entry would approximately require $\log_2 bi$ time for an index with *bi* blocks.

The idea behind multi-level indexes is to create an index over the index file which can be used to store the pointers to the disk blocks containing the index entries. The original index file is said to be the **first file** which is the first level and the index to the first file is said to be the second level index. Using this second index file the searching time can be reduced. This process can be repeated to the next level so as to create **third level** index and so on.

Dense and Sparse Indexing

In dense indexing, the index value is stored for every value of the search key while in sparse indexing an index entry is stored for only some of the search key values. E.g. primary indexing is a sparse indexing scheme as the index entries are stored for every block of the data file and not every value of the search key.

7.7 INDEX-SEQUENTIAL FILE ORGANIZATION

This is the type of file organization in which the primary index is used for indexing the file. Since the primary index is defined on the ordering key field, the records are physically stored on the disk in the order of the indexing field and every record has a unique value for this field. A primary index itself is an ordered file of fixed length with two fields. The first field is the primary key of the data field and the second is a pointer to the disk block. There is one entry for each disk block in the index file. Each index entry has the value of the primary key field of the first record of a disk block and a pointer to the block. The structure of a typical entry in an index file with Name as the primary key (assuming that no two records have the same Name field value) is illustrated in Figure 7.6.

Key	Address
Akankasha	Address of block 1
Chandrika	Address of block 2
Nilesh	Address of block 3
Renu	Address of block 4
Santosh	Address of block 5

Figure 7.6: Structure of a typical index file

As discussed earlier, this is a sparse indexing scheme since the index entries are not maintained for every record of the data file also the index entries are small with respect to the size of a record (has only two fields), the index file needs substantially fewer blocks than the data file. Thus, a binary search over index blocks requires fewer block accesses than a binary search on the data file. Further, in order to retrieve a record when the value of its primary key is given we simply perform a binary search on the index file so as to find the appropriate index entry and then retrieve the corresponding data block. E.g. for an index value K such that K(i) < K <

K(i+1) where *i* is the *i*th entry in the index file and P(i) is the address of the corresponding data block, we first find the index entry *i* and then locate the data file block with address P(i) to find the desired record. An example of the index file is shown in figure 7.7.

Advantages:

1. It provides faster access to the records.
2. Indexes can be defined as per requirement on any field irrespective of the physical ordering of the records.

Disadvantage:

1. Creating and maintaining index files is an overhead on the Database Management System.

Check Your Progress

Q1. Why does the index file for a primary index need comparatively much less number of blocks for storage than the actual data blocks?

Q2. Why is it possible to have several secondary indexes on a file but only one primary or clustering index?

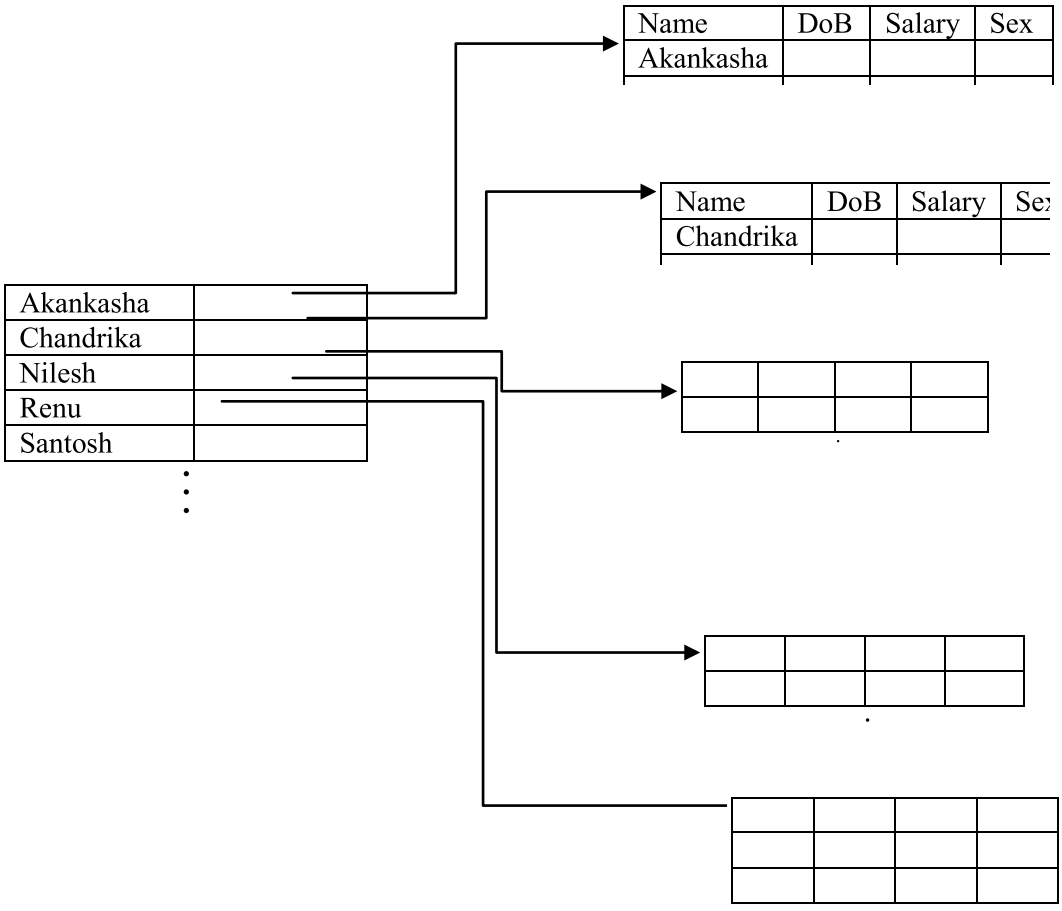


Figure 7.7: example of an index file with primary index

7.8 MULTI-KEY FILE ORGANIZATION

So far we have seen indexed file organizations based on primary or secondary keys. In all these types of file organizations the records were accessed based on only single field of the data file. However, in practice there might be situations where accessing records using multiple keys can be easier and faster than those based on single key. For example, let us consider an Employee file as shown in Figure 7.8 below. Here, the EID is defined as the primary key of the file. Now, suppose we want to retrieve the details of employee with DeptNo = 3 and Age = 35. Since both of them are non-key attributes, there can be multiple records corresponding to each value of the keys. In this case, the records can be accessed in multiple ways:

1. Firstly, if any one of the two i.e. Age and DeptNo are defined as index field, we can search the records based on that key and using it select the records that satisfy the search condition for the other field.
2. If an index is defined for both of the fields then we may use both the indexes to retrieve the corresponding addresses and then take the intersection of them to get the records that satisfy both the conditions.

EID	Name	Age	DeptNo	Salary
51	Jain	50	1	50,000
25	Rishi	40	8	45,000
64	Hari	30	5	30,000
37	Meena	35	3	40,000

Figure 7.8: Employee table

None of the above methods would be efficient if, the records that satisfy one of the search conditions are very large and the intersection is small. In such cases, we may think of an index that can take the combination of <DeptNo, Age> as the search query. A number of possibilities exist for doing this. The type of file organization where we use a combination of keys as index is known as Multi-key file organization. In this type of file organization, an index is created on a set of multiple attributes <A1, A2, A3…An>. The search key values are also defined as n valued tuples <v1, v2, v3…vn>. The composite search key is then determined by a lexicographic ordering of these tuple values. We discuss here some of the techniques commonly used in multi-key file organization.

7.8.1 Partitioned Hashing

It is an extension of static external hashing that allows access on multiple keys. It can be used only for equality queries. Here a key consists of n components; the hash function produces the results with n separate hash addresses. The bucket address is a concatenation of these n addresses. The required composite search key can then be

searched by looking up the appropriate buckets that match the parts of the address in which we are interested.

Advantage:

1. It can be easily extended to any number of attributes.

Disadvantage:

1. It cannot handle range queries on any of the component attributes.

7.8.2 Grid Files

In this scheme, a grid array is constructed with one linear scale for each search attribute. The scales are made in such a way that a uniform distribution is achieved over the attributes. Each cell points to some bucket address where the records corresponding to that cell are stored. An example of the grid file is shown in Figure 7.9. We can see in the figure that department numbers 1 and 2 are mapped to the value 0 on the scale, similarly dept number 8 is mapped to the single value 4. As we can see from the figure, age is divided into groups. A query for Dno = 3 and Age = 45 will map to cell number (1,3) in the grid array. The grid array can conceptually be applied to any number of search key. For a search query with n keys the grid will have n dimensions.

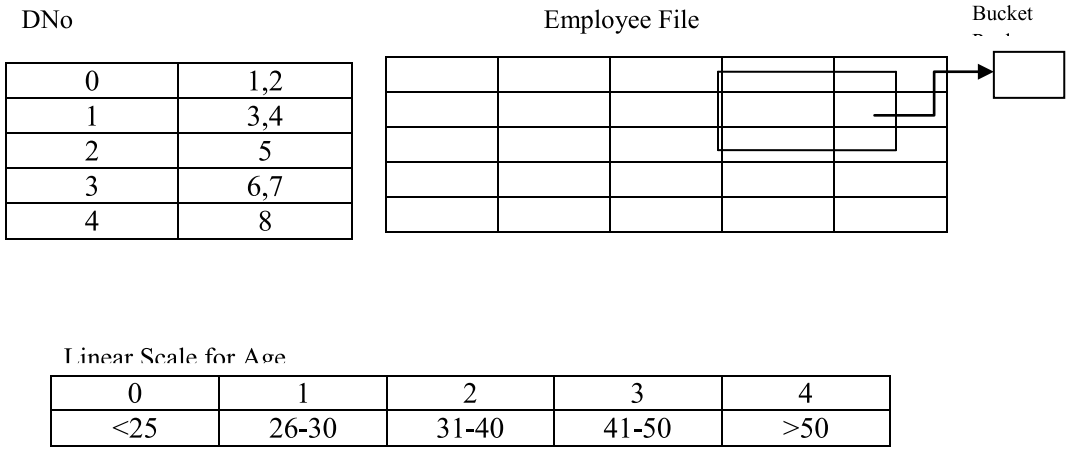


Figure 7.9: A grid array on DNo and Age attribute

Advantage:

1. It reduces time for multiple key accesses.

Disadvantages:

1. It results into space overhead in terms of grid structures.
2. For dynamic files, frequent file reorganization is required which increases the maintenance cost.

7.9 SUMMARY

In this unit we studied various file organization methods. We discussed briefly about the structure of the magnetic disk and how data is stored and retrieved from the disk. We then discussed the way database files can be stored on the disk. Then we started a detailed discussion about the physical placement of records of the database file on the disk which is also known as file organization. We discussed different types of file organization techniques such as heap file organization, sequential file organization, direct file organization, indexed file organization and multi-key file organization.

While heap file organization is simplest and easy to understand, it creates problems in retrieval, deletion and modification of records. Sequential file organization stores the records sequentially based on some field of the data, it is the least used technique of file organization. Hashed file organization provides the fastest access mode based on the hash key field; however it works mainly on equality comparisons with the hash key field. Indexed file organization provides access structures in addition to the primary file organization. Multi-key file organization provides indexes based on multiple keys rather than a single key. They are very useful for fast processing of queries.

7.10 REVIEW QUESTIONS

1. What is the difference between primary and secondary storage?
2. What is the difference between fixed length records and variable length records? Discuss the reasons for having variable length records in a database file.
3. What is the difference between file organization and access method?
4. What is the difference between primary, secondary and clustering indexes? Which amongst them are dense indexes?
5. How does multi-level indexing improve efficiency in searching index file?

UNIT-8

TRANSACTION PROCESSING CONCEPT

Structure

Bibliography

INTRODUCTION

In the last unit we discussed file organization and details regarding physical placement of database records on the disk as also various techniques for retrieving records as and when required. In this unit, we shall move a step forward and discuss how the Database Management System handles insertion, deletion and updation processes in the database.

A user normally does not require access to the whole database at a time and therefore, only the portion of the database (typically a few blocks) which is required by the user is loaded into the main memory. Any kind of operation on the data done by the user is initially saved in the main memory itself. Moreover, a DBMS typically allows multiple users to access the database at a time. It is the responsibility of the database management system to write the results of any kind of update operation performed by the users on the database back to the disk. As you might have guessed, this process requires ensuring that the updates are not lost before they are written back to the disk due to some kind of failure as well as maintaining the consistency of the database before and after the updates are applied. In this unit, we shall discuss in detail the challenges in implementing these database operations and the techniques used for managing them. This unit is organized as follows:

Firstly, we would discuss the concept of concurrency and its advantages; this would be followed by an introduction to the concept of a transaction. We will then discuss transaction schedules, their serializability and categorization of schedules based on the concept of serializability. This will complete the discussion related to concurrency of the system and how DBMS handles concurrency. Next, we would move on to discuss the concept of recoverability and various techniques used by the DBMS for recovery. This will also involve a discussion on the problem of deadlock which arises due to implementing concurrency in the system and a brief discussion on handling deadlock. The last section will give a summary of what we have learned in this unit and will be followed by review questions.

Objective

After learning this unit, you should be able to:

1. Explain the concept of concurrency and its importance
2. Explain what a transaction is, the concept of serializability and its usage in transaction management
3. Understand and explain the different ways of recovery of a system

4. Define the concept of deadlock and theoretically explain its handling

8.1 CONCEPT OF CONCURRENCY

You must have studied in Operating Systems course about single and multi-user systems. For the sake of revision, we would discuss them in brief here. **Single-user** systems are the ones where at most one user can access the system at a time. Such systems are mostly offline and are typically limited to a single PC. All the things that we have studied up till now have been taught keeping in mind that a single user is accessing the system at a time. Such systems are easy to manage as any kind of operation on the database will be performed by the sole user who accesses it and the database integrity can always be insured. **Multi-user** systems on the other hand, can be accessed by more than one users at a time such as the railway ticket reservation system. Such systems are distributed over a large number of access devices and are often online. These systems are based on the concept of multiprogramming. **Multi-programming** systems allow processes to run in **interleaved** manner. What this means is that a process will be executed for some time after which it will be suspended and some other process will be executed. The maximum number of processes that a system can execute efficiently in this manner is known as the **degree of multiprogramming**. Concurrent execution of processes helps in improving the CPU utilization and hence the efficiency of the system. This is because a process while executing may require I/O and other types of interrupts; for example, reading a file from disk. Now, when a process is reading a file from disk the CPU sits idle until that task is completed. The concept of concurrency involves temporarily suspending the running process at this point until it completes the interrupt and restart its execution at a later point of time. The CPU during this period can execute some other process and hence, the efficiency of the system can be increased. Since, all these things are hidden from the end user; it gives the impression that the system is running more than one program at the same time.

For providing such functionalities however, the concurrency control system has to manage the interleaving of different processes. It does increase the system overhead as the concurrent execution of processes may also result into different types of problems. We shall discuss how the concurrency control manager handles this in great detail in the coming sections. But before that we need to be aware of the concept of a transaction and its properties.

One last point that you should keep in mind while reading this unit is that it is possible to parallely execute different processes (without any kind of interleaving) in a system using more than processors. However, most of the discussion in this unit will be based on the idea that the system has a single processor and multi-programming is achieved by means of interleaving only.

Advantages of Concurrency:

1. It leads to better utilization of CPU and hence increases the system throughput (remember, throughput is the number of processes completed per unit time)
2. It reduces the waiting time for transactions. Concurrent execution allows CPU to be allocated to all the processes currently in the queue, concurrently and hence reduces the overall waiting time for execution of all the transactions.

Disadvantages

There are basically three types of problems that may arise due to concurrent execution of transactions. These are:

1. Lost Update Problem
2. Dirty Read Problem, and
3. Incorrect Summary Problem

   These problems will be discussed in detail after we complete the study of the concept of transactions.

8.2 TRANSACTION SYSTEM AND SCHEDULES

8.2.1 Transaction

A transaction is a **logical unit** of database processing. Being a ”logical” unit of work implies that a transaction may actually include more than one operations of database processing but it will logically be treated as a single unit by the transaction management system and since it is a “unit” of work; it will either be performed in its entirety or not performed at all.

Usually a transaction includes one or more database operations such as read, write, insert, delete, update etc. and a single application program usually has more than one transactions embedded in it. Transactions can be specified using BEGIN and END statements in SQL. These statements define transaction boundaries and all SQL statements within them, execute together.

The sequence of operations defined within a transaction take the database from one consistent state to another consistent state without necessarily preserving the database consistency during the intermediate steps. Consider for example, a transaction in an inventory management system, where 10 quantities of an item are to be taken from the stock and added to a customer’s bill. It is clear that after 10 is subtracted from the database item (say X) and before it is added to the customer’s bill item (say Y); the

database is in an inconsistent state. However, the consistency is preserved before the transaction started and it is also preserved after the complete execution of the transaction. This should give you an idea about why it is required to execute a transaction in its entirety i.e. either executed totally or not executed at all. It is the responsibility of transaction management system of the database to ensure atomicity of the transaction.

We will now discuss different states of a transaction during execution but before that let us briefly discuss the database model considered for understanding transaction processing concepts.

A transaction can potentially access any database item. This item could be a single field in a record or the entire record, it could be a database table or even more than one tables in the database. For the sake of convenience, we refer any item accessed by a transaction as a named entity say 'X'. All the discussion following this would be independent of the size of the data item accessed by the transaction and it will be based on this simplified model of database. Using this (simplified) model all possible operations that can be performed by a transaction can be covered by the following two operations:

1. READ(X): read the database item X to a local program variable which is also supposed to be named X for simplicity.
2. WRITE(X): if the transaction somehow modifies the variable X, it needs to be written back to database (in place of original item X). This is done using the write operation.

8.2.2 Different States of a Transaction

A transaction can be in one of the following four different states at any point of time:

1. **Active**: This state implies that the transaction is being executed. This is the initial state of a transaction
2. **Partially Committed:** A transaction is said to be partially committed after it has successfully executed its last statement
3. **Committed:** After a transaction is partially committed, the transaction recovery subsystem needs to make sure that the transaction will not be undone in case of a system failure. (For example, by recording the changes made by the transaction in the system log). Once this check is done the transaction is said to be committed.
4. **Failed:** This state implies that either the transaction could not be executed normally or the changes made by the partially committed

transaction could not be recorded and hence the transaction must be aborted.

5. **Aborted:** This implies that the transaction has ended unsuccessfully and hence any changes made by the transaction that may have been applied to the database must be undone (also known as **rollback)**. The state transition diagram of a transaction is given in Fig 8.1.

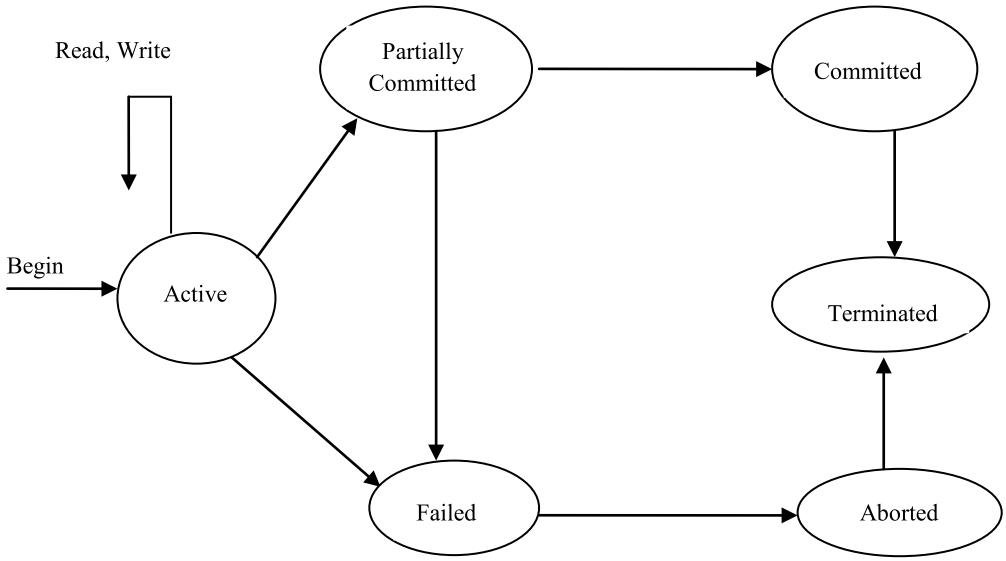


Fig 8.1: Different states of a transaction

8.2.3 Desirable Properties of a Transaction

In order to ensure the integrity of the data items accessed by various transactions; every transaction is supposed to possess some properties. These are also known as **ACID** properties:

1. **Atomicity**: Implies that each transaction is an atomic unit of processing i.e. it is either executed in its entirety or not executed at all. It is the responsibility of the transaction recovery subsystem to ensure atomicity of a process.

2. **Consistency**: A transaction must after its successful execution move the database from one consistent state to other. It is generally considered to be the responsibility of the programmer to ensure that if the database was in a consistent state before the execution of transaction; it remains in a consistent state after the successful termination of the transaction.

3. **Isolation**: Every transaction should execute in isolation with all others i.e. concurrently executing transactions should not interfere

with each other. Isolation is enforced by the concurrency control subsystem of the DBMS.

4. **Durability**: The changes made in the database by a committed transaction must persist in the database even in the case of a system failure i.e. the information saved by committed transactions must not be lost. Ensuring durability is the responsibility of the transaction recovery subsystem of DBMS.

8.2.4 Need for Controlling Concurrent Execution of Transactions

We shall now discuss the three types of problems that may arise due to concurrent execution of transactions. For understanding these problems we shall assume that two transactions denoted by T1 and T2 are executing concurrently in the system. Let us suppose that transaction T1 reads a database item X, subtracts 20 from it and adds this value to database item Y, while transaction T2 reads the database item X and doubles its value as shown below (Fig 8.2):

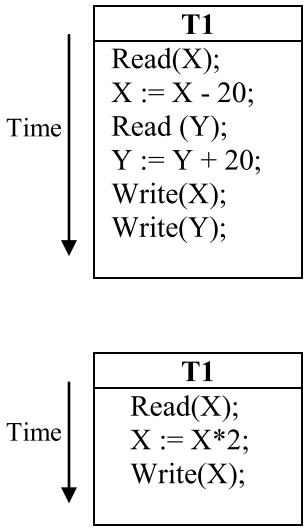


Fig 8.2: Two transactions T1 and T2 submitted simultaneously to the DBMS

Lost Update Problem: This type of problem arises when two transactions access the same database item and one of them updates it, but due to interleaving of transactions, the second transaction, which was also accessing the same database item reads it before the first transaction writes the updated value back to the database. In such a scenario, the updated value of one of the transactions will be lost based upon which transaction gets to execute the 'Write' statement first. Let us understand this with the

help of an example. In Fig 8.3, we show a possible scenario of concurrent execution of the two transactions T1 and T2.

In the first part of Fig. 8.3, transaction T1 reads the item X and subtracts 20 from it before it is interleaved and transaction T2 reads the value of X and multiplies it with 2. The transaction T2 is however now interleaved and transaction T1 completes with updating the value of X to its new value (X-20). Now transaction T2 executes its last statement and updates the value of X to the new value (X*2). Thus, the update performed by transaction T1 is simply lost. For an external user, it would seem as if the transaction T1 never manipulated the value of X! Similar would be the case for Transaction T2 in the second part of Fig. 8.3.

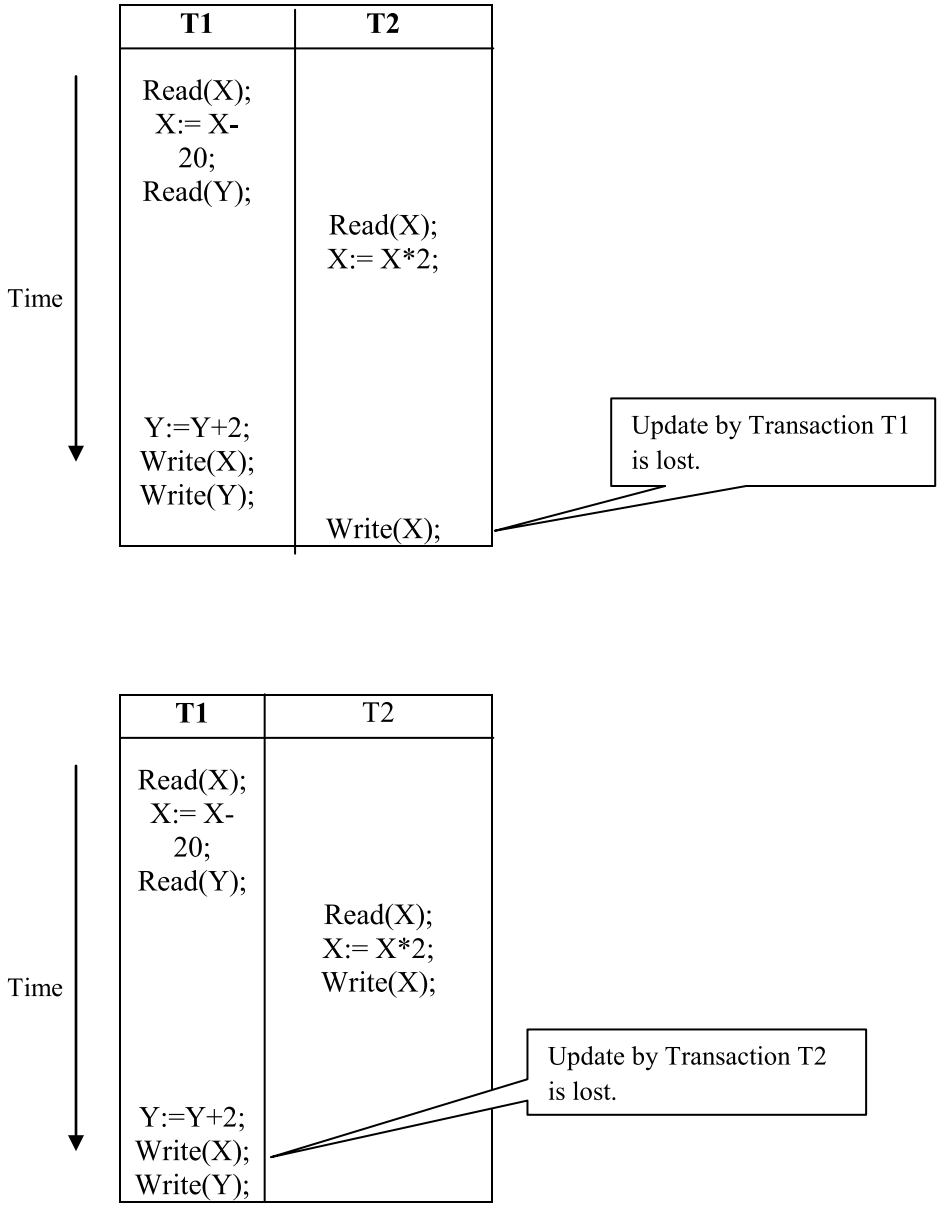


Fig. 8.3: The lost update problem arising due to concurrent execution of transactions

Dirty Read Problem: Now, consider another scenario of concurrent execution of transactions. Here, transaction T1 reads the item X and updates its value to a new value. However, the transaction T1 later fails due to some reason and therefore, the value of item X updated by the transaction T1 needs to be changed to its original value. Suppose, another transaction T2 which is also executing simultaneously in an interleaved manner somehow reads the value of X before it is changed back to its old value, then the value of X as read by T2 is incorrect. This type of read is known as *Dirty Read*. The problem of dirty read or incorrect read is represented in Fig 8.4.

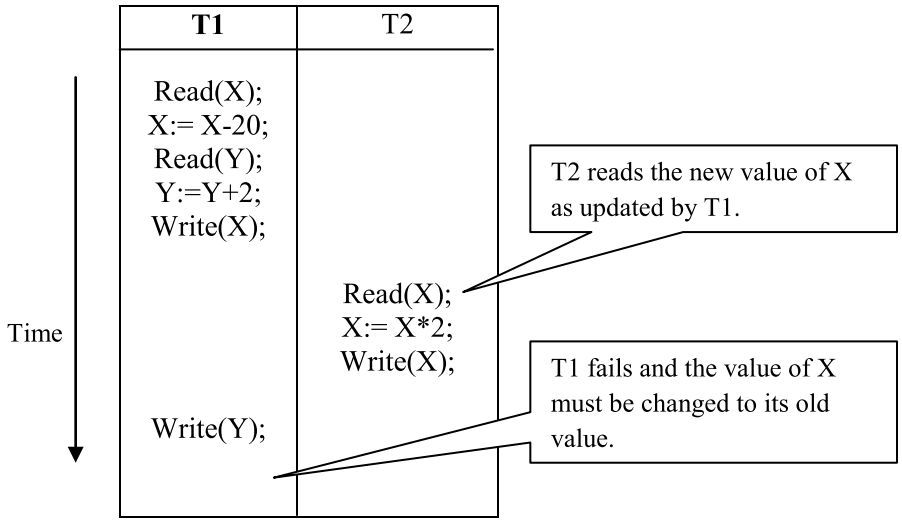


Fig. 8.4: The dirty read problem arising due to concurrent execution of transactions

Incorrect Summary Problem: Some database transactions need to calculate the aggregate over values of certain variables. In case of concurrent execution, the transaction calculating the aggregate value might read some values of variables before update and some values after update, this might result into an incorrect aggregate value. Let us take an example to make the things more clear. Suppose transaction T1 is trying to get the aggregate of variables X and Y in the database while transaction T2 reads item Y and subtracts 20 from it and adds this value to variable X. A possible order of execution of transactions T1 and T2 is given in Fig. 8.5. Here, transaction T1 has read the old value of X but new value of Y due to interleaving of operations and thus calculates the wrong aggregate.

By looking at these examples, it must have become clear to you that some kind of concurrency control must be applied in the database in order to maintain the consistency of the database when transactions are executing in an interleaved manner. In the following sections, we shall discuss the

concept of serializability and how it is used for ensuring the consistency of database during concurrent execution of transactions.

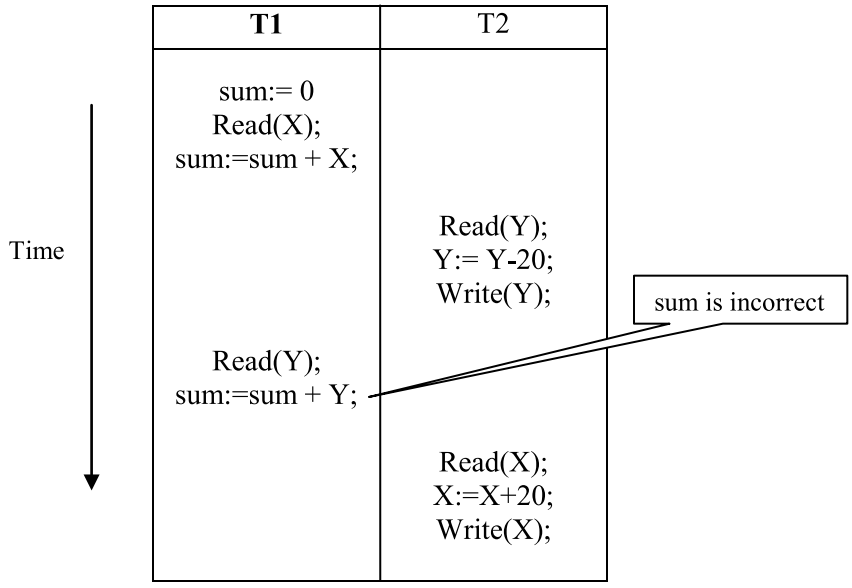


Fig. 8.5: The incorrect summary problem arising due to concurrent execution of transactions

8.2.5 Schedules

A schedule S of transactions of a database system is defined as the sequence of operations of the transactions appearing in S in their chronological order. In other words, a schedule S of n transactions T1, T2… Tn consists of all the operations of transactions T1, T2… Tn in the order in which they are executed with time. Here, it should be noted that normally a schedule includes all the operations of each transaction including commit and abort operations. However, for the purpose of concurrency control and recovery, only read and write operations are considered.

Check Your Progress

1. What is the difference between partially committed and committed transactions?
2. What are the advantages of concurrency?
3. How is dirty read problem different from incorrect summary problem when both of them involve reading of a database item by one transaction that is updated by some other transaction.

8.3 SERIALIZABILITY OF SCHEDULES

Serial and Non-serial Schedules: A schedule S is said to be a serial schedule if for every transaction T that appears in S, all the operations of T are executed consecutively otherwise the schedule is said to be non-serial. Clearly, a serial schedule corresponds to a system where there is no interleaving and hence no concurrent processing of transactions occurs. For concurrent transactions, the schedule is necessarily non-serial. Also, since serial schedules correspond to transactions executing serially one after other, serial schedules are always correct. For a schedule consisting of n transactions, there are n! possible serial schedules.

Serializable Schedules: There is another category of schedules which are non-serial but are **equivalent** to some serial schedule consisting of the same set of transactions as the non-serial schedule. Such schedules are known as Serializable schedules. Any schedule consisting of n transactions that is serializable, is equivalent to at least one of the n! possible serial schedules over the same n transactions. As serial schedules are always correct, any schedule that is Serializable is also correct.

We now turn our attention to equivalence of two schedules. There are several different ways in which two schedules can be said to be equivalent. The simplest method is to compare the effect of transactions on the database. According to this criterion, two schedules can be said to be equivalent if they result into same final state of the database. Such schedules are known as **Result Equivalent.** Result equivalence of schedules is however not considered a good criterion because two schedules can also be result equivalent accidentally. The more commonly used criteria for deciding equivalence of schedules is conflict equivalence and view equivalence. We shall discuss these concepts and the serializability of schedules based on conflict and view equivalence in the next section.

8.4 CONFLICT AND VIEW SERIALIZABLE SCHEDULES

8.4.1 Conflict Serializability

Conflict serializability is defined in terms of conflict equivalence of schedules which is concerned only with the conflicting operations in a schedule and not all operations. We shall first discuss when two operations in a schedule are said to conflict and then discuss equivalence of schedules based on it.

Two operations in a schedule are said to be conflicting if the following conditions hold:

i The two operations belong to different transactions

ii Both of them access the same data item and

iii At least one of the two operations is a *Write* operation

Based on the above criteria, two schedules S and S' are said to be conflict equivalent if the order of conflicting operations in the two schedules are same. Further, a schedule S is said to be conflict serializable if it is conflict equivalent to some serial schedule of the same set of transactions. An example illustrating conflict equivalent schedules is given in Fig. 8.6 and 8.7. Fig. 8.6 shows two possible serial schedules of two transactions T1 and T2 while Fig. 8.7 shows two non-serial schedules of the same transactions T1 and T2. As can be seen from figure, Schedule D is conflict equivalent to Schedule A while Schedule C is not equivalent to any serial schedule.

8.4.2 View Serializability

Extending on the same lines, we can define a schedule as view serializable if it is view equivalent to some serial schedule of the same set of transactions. Two schedules S and S' consisting of the same set of n transactions T1, T2… Tn are set to be view equivalent if the following three conditions hold for every transaction Ti:

i. If Ti reads the initial value of data item X in S then, Ti must also read the initial value of X in S'

ii. If Ti reads the value of a data item X in S and that value of X was produced as a result of a *Write(X)*operation executed by another transaction Tj in S then, Ti must read the value of X produced by *Write(X)* operation performed by Tj in S' also and

iii. If the last *Write(X)* operation on data item X in S is performed by a transaction Ti then Ti must also perform the last *Write(X)* operation on data item X in S'.

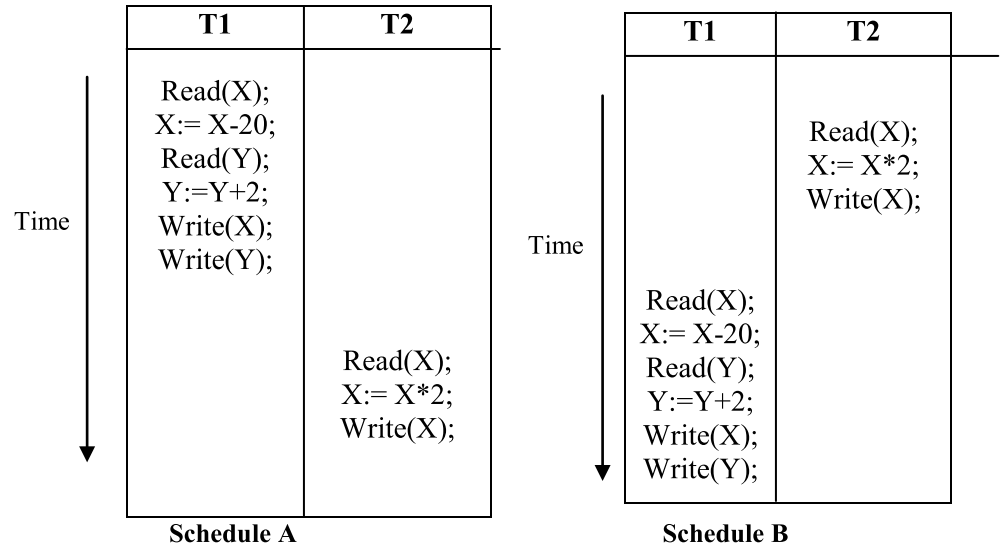


Fig. 8.6: Serial schedules consisting of transactions T1 and T2.

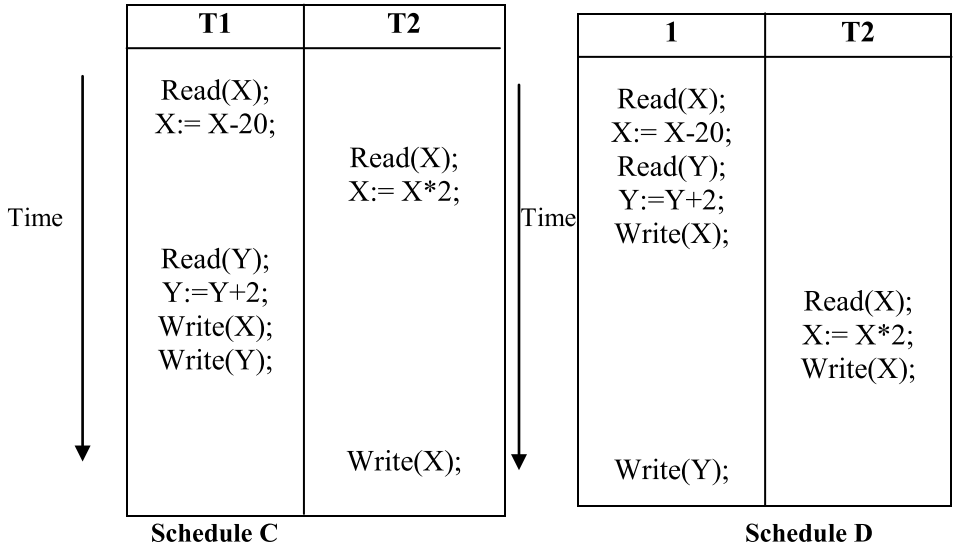


Fig. 8.7: Two non-serial schedules of transactions T1 and T2

The first two conditions ensure that each transaction in the two schedules reads the same value of the database item and hence produces same result. Further, the inclusion of third condition ensures that both the schedules produce the same final state of the system.

It should be noted that conflict serializability is more restrictive as compared to view serializability i.e. a conflict serializable schedule is always view serializable while a view serializable schedule may or may not be conflict serializable. The difference between the two schemes lies in the assumption of constrained writing. The **constrained write assumption** requires each *Write(X)* operation to be preceded by a *Read(X)* operation. In **unconstrained writing** a transaction may write a new value of database item X that is independent of its previous value. Such a write

is called **blind write**. Any schedule that is conflict serializable but not view serializable contains transactions that perform blind writes. Let us understand this with the help of an example. The schedule S consists of three transactions T1, T2 and T3 as shown in Fig. 8.8. Cleary, S is view serializable to the serial schedule consisting of transaction T1 followed by T2 followed by T3. S is however, not conflict serializable to any possible serial schedule of transactions T1, T2 and T3.

T1	T2	T3
Read(X);		
	Write(X);	
Write(X);		
		Write(X);

Fig. 8.8: A schedule that is view serializable but not conflict serializable

8.5 Testing for Serializability

In order to figure out if a given schedule S is serializable or not, you don't have to draw all possible sequences of transactions in S rather, algorithms exist for testing the serializability of a given schedule. In this section, we shall discuss one such algorithm which is used for testing the conflict serializability of two schedules. Testing for view serializability is a comparatively difficult task. The problem of testing view serializability of two schedules is NP-Hard and therefore, finding a polynomial time algorithm for testing view serializability of a schedule is highly unlikely.

Suppose, we want to test the serializability of a schedule S which consists of n transactions T1, T2, … Tn. For testing the conflict serializability of S, only *Read(X)* and *Write(X)* transactions are considered. The first step is to generate a precedence graph. Precedence graph contains a set of n nodes which correspond to the n transactions that appear in S. Any two transactions Ti and Tj ($1 \leq i, j \geq n$) are connected by an edge if an operation in Ti precedes a conflicting operation in Tj i.e. the operation in Ti occurs before the operation Tj in schedule S. This process is repeated for each set of conflicting operations. The step-by-step algorithm for constructing the precedence graph is given below:

Algorithm for testing Conflict Serializability of a Schedule S

Step 1: Create a node in graph G for each transaction Ti occurring in S

Step 2: Create an edge starting from node Ti and ending at node Tj if, Ti executes

a *Read(X)* before Tj executes a *Write(X)*.

Step 3: Create an edge starting from Ti and ending at Tj if, Ti executes a *Write(X)* before Tj executes a *Read(X)*.

Step 4: Create an edge starting from Ti and ending at Tj if, Ti executes a *Write(X)* before Tj executes a *Write(X)*.

Schedule S is serializable if and only if the precedence graph generated in the above manner does not contain any cycles.

Check Your Progress

1. Test if the schedule given in Fig. 8.9(a) below is serializable or not.

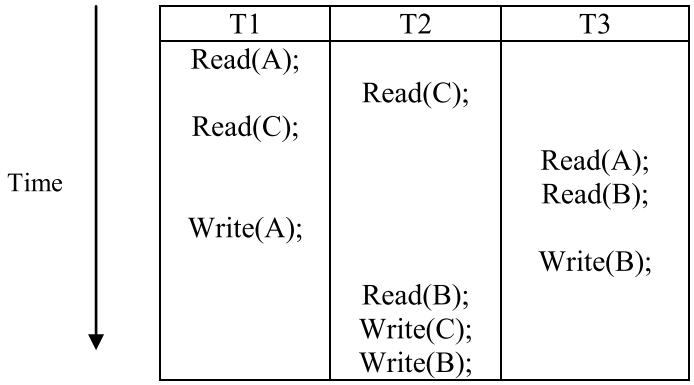
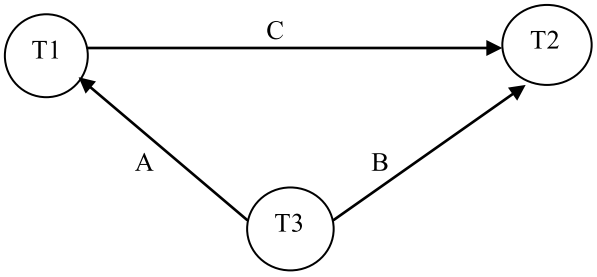


Fig. 8.9(a): Schedule consisting of three concurrently executing transactions with the progress of time.

Solution:



Since, the precedence graph does **not contain a cycle**, the schedules are Serializable. The corresponding serial schedule is *T3 followed by T1 followed by T2* as shown below (Fig. 8.9(b)):

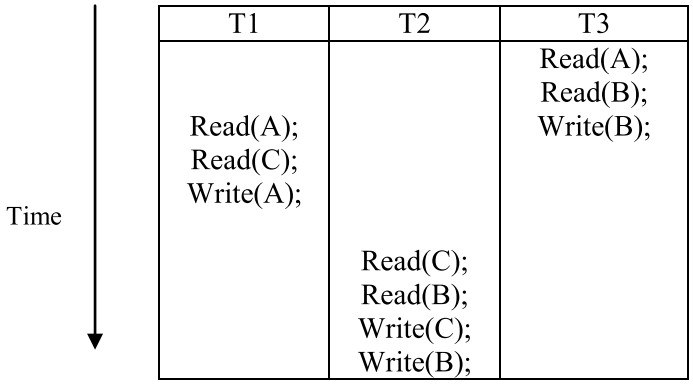


Fig. 8.9(b): Schedule given in figure 8(a) re-written as a serial schedule

Hint: You can take dummy values for variables A, B and C and check that both schedules 8(a) and 8(b) give exactly same result.

Uses of Serializability: As we have already discussed serializable schedules are always correct and therefore, the operations performed by serializable schedules will never leave the database in an inconsistent state. Hence, it is desirable to ensure serializability of every schedule in a system where transactions are executing concurrently. Practically, however it would become very difficult to test the serializability of schedules in a large database system where new transactions are submitted all the time such as an online reservation system. Further, testing the schedules for serializability after all the transactions have been performed will amount to cancellation of a large number of transactions. Therefore, most database systems apply concurrency control techniques that guarantee the serializability of the schedules. One such technique which is widely used in various database systems is two-phase locking. There are other techniques also such as timestamp ordering, multiversion protocols etc.

8.6 RECOVERABILITY

The fundamental goal of a transaction management system is to ensure that whenever a transaction is submitted to the DBMS, either all the operations of the transaction are executed successfully and the changes made by the transaction are recorded in the database or no changes are made by the transaction as also no other concurrently executing transactions are affected by it. It should never be possible for a transaction to execute partially i.e. some operations are applied to the database while others are not. If a transaction fails after executing some of its operations, the database must be recovered from any inconsistency that might have arisen due to the failure of the transaction. A transaction might fail due to various reasons. We shall discuss the reasons for failure of a transaction and techniques used for recovery from failures in more detail in the next

section. Let us first understand the concept of recoverability in a database system.

So far, we have characterized the schedules on the basis of their serializability i.e. from the view point of maintenance of consistency of the resulting database system. Schedules can also be categorized on the basis of their recoverability from failures. Suppose, a transaction fails due to some reason then it must be aborted and any changes made by the transaction to the database must be undone. Further, if there are other transactions that are dependent on the failed transaction (they have read a data item that was written by the failed transaction), these transactions must also be aborted. The schedules for which this is possible are called recoverable schedules while the schedules for which this cannot be done are called irrecoverable schedules. Let us understand this with the help of an example.

Normally, we assume that any transaction that has committed should never be rolled back i.e. the changes made by a committed transaction must persist in the database even if some other concurrently executing transaction fails. Now, keeping this constraint in mind, imagine a scenario where a transaction T1 reads a data item X updated by another transaction T2. T1 makes changes to this data item, records the new value successfully in the database, completes its execution and reaches the commit point. Meanwhile, transaction T2 fails for some reason and must be aborted. Now, if we were to undo the changes made by T2 we would necessarily require transaction T1 to be rolled back which cannot be done as T1 has already committed. Thus, this schedule is irrecoverable and must not be permitted.

In a recoverable schedule, no committed transaction is ever rolled back. However, a non-committed transaction may need to be rolled back during the recovery process because it has read a data item updated by the failed transaction. This phenomenon is known as **cascading rollback**. There are techniques for avoiding cascading rollbacks. A schedule that does not involve cascading rollback is known as cascadeless schedule. A schedule S is cascadeless if, every transaction Ti in S, only reads values of data items that are written by committed transactions.

8.7 RECOVERY FROM TRANSACTION FAILURE

In this section, we shall first discuss the different types of failures that might occur in a database system. We will then focus on a specific category of failures i.e. transaction failures. We shall study the commonly used techniques for recovering from transaction failures.

8.7.1 Types of Failures

Logical Failure: Logical errors might cause a transaction to fail. Such errors may arise due to some operations in the transaction itself, such as a division by zero, integer overflow etc. A transaction might also fail due to local errors such as non-availability of data item demanded by a *Read(X)* command. Further, the concurrency control system might demand some transaction to be aborted for ensuring consistency of the database.

System Failure: This type of failure affects the whole database system and not just a few transactions. It is caused when the system reaches an undesirable state where no transactions can be executed such as the occurrence of a deadlock, system crash due to problem in say DBMS software or a bug in operating system itself, memory crash etc. Such failures do not physically damage the system. Further, these types of failures only affect the volatile data of the system and the data stored in permanent storage are not affected.

Media Failure: Media failures are the result of physical damage to the system such as disk failure. Such failures might also affect the permanently stored data in the database.

In the next two sub-sections, we shall discuss techniques of recovery from transaction failures which occur most commonly in database systems. Recovery from media failure and catastrophic failures (power-cut, fire etc.) is rather complicated. We shall not go into the details of recovery from such types of failures. These types of failures are usually avoided by keeping multiple copies of database apart from the main system.

8.7.2 Log-Based Recovery

The most widely used recovery mechanism in database systems is the log-based recovery. The system maintains a log of every transaction taking place in the database. The information stored in the log can be used for recovery. The system log is stored on the disk so that it is not damaged except in case of disk failures or catastrophic failures. The log is also periodically archived on a tape in order to allow recovery of database in case of catastrophic failures. The information stored in a typical system log is shown below.

1. **<Start, T> -** It indicates that the transaction T has started.
2. **<Write, T, X, *old_value*, *new_value*> -** Indicates that transaction T has updated the value of database item X from *old_value* to *new_value.*
3. **<Read, T, X> -** indicates that transaction T has read the value of database item X.
4. **<Commit, T> -** indicates that the transaction T has completed successfully and the updates done by it can be recorded permanently to the database.

5. **<Abort, T>** - indicates that the transaction T has been aborted.

It should be noted that the *Read(X)* operation by a transaction need not be stored in the log for the purpose of recovery (Step No. 3 above). However, it might be stored if the system log is also additionally used for the purpose of auditing etc. The recovery of a database system from a crash amounts to reading the log entries and either undoing or redoing the operations individually as stored in the log. You might be wondering why a redo operation might be required during the recovery process. Suppose, a transaction executed successfully and recorded all its entries to the system log and subsequently moved to committed stage. However, a system crash occurred in the meantime before the changes made by the transaction could be recorded to the actual database stored on the disk. In such a case, the transaction would be required to be redone at the time of system recovery which can be done by reading the log and changing all the variable values updated by the concerned transaction from *old_value* to the *new_value*.

8.7.3 Checkpoints

As you have just studied, the DBMS maintains system log in order to recover from transaction failures. In case of a failure, the operations are either undone or redone based on the information stored in the log. For conducting the recovery operation, we need to search the entire log. This is not the ideal approach mainly because performing all the operations stored in the log would result into unnecessary re-doing of a number of operations (from older transactions) especially for large databases. This is because most of the transactions would have already written their updates to the disk and hence re-doing them would cause unnecessary delay in the recovery process. In order to avoid this, the system uses **checkpoints. Checkpoints** ensure that all the updates done by the successful transactions are recorded on the disk and the system is informed that the updates up to this point should not be redone in case of a failure. More specifically, following actions are performed during a checkpoint:

1. All running transactions are temporarily suspended
2. All the buffers residing in the memory that are modified by some transaction are written to the disk
3. A log record known as the <checkpoint> is written to the log
4. All log entries in the main memory are force-written to the disk

After, performing the checkpoints, the system resumes execution of the suspended transactions. During the recovery process, undertaken after a system failure, the system searches for the last checkpoint in the log entries and all the operations performed by transactions that committed before the last checkpoint are not redone as all of their updates have already been recorded to the disk.

It is the responsibility of the recovery management system to decide when to perform a checkpoint. A checkpoint can be performed after

certain fixed time period such as after every *n* minutes or after every *t* number of committed transactions. A checkpoint entry also requires some additional information regarding the currently active transactions in order to facilitate re-doing of operations during recovery.

8.8 DEADLOCK HANDLING

As we have discussed earlier, the concurrency control subsystem in the DBMS ensures the serializability of the transactions and this is done by using a number of techniques most common among them is two-phase locking. Implementation of any kind of locking technique results into a type of problem known as **deadlock**. Before discussing deadlock however, we need to gain some understanding of the concept of locks and the method of two-phase locking. In the next subsection, we shall briefly discuss these concepts just sufficient to give you an understanding of the deadlock situation and how the system handles deadlock.

8.8.1 LOCKS

A lock is associated with a database item X and indicates the state of the item. If the database item is locked then it cannot be accessed by a transaction. The locks are issued by the currently running transactions and are managed by the concurrency control subsystem. Locks are used for the purpose of concurrency control. For each database item X, a lock is defined that is associated with two operations *lock(X)* and *unlock(X)*. Whenever, a transaction requires access to a database item, it first issues a request for a lock on that item. If the variable is not already locked by some other transaction, the system immediately issues the lock to the requesting transaction otherwise the transaction has to wait until the lock is granted. In its simplest form, the lock is a binary variable that can take two values 0 (unlocked) and 1 (locked). Such a lock is called a binary lock. Binary lock guarantees mutual exclusion i.e. only one transaction can access the database item at a time. In order to ensure successful working of the lock, the operation of request for granting the lock and releasing the lock must be implemented as indivisible units also known as the critical section in operating systems. It implies that whenever a transaction requests for a lock on a database item X, it either ends up getting the lock or it is sent to the waiting queue for lock to be granted. It is never interleaved in between by some other transaction during this process.

Binary lock is generally considered too restrictive for database operations as it does not discriminate between a *Read(X)* and *Write(X)* operation. Generally, for when a transaction reads an item X, we might allow some other transaction to read the same item, though simultaneous write operations should never be allowed. For this purpose, multi-mode locks are used. Here, three locking operations are available viz. *Read_lock(X)*, *Write_lock(X)* and *Unlock(X)*. These locks are also known as read/write locks or shared/exclusive locks. The *R*ead_lock(X) operation is a shared

lock as multiple transactions can acquire a Read_lock on the item X. The Write_lock(X) operation on the other hand is exclusive lock because a write lock is exclusively granted to a single transaction at a time.

8.8.2 Two-Phase Locking

In two-phase locking protocol, all the locking operations of a transaction (Read_lock and Write_lock) precede the first unlock operation. The operations in such a transaction can be divided into two phases: **expanding phase** and **shrinking phase**. The first phase is the expanding phase where the transaction acquires locks on all the items needed by it. This is followed by the shrinking phase, where the transaction releases the locks acquired during the expanding phase. Also, if the transaction management system allows lock conversion (changing a Read_lock to a Write_lock and vice-versa) then upgrading of lock i.e. conversion from Read_lock to Write_lock is done in the expanding phase while downgrading (conversion of Write_lock into a Read_lock) is done in the shrinking phase. If the two-phase locking system is enforced for all the transactions submitted to the DBMS then, the resulting schedule is **guaranteed to be serializable** and hence the need for testing the serializability of schedules is eliminated. Two-phase locking reduces the amount of concurrency possible in a schedule, it is however desirable as it eliminates the need for checking the schedules for serializability.

An example illustrating the above concepts is given in Fig. 8.10(a).

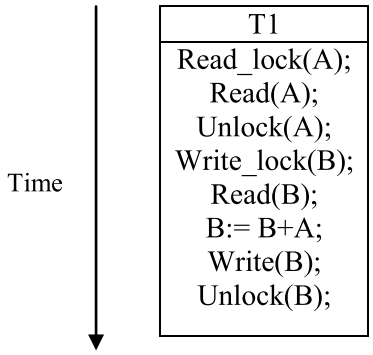


Fig. 8.10(a): Two-phase locking: A schedule that does not follow two phase locking

Here, transaction T1 does not follow two phase locking as the operation *Unlock(A)* precedes *Write_lock(B)* which is not permitted in two phase locking. In order to make transaction T1 follow the two phase locking protocol we may re-order its operations as below (Fig. 8.10(b)). As can be seen from Fig. 8.10(b), all the Read_lock and Write_lock operations occur before the first Unlock operation in T1' and hence T1' follows two-phase locking.

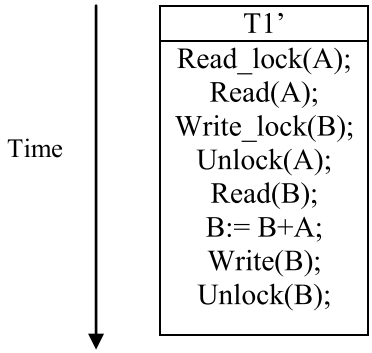


Fig. 8.10(b): Two-phase locking: Schedule T1 (Fig. 8.10(a)) re-written so as to follow two phase locking protocol

8.8.3 Deadlocks

We have studied in the last section about two phase locking protocol which is the most commonly used protocol for concurrency control in DBMS. It requires a transaction to lock all the data items it needs during the expanding phase and release them during the shrinking phase. This mechanism results into a problem known as deadlock. Deadlock is a situation where each transaction T in a set of two or more transactions waits for acquiring lock on an item locked by another transaction in the same set. Thus, the system enters into a state where no productive work could be performed by any transaction because each transaction is waiting for other transaction to release the lock on some item. Fig. 8.11(a) shows an example of a deadlock situation.

Handling of deadlock is an important part of a concurrency control subsystem in a DBMS. Deadlock handling in the database system is either based on a preventive mechanism or on detection mechanism. In preventive mechanism, certain protocols are used that prevent deadlock from occurring and hence ensure that the system never enters into a deadlock. While in detection mechanism, the system uses deadlock detection protocol to find out if a deadlock actually exists in the system. The second approach is more practical and is useful when it is known beforehand that a deadlock will rarely occur i.e. there are little chances that two transactions will require access to same database item at the same time. This is possible if the transactions are small and each transaction locks only a few database items. For long transactions and for transactions that require large number of database items, deadlock prevention protocols are better suited.

Deadlock Prevention:

Deadlock can be prevented in two phase locking if every transaction is allowed to lock *all the items it needs in advance*. If some of the items are not available, then the transaction does not lock any item and waits until it gets all the locks in advance. This approach is not practical as it further limits the amount of concurrency in the system. Another protocol for deadlock prevention requires ordering all the database items according to

their priority ranking and ensuring that a transaction that needs more than one item locks them in the order of their ranking. This method also limits the amount of concurrency in the system. A number of other protocols are also used. We shall not go into the details of these protocols for the sake of brevity.

Deadlock Detection:

As discussed earlier, the deadlock detection protocols test the system for existence of deadlock. The simplest method to detect if a deadlock exists in the system is to construct and maintain a **wait-for** graph (Fig. 8.11(b)). The wait-for graph is a directed graph where each node represents a transaction in the system. An edge is drawn from transaction Ti to Tj if, Ti is waiting for an item locked by transaction Tj. When transaction Tj releases the item required by transaction Ti, the lock is granted to transaction Ti and the edge is dropped. Deadlock is detected by existence of a cycle in the wait-for graph.

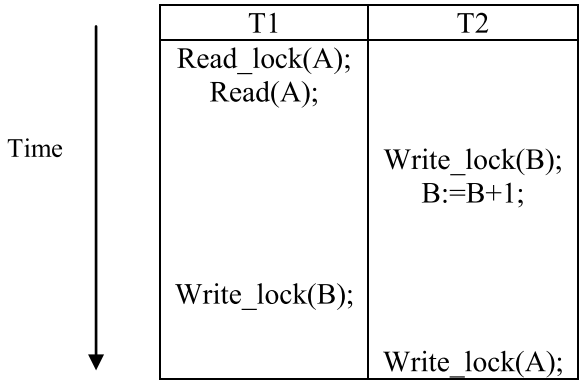


Fig. 8.11(a): Deadlock: Partial Schedule consisting of two transactions T1 and T2 that are in a deadlock situation.

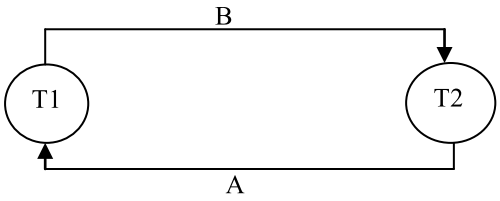


Fig. 8.11(b): Deadlock: Wait for graph for the Partial Schedule shown in Fig. 8.10(a) – cycle indicates existence of deadlock

Once, it is known that deadlock exists in the system some transactions must be aborted in order to end the state of deadlock. The process of choosing a transaction for abortion is known as **victim-selection**. Algorithms exist for deciding which processes to abort for ending the deadlock. A crucial part of a deadlock detection mechanism is deciding when to check a system for existence of a deadlock. For this purpose,

different criteria such as number of transactions executing currently, the average waiting time of transactions for locking an item may be used.

Check Your Progress

Test whether the system represented by the following partial schedule (Fig. 8.12) is in deadlock state or not.

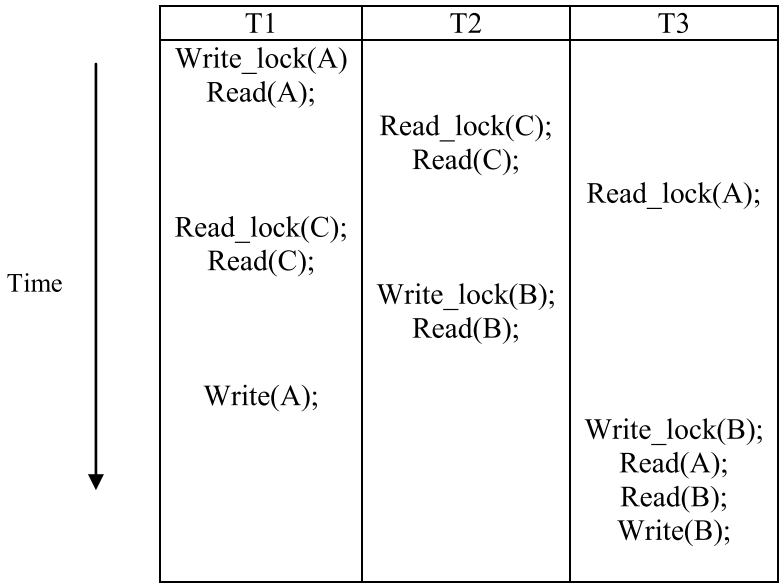


Fig. 8.12: Schedule consisting of three concurrently executing transactions with the progress of time.

8.9 SUMMARY

In this unit, we have mainly focussed on learning about concurrency. We discussed the concepts related to concurrency, the advantages of concurrent execution of transactions and the mechanism for controlling concurrently executing transactions.

We started with defining the concept of concurrency its advantages and disadvantages and also learned that almost of the real-world database systems are based on concurrency. Then we defined database transactions which are logical units of processing that are always executed in entirety. We learned that, every transaction is required to possess four fundamental properties known as **ACID** properties in order to ensure the integrity of the database system. We saw how concurrent execution of transactions helps in better utilization of CPU time and hence results into faster processing of transactions. Concurrently executing transactions however might interfere with each other and might leave the database in an inconsistent state. It is therefore, important to control the transactions executing concurrently. We discussed about the **concurrency control**

subsystem of the DBMS which is responsible for ensuring the proper execution of concurrent transactions. The concurrency control subsystem uses **serializability** of the schedules of transactions in the database as the criterion for ensuring consistency of database. It follows certain protocols which make sure that the resulting schedule for all the concurrently executing transactions in the system is **serializable** and hence correct.

After learning about concurrency, we discussed several types of failures that might occur in a database system and learned that different mechanisms are used in databse management systems for recovery from failures. **Recovery** from transaction failures is mainly done by maintaining a **log** of all the transactions executing in the database. In case of a failure, the system checks the log entry and restores the database into a consistent state by either undoing or re-doing the operations stored in the log. **Checkpoints** help in deciding the last consistent state of the database system and reduce the amount of re-work required to be done in case of a failure. Lastly, we discussed about deadlocks. **Deadlocks** occur in database systems because of implementation of concurrency control protocol (**two-phase locking**). Database systems handle deadlock either based on deadlock prevention strategies or based on mechanisms of deadlock detection.

8.10 Review Questions

1. Discuss different types of failures that might occur in a database system. Read more about how recovery is done for them.
2. What is the difference between deadlock detection and deadlock prevention? Which of the mechanisms is better suited for a large database with long duration transactions.
3. Discuss the importance of a system log. Briefly discuss the entries in system log. Where is the system log stored?
4. What are the desirable properties of a transaction? Discuss.

Bibliography

1. Fundamentals of Database Systems, Fifth Edition, R. Elmasari and S.B. Navathe
2. Database System Concepts, Fourth Edition, Silberschatz, Korth and Sudarshan
3. An Introduction to Database Systems, Seventh Edition, C. J. Date, Pearson Education

Notes

Notes

Notes

Notes

Notes