DIGITAL NOTES ON

DATABASE MANAGEMENT SYSTEMS

B.TECH II YEAR - II SEM
(2020-21)

DEPARTMENT OF INFORMATION TECHNOLOGY

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

(Autonomous Institution – UGC, Govt. of India)

(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – ‘A’ Grade - ISO 9001:2015 Certified)

Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, INDIA.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY DEPARTMENT OF INFORMATION TECHNOLOGY

SYLLABUS

II Year B. Tech. IT – II Sem **L/T/P/ C 3/0/0/3**

(R15A0509) DATABASE MANAGEMENT SYSTEMS

Objectives:

- To Understand the basic concepts and the applications of database systems
- To Master the basics of SQL and construct queries using SQL
- To understand the relational database design principles
- To become familiar with the basic issues of transaction processing and concurrency control
- To become familiar with database storage structures and access techniques

UNIT I:

Database System Applications, Purpose of Database Systems, View of Data – Data Abstraction – Instances and Schemas – Data Models – the ER Model – Relational Model – Other Models – Database Languages – DDL – DML – database Access for applications Programs – Database Users and Administrator – Transaction Management – Database Architecture – Storage Manager – the Query Processor.

Introduction to the Relational Model – Structure – Database Schema, Keys – Schema Diagrams. Database design and ER diagrams – ER Model - Entities, Attributes and Entity sets – Relationships and Relationship sets – ER Design Issues – Concept Design – Conceptual Design with relevant Examples. Relational Query Languages, Relational Operations.

UNIT II:

Relational Algebra – Selection and projection set operations – renaming – Joins – Division – Examples of Algebra overviews – Relational calculus – Tuple Relational Calculus (TRC) – Domain relational calculus (DRC).

Overview of the SQL Query Language – Basic Structure of SQL Queries, Set Operations, Aggregate Functions – GROUPBY – HAVING, Nested Sub queries, Views, Triggers, Procedures.

UNIT III:

Normalization – Introduction, Non loss decomposition and functional dependencies, First, Second, and third normal forms – dependency preservation, Boyce/Codd normal form.

Higher Normal Forms - Introduction, Multi-valued dependencies and Fourth normal form, Join dependencies and Fifth normal form

UNIT IV:

Transaction Concept- Transaction State- Implementation of Atomicity and Durability – Concurrent Executions – Serializability- Recoverability – Implementation of Isolation – Testing for serializability- Lock –Based Protocols – Timestamp Based Protocols- Validation- Based Protocols – Multiple Granularity.

UNIT V:

Recovery and Atomicity – Log – Based Recovery – Recovery with Concurrent Transactions – Check Points - Buffer Management – Failure with loss of nonvolatile storage-Advance Recovery systems- ARIES Algorithm, Remote Backup systems.

File organization – various kinds of indexes - B+ Trees- Query Processing – Relational Query Optimization.

TEXT BOOKS:

1. Database System Concepts, Silberschatz, Korth, McGraw hill, Sixth Edition.(All UNITS except III th)
2. Database Management Systems, Raghu Ramakrishnan, Johannes Gehrke, TATA McGrawHill 3rd Edition.

REFERENCE BOOKS:

1. Fundamentals of Database Systems, Elmasri Navathe Pearson Education.
2. An Introduction to Database systems, C.J. Date, A.Kannan, S.Swami Nadhan, Pearson, Eight Edition for UNIT III.

Outcomes:

- Demonstrate the basic elements of a relational database management system
- Ability to identify the data models for relevant problems
- Ability to design entity relationship and convert entity relationship diagrams into RDBMS and formulate SQL queries on the respect data
- Apply normalization for the development of application software’s

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

INDEX

UNIT-1

Introduction to Database Management System

The database is a collection of inter-related data which is used to retrieve, insert and delete the data efficiently.

It is also used to organize the data in the form of a table, schema, views, and reports, etc.

For example: The college Database organizes the data about the admin, staff, students and faculty etc.

Using the database, you can easily retrieve, insert, and delete the information.

Database Management System

- Database management system is a software which is used to manage the database. For example: MySQL, Oracle, etc are a very popular commercial database which is used in different applications.
- DBMS provides an interface to perform various operations like database creation, storing data in it, updating data, creating a table in the database and a lot more.
- It provides protection and security to the database. In the case of multiple users, it also maintains data consistency.

Characteristics of DBMS

- It uses a digital repository established on a server to store and manage the information.
- It can provide a clear and logical view of the process that manipulates data.
- DBMS contains automatic backup and recovery procedures.
- It contains ACID properties which maintain data in a healthy state in case of failure.
- It can reduce the complex relationship between data.
- It is used to support manipulation and processing of data.
- It is used to provide security of data.
- It can view the database from different viewpoints according to the requirements of the user.

Advantages of DBMS

- **Controls database redundancy:** It can control data redundancy because it stores all the data in one single database file and that recorded data is placed in the database.
- **Data sharing:** In DBMS, the authorized users of an organization can share the data among multiple users.
- **Easily Maintenance:** It can be easily maintainable due to the centralized nature of the database system.
- **Reduce time:** It reduces development time and maintenance need.
- **Backup:** It provides backup and recovery subsystems which create automatic backup of data from hardware and software failures and restores the data if required.

- **multiple user interface:** It provides different types of user interfaces like graphical user interfaces, application program interfaces

Disadvantages of DBMS

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.
- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

DBMS VS FILES

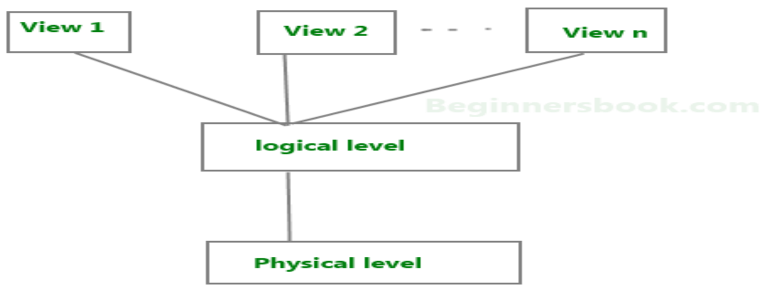
DBMS	File System
DBMS is a collection of data. In DBMS, the user is not required to write the procedures.	File system is a collection of data. In this system, the user has to write the procedures for managing the database.
DBMS gives an abstract view of data that hides the details.	File system provides the detail of the data representation and storage of data.
DBMS provides a crash recovery mechanism, i.e., DBMS protects the user from the system failure.	File system doesn't have a crash mechanism, i.e., if the system crashes while entering some data, then the content of the file will lost.
DBMS provides a good protection mechanism.	It is very difficult to protect a file under the file system.
DBMS contains a wide variety of sophisticated techniques to store and retrieve the data.	File system can't efficiently store and retrieve the data.
DBMS takes care of Concurrent access of data using some form of locking.	In the File system, concurrent access has many problems like redirecting the file while other deleting some information or updating some information.

View of Data in DBMS

Abstraction is one of the main features of database systems. Hiding irrelevant details from user and providing abstract view of data to users, helps in easy and efficient **user-database** interaction. In the previous tutorial, we discussed the three level of DBMS architecture, The top level of that architecture is “view level”. The view level provides the “**view of data**” to the users and hides the irrelevant details such as data relationship, database schema, constraints, security etc from the user.

Data Abstraction in DBMS

Database systems are made-up of complex data structures. To ease the user interaction with database, the developers hide internal irrelevant details from users. This process of hiding irrelevant details from user is called data abstraction.



Three Levels of data abstraction

We have three levels of abstraction:

Physical level: This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.

Logical level: This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.

View level: Highest level of data abstraction. This level describes the user interaction with database system.

Example: Let's say we are storing customer information in a customer table. At **physical level** these records can be described as blocks of storage (bytes, gigabytes, terabytes etc.) in memory. These details are often hidden from the programmers.

At the **logical level** these records can be described as fields and attributes along with their data types, their relationship among each other can be logically implemented. The programmers generally work at this level because they are aware of such things about database systems.

At **view level**, user just interact with system with the help of GUI and enter the details at the screen, they are not aware of how the data is stored and what data is stored; such details are hidden from them.

DBMS Schema

Definition of schema: Design of a database is called the schema. Schema is of three types: Physical schema, logical schema and view schema.

For example: In the following diagram, we have a schema that shows the relationship between three tables: Course, Student and Section. The diagram only shows the design of the database, it doesn't show the data present in those tables. Schema is only a structural view(design) of a database.

DBMS Instance

Definition of instance: The data stored in database at a particular moment of time is called instance of database. Database schema defines the variable declarations in tables that belong to a particular database; the value of these variables at a moment of time is called the instance of that database.

Data Models in DBMS

Data Model is a logical structure of Database. It describes the design of database to reflect entities, attributes, relationship among data, constrains etc.

Types of Data Models

There are several types of data models in DBMS. We will cover them in detail in separate articles(Links to those separate tutorials are already provided below). In this guide, we will just see a basic overview of types of models.

Object based logical Models – Describe data at the conceptual and view levels.

1. E-R Model
2. Object oriented Model

Record based logical Models – Like Object based model, they also describe data at the conceptual and view levels. These models specify logical structure of database with records, fields and attributes.

1. Relational Model
2. Hierarchical Model
3. Network Model – Network Model is same as hierarchical model except that it has graph-like structure rather than a tree-based structure. Unlike hierarchical model, this model allows each record to have more than one parent record.

Physical Data Models – These models describe data at the lowest level of abstraction.

Entity-Relationship Data Model:

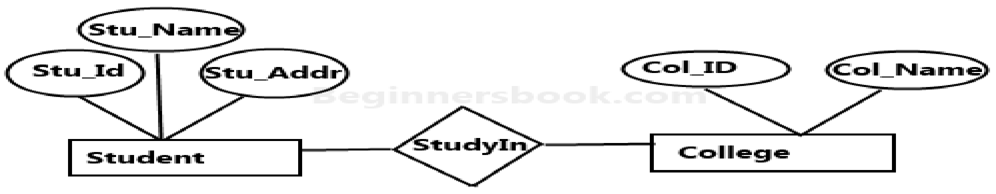
- An ER model is the logical representation of data as objects and relationships among them.

- These objects are known as entities, and relationship is an association among these entities.
- This model was designed by Peter Chen and published in 1976 papers.
- It was widely used in database designing. A set of attributes describe the entities.

Example: student_name, student_id describes the 'student' entity. A set of the same type of entities is known as an 'Entity set', and the set of the same type of relationships is known as 'relationship set'.

Entity Relationship Diagram (ER Diagram)

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Lets have a look at a simple ER diagram to understand this concept.



Sample E-R Diagram

Rectangle: Represents Entity sets.

Ellipses: Attributes

Diamonds: Relationship Set

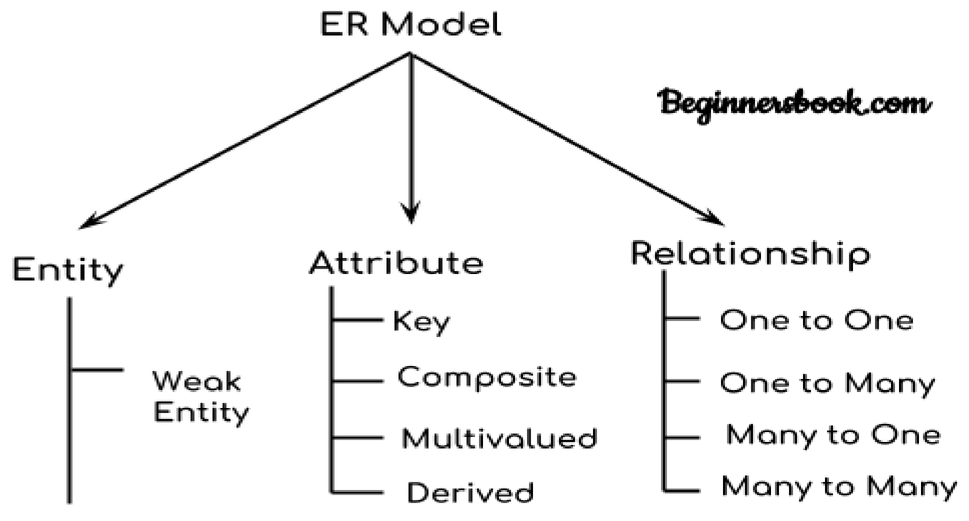
Lines: They link attributes to Entity Sets and Entity sets to Relationship Set

Double Ellipses: Multivalued Attributes

Dashed Ellipses: Derived Attributes

Double Rectangles: Weak Entity Sets

Double Lines: Total participation of an entity in a relationship set



Components of ER Diagram

As shown in the above diagram, an ER diagram has three main components:

1. Entity
2. Attribute
3. Relationship

1. Entity

An entity is an object or component of data. An entity is represented as rectangle in an ER diagram.
For example: In the following ER diagram we have two entities Student and College and these two entities have many to one relationship as many students study in a single college. We will read more about relationships later, for now focus on entities.

Weak Entity:
An entity that cannot be uniquely identified by its own attributes and relies on the relationship with other entity is called weak entity. The weak entity is represented by a double rectangle. For example – a bank account cannot be uniquely identified without knowing the bank to which the account belongs, so bank account is a weak entity.

2. Attribute

An attribute describes the property of an entity. An attribute is represented as Oval in an ER diagram. There are four types of attributes:

1. Key attribute
2. Composite attribute
3. Multivalued attribute
4. Derived attribute

3. Relationship

A relationship is represented by diamond shape in ER diagram, it shows the relationship among entities. There are four types of relationships:

1. One to One
2. One to Many
3. Many to One
4. Many to Many

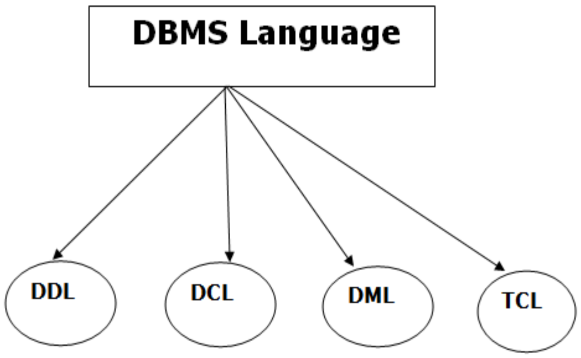
Object-Based Data Model. Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object- oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.

Semi-structured Data Model. The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to 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.

Historically, the **network data model** and the **hierarchical data model** preceded the relational data model. These models were tied closely to the underlying implementation, and complicated the task of modeling data. As a result they are used little now, except in old database code that is still in service in some places.

Database Languages

- A DBMS has appropriate languages and interfaces to express database queries and updates.
- Database languages can be used to read, store and update the data in the database.



1. Data Definition Language

- **DDL** stands for **D**ata **D**efinition **L**anguage. It is used to define database structure or pattern.
- It is used to create schema, tables, indexes, constraints, etc. in the database.
- Using the DDL statements, you can create the skeleton of the database.
- Data definition language is used to store the information of metadata like the number of tables and schemas, their names, indexes, columns in each table, constraints, etc.

Here are some tasks that come under DDL:

- **Create:** It is used to create objects in the database.
- **Alter:** It is used to alter the structure of the database.
- **Drop:** It is used to delete objects from the database.
- **Truncate:** It is used to remove all records from a table.
- **Rename:** It is used to rename an object.
- **Comment:** It is used to comment on the data dictionary.

These commands are used to update the database schema that's why they come under Data definition language.

2. Data Manipulation Language

DML stands for **D**ata **M**anipulation **L**anguage. It is used for accessing and manipulating data in a database. It handles user requests.

Here are some tasks that come under DML:

- **Select:** It is used to retrieve data from a database.
- **Insert:** It is used to insert data into a table.
- **Update:** It is used to update existing data within a table.
- **Delete:** It is used to delete all records from a table.
- **Merge:** It performs UPSERT operation, i.e., insert or update operations.
- **Call:** It is used to call a structured query language or a Java subprogram.
- **Explain Plan:** It has the parameter of explaining data.
- **Lock Table:** It controls concurrency.

3. Data Control Language

- **DCL** stands for **D**ata **C**ontrol **L**anguage. It is used to retrieve the stored or saved data.
- The DCL execution is transactional. It also has rollback parameters.

  (But in Oracle database, the execution of data control language does not have the feature of rolling back.)

Here are some tasks that come under DCL:

- **Grant:** It is used to give user access privileges to a database.
- **Revoke:** It is used to take back permissions from the user.

There are the following operations which have the authorization of Revoke:

CONNECT, INSERT, USAGE, EXECUTE, DELETE, UPDATE and SELECT.

4. Transaction Control Language

TCL is used to run the changes made by the DML statement. TCL can be grouped into a logical transaction.

Here are some tasks that come under TCL:

- **Commit:** It is used to save the transaction on the database.
- **Rollback:** It is used to restore the database to original since the last Commit.

Database access languages and application programming

Database access languages and application programming interfaces: DBMS provides dataaccess through a query language. Query language is a nonprocedural language – one thatlets the user specify what must be done without having to specify how. SQL is the de facto query language and data access standard supported by the majority of DMBS vendors.

Database communication interfaces: a current gen DBMS accepts end-user requests via multiple, different network environments. For ex, DMBS might provide access to database via the internet through the use of web browsers such as chrome or Mozilla.

Database Administrator (DBA) :

Database Administrator (DBA) is a person/team who defines the schema and also controls the 3 levels of database.

The DBA will then create a new account id and password for the user if he/she need to access the data base.

DBA is also responsible for providing security to the data base and he allows only the authorized users to access/modify the data base.

- DBA also monitors the recovery and back up and provide technical support.
- The DBA has a DBA account in the DBMS which called a system or superuser account.
- DBA repairs damage caused due to hardware and/or software failures.

Database users are the persons who interact with the database and take the benefits of database.

They are differentiated into different types based on the way they expect to interact with the system.

- **Naive users**: They are the unsophisticated users who interact with the system by using permanent applications that already exist. Example: Online Library Management System, ATMs (Automated Teller Machine), etc.
- **Application programmers**: They are the computer professionals who interact with system through DML. They write application programs.
- **Sophisticated users**: They interact with the system by writing SQL queries directly through the query processor without writing application programs.
- **Specialized users**: They are also sophisticated users who write specialized database applications that do not fit into the traditional data processing framework. Example: Expert System, Knowledge Based System, etc.

What is a Database Transaction?

A **Database Transaction** is a logical unit of processing in a DBMS which entails one or more database access operation. In a nutshell, database transactions represent real-world events of any enterprise. All types of database access operation which are held between the beginning and end transaction statements are considered as a single logical transaction in DBMS. During the transaction the database is inconsistent. Only once the database is committed the state is changed from one consistent state to another.

What are ACID Properties?

ACID Properties are used for maintaining the integrity of database during transaction processing. ACID in DBMS stands for **A**tomicity, **C**onsistency, **I**solation, and **D**urability.

- **Atomicity:** A transaction is a single unit of operation. You either execute it entirely or do not execute it at all. There cannot be partial execution.
- **Consistency:** Once the transaction is executed, it should move from one consistent state to another.
- **Isolation:** Transaction should be executed in isolation from other transactions (no Locks). During concurrent transaction execution, intermediate transaction results from simultaneously executed transactions should not be made available to each other. (Level 0,1,2,3)
- **Durability:** · After successful completion of a transaction, the changes in the database should persist. Even in the case of system failures.

Database System Architecture

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the **storage manager** and the **query processor** components. The storage manager is important because databases typically require a large amount of storage space. The query processor is important because it helps the database system simplify and facilitate access to data.

Storage Manager :

Storage Manager is a program that provides an interface between the data stored in the database and the queries received. It is also known as Database Control System. It maintains the consistency and integrity of the database by applying the constraints and executes the DCL statements. It is responsible for updating, storing, deleting, and retrieving data in the database.

Authorization Manager –

It ensures role-based access control, i.e,. checks whether the particular person is privileged to perform the requested operation or not.

Integrity Manager –

It checks the integrity constraints when the database is modified.

Transaction Manager –

It controls concurrent access by performing the operations in a scheduled way that it receives the transaction. Thus, it ensures that the database remains in the consistent state before and after the execution of a transaction.

File Manager –

It manages the file space and the data structure used to represent information in the database.

Buffer Manager –

It is responsible for cache memory and the transfer of data between the secondary storage and main memory.

It contains the following components –

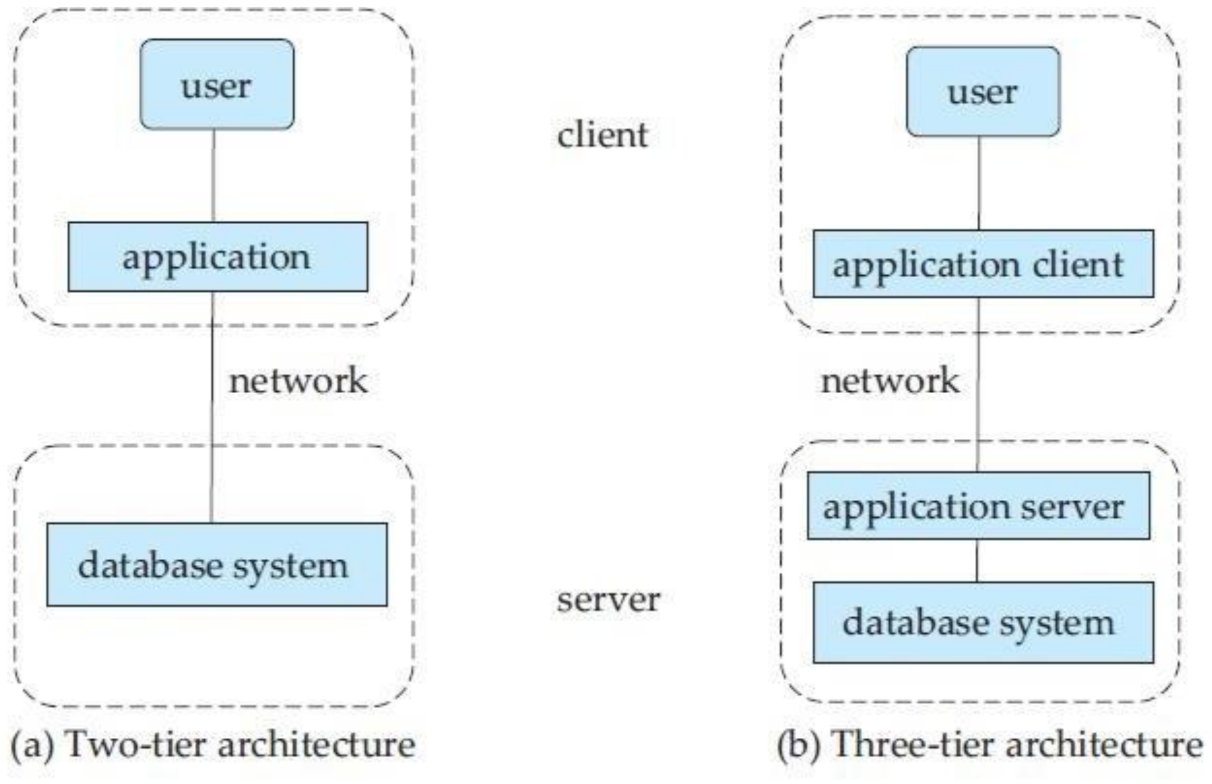


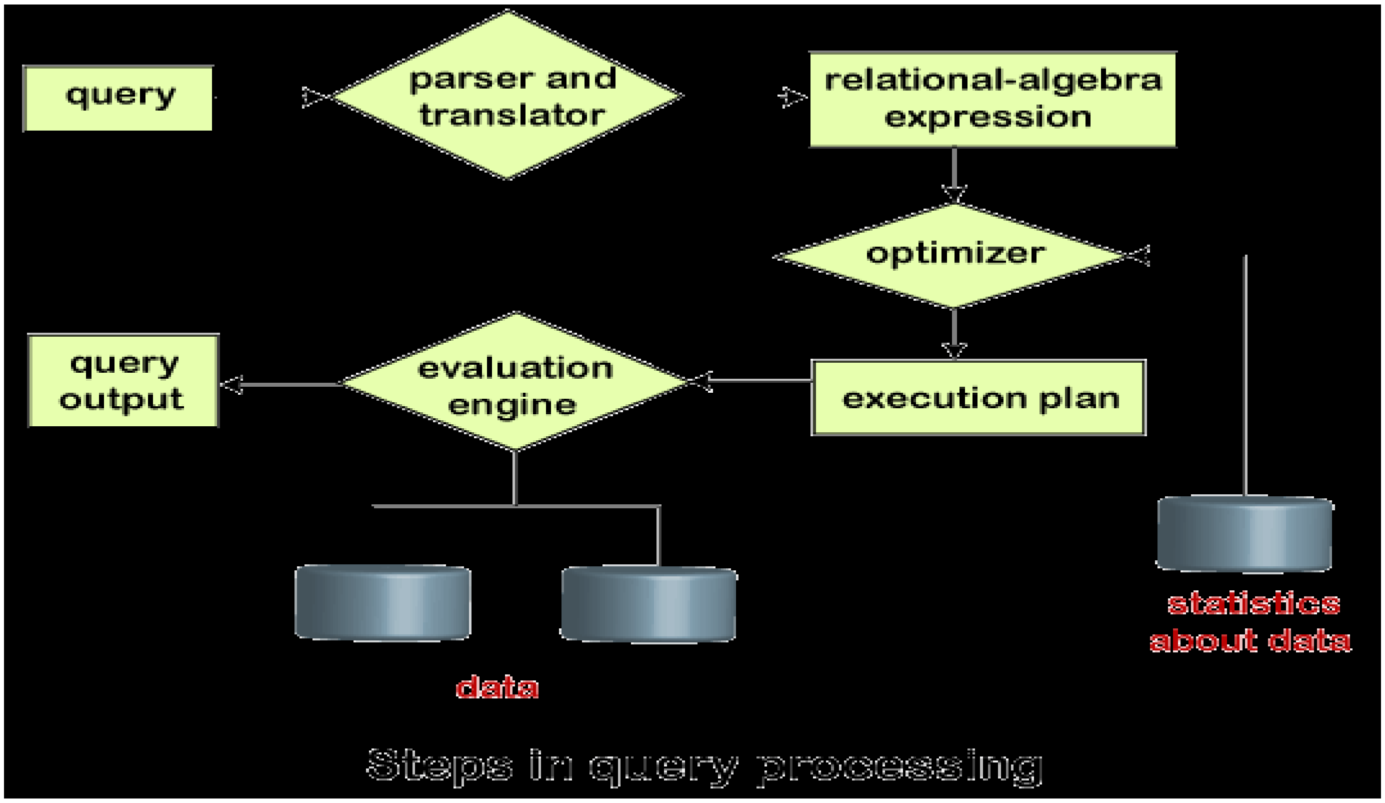
Figure 1.4: Two-tier and three-tier architectures.

Query processor:

The query processor in a database management system receives as input a query request in the form of SQL text, parses it, generates an execution plan, and completes the processing by executing the plan and returning the results to the client.

Query Processing is the activity performed in extracting data from the database. In query processing, it takes various steps for fetching the data from the database. The steps involved are:

1. Parsing and translation
2. Optimization
3. Evaluation



What is ER Modeling?

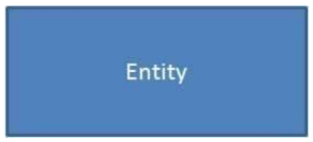
A graphical technique for understanding and organizing the data independent of the actual database implementation

We need to be familiar with the following terms to go further.

Entity

Any thing that has an independent existence and about which we collect data. It is also known as entity type.

In ER modeling, notation for entity is given below.



Entity instance

Entity instance is a particular member of the entity type.

Example for entity instance : A particular employee

Regular Entity

An entity which has its own key attribute is a regular entity.

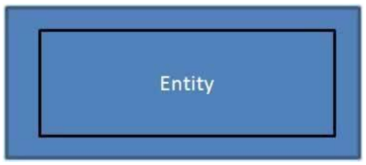
Example for regular entity : Employee.

Weak entity

An entity which depends on other entity for its existence and doesn't have any key attribute of its own is a weak entity.

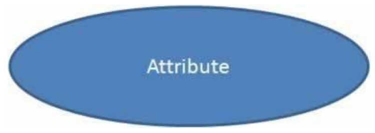
Example for a weak entity : In a parent/child relationship, a parent is considered as a strong entity and the child is a weak entity.

In ER modeling, notation for weak entity is given below.



Attributes

Properties/characteristics which describe entities are called attributes. In
ER modeling, notation for attribute is given below.



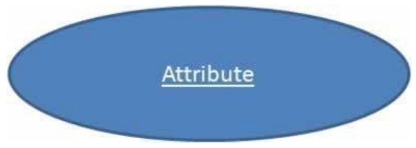
Domain of Attributes

The set of possible values that an attribute can take is called the domain of the attribute. For example, the attribute day may take any value from the set {Monday, Tuesday ... Friday}. Hence this set can be termed as the domain of the attribute day.

Key attribute

The attribute (or combination of attributes) which is unique for every entity instance is called key attribute.

E.g the employee_id of an employee, pan_card_number of a person etc.If the key attribute consists of two or more attributes in combination, it is called a composite key.



In ER modeling, notation for key attribute is given below.

Simple attribute

If an attribute cannot be divided into simpler components, it is a simple attribute.
Example for simple attribute : employee_id of an employee.

Composite attribute

If an attribute can be split into components, it is called a composite attribute.

Example for composite attribute : Name of the employee which can be split into First_name, Middle_name, and Last_name.

Single valued Attributes

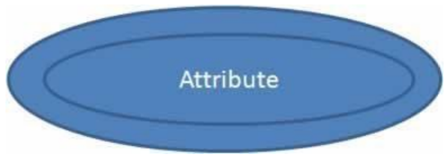
If an attribute can take only a single value for each entity instance, it is a single valued attribute. example for single valued attribute : age of a student. It can take only one value for a particular student.

Multi-valued Attributes

If an attribute can take more than one value for each entity instance, it is a multi-valued attribute. Multi- valued

example for multi valued attribute : telephone number of an employee, a particular employee may have multiple telephone numbers.

In ER modeling, notation for multi-valued attribute is given below.



Stored Attribute

An attribute which need to be stored permanently is a stored attribute

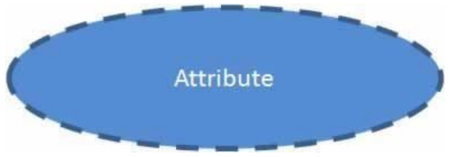
Example for stored attribute : name of a student

Derived Attribute

An attribute which can be calculated or derived based on other attributes is a derived attribute.

Example for derived attribute : age of employee which can be calculated from date of birth and current date.

In ER modeling, notation for derived attribute is given below.



Relationships

Associations between entities are called relationships

Example : An employee works for an organization. Here "works for" is a relation between the entities employee and organization.In ER modeling, notation for relationship is given below.

However in ER Modeling, To connect a weak Entity with others, you should use a weak relationship notation as given below

Degree of a Relationship

Degree of a relationship is the number of entity types involved. The n-ary relationship is the general form for degree n. Special cases are unary, binary, and ternary ,where the degree is 1, 2, and 3, respectively.

Example for unary relationship : An employee ia a manager of another employee Example for binary relationship : An employee works-for department. Example for ternary relationship : customer purchase item from a shop keeper **Cardinality of a Relationship.**

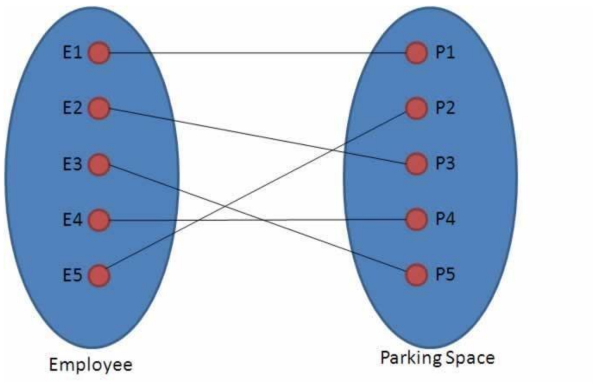
Relationship cardinalities specify how many of each entity type is allowed. Relationships can have four possible connectivities as given below.

1. One to one (1:1) relationship
2. One to many (1:N) relationship
3. Many to one (M:1) relationship
4. Many to many (M:N) relationship

The minimum and maximum values of this connectivity is called the cardinality of the relationship

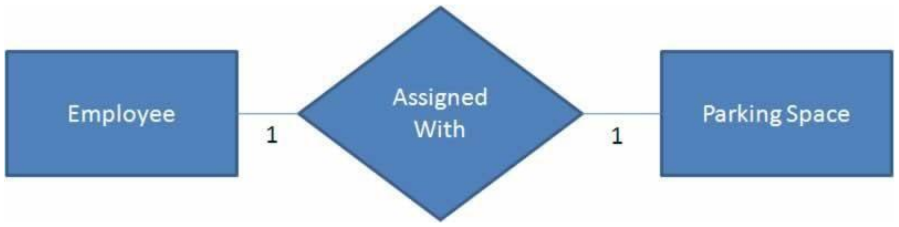
Example for Cardinality – One-to-One (1:1)

Employee is assigned with a parking space.



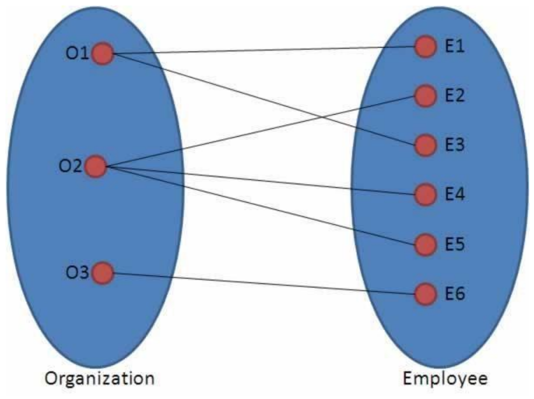
One employee is assigned with only one parking space and one parking space is assigned to only one employee. Hence it is a 1:1 relationship and cardinality is One-To-One (1:1)

In ER modeling, this can be mentioned using notations as given below



Example for Cardinality – One-to-Many (1:N)

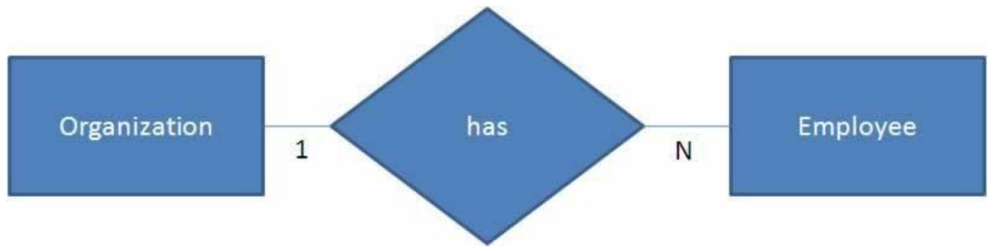
Organization has employees



One organization can have many employees , but one employee works in only one organization.

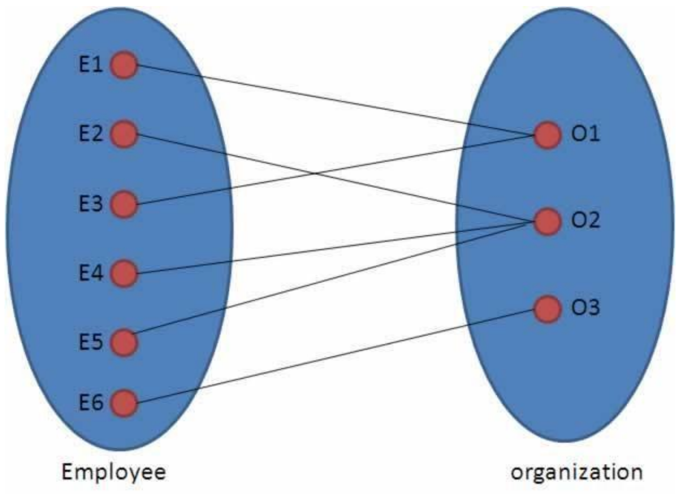
Hence it is a 1:N relationship and cardinality is One-To-Many (1:N)

In ER modeling, this can be mentioned using notations as given below



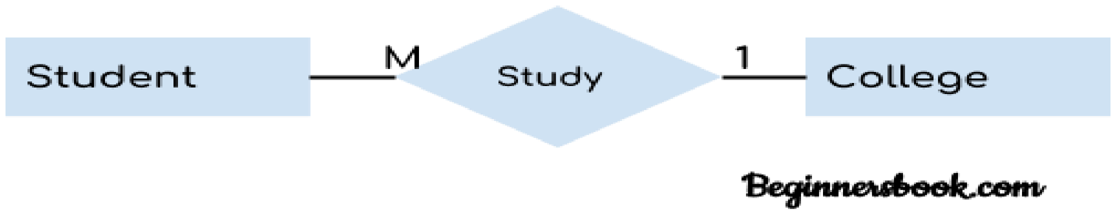
Example for Cardinality – Many-to-One (M :1)

It is the reverse of the One to Many relationship. employee works in organization

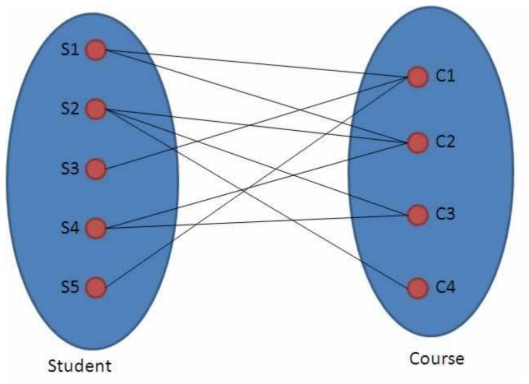


One employee works in only one organization But one organization can have many employees. Hence it is a M:1 relationship and cardinality is Many-to-One (M :1)

In ER modeling, this can be mentioned using notations as given below.



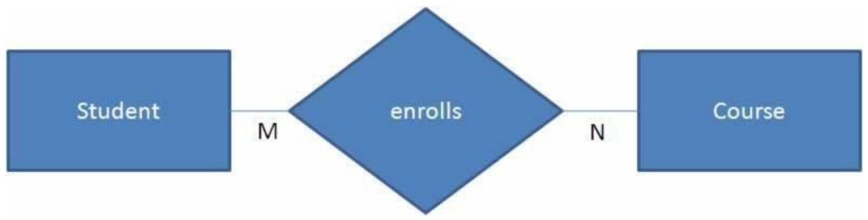
Cardinality – Many-to-Many (M:N)



Students enrolls for courses

One student can enroll for many courses and one course can be enrolled by many students. Hence it is a M:N relationship and cardinality is Many-to-Many (M:N)

In ER modeling, this can be mentioned using notations as given below



Relationship Participation

1. Total

In total participation, every entity instance will be connected through the relationship to another instance of the other participating entity types

2. Partial

Example for relationship participation

Consider the relationship - Employee is head of the department.

Here all employees will not be the head of the department. Only one employee will be the head of the department. In other words, only few instances of employee entity participate in the above relationship. So employee entity's participation is partial in the said relationship.

Advantages and Disadvantages of ER Modeling (Merits and Demerits of ER Modeling)

Advantages

1. ER Modeling is simple and easily understandable. It is represented in business users language and it can be understood by non-technical specialist.
2. Intuitive and helps in Physical Database creation.
3. Can be generalized and specialized based on needs.
4. Can help in database design.
5. Gives a higher level description of the system.

Disadvantages

1. Physical design derived from E-R Model may have some amount of ambiguities or inconsistency.
2. Sometime diagrams may lead to misinterpretations

Relational Model

The relational model is today the primary data model for commercial data processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier data models such as the network model or the hierarchical model.

Structure of Relational Databases:

A relational database consists of a collection of **tables**, each of which is assigned a unique name. For example, consider the *instructor* table of Figure:1.5, which stores information about instructors. The table has four column headers: *ID*, *name*, *dept name*, and *salary*. Each row of this table records information about an instructor, consisting of the instructor’s *ID*, *name*, *dept name*, and *salary*.

Database Schema

When we talk about a database, we must differentiate between the **database schema**, which is the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant in time. The concept of a relation corresponds to the programming- language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

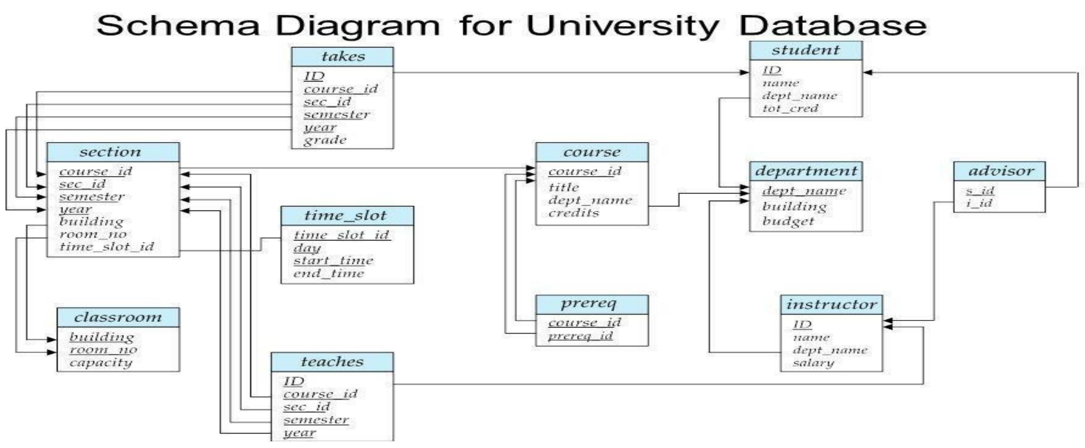
Keys

A **superkey** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey. The *name* attribute of *instructor*, on the other hand, is not a superkey, because several instructors might have the same name.

A superkey may contain extraneous attributes. For example, the combination of *ID* and *name* is a superkey for the relation *instructor*. If *K* is a superkey, then so is any superset of *K*. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **candidate keys**.
It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined. A relation, say *r*1, may include among its attributes the primary key of another relation, say *r*2. This attribute is called a **foreign key** from *r*1, referencing *r*2.

Schema Diagrams

A database schema, along with primary key and foreign key dependencies, can be depicted by
schema diagrams. Figure 1.12 shows the schema diagram for our university organization.



UNIT-2

Relational Algebra and Calculus

RELATIONAL ALGEBRA

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- Select
- Project
- Union
- Set different
- Cartesian product
- Rename

Select Operation (σ)

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}(\text{Books})$

Output − Selects tuples from books where subject is 'database'.

Project Operation (∏)

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1}, A_2, A_n$ (r)

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{\text{subject, author}}$ (Books)

Selects and projects columns named as subject and author from the relation Books.

Union Operation (∪)

It performs binary union between two given relations and is defined as –

$r \cup s = \{ t \mid t \in r \text{ or } t \in s\}$

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{\text{author}} (\text{Books}) \cup \prod_{\text{author}} (\text{Articles})$

Output − Projects the names of the authors who have either written a book or an article or both.

Set Difference (−)

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r − s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{\text{author}} (\text{Books}) - \prod_{\text{author}} (\text{Articles})$

Output − Provides the name of authors who have written books but not articles.

Cartesian Product (X)

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

$r \text{ X } s = \{ q\,t \mid q \in r \text{ and } t \in s\}$

$\sigma_{\text{author = 'tutorialspoint'}}(\text{Books X Articles})$

Output − Yields a relation, which shows all the books and articles written by tutorialspoint.

Rename Operation (ρ)

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** *ρ*.

Notation − $\rho_{x} (E)$

Where the result of expression **E** is saved with name of **x**.

Additional operations are −

- Set intersection
- Assignment
- Natural join

Joins

The *join* operation is one of the most useful operations in relational algebra and is the most commonly used way to combine information from two or more relations. Although a join can be defined as a cross-product followed by selections and projections, joins arise much more frequently in practice than plain cross-products.joins have received a lot of attention, and there are several variants of the join operation.

Condition Joins

The most general version of the join operation accepts a *join condition c* and a pair of relation instances as arguments, and returns a relation instance. The *join condition* is identical to a *selection condition* in form. The operation is defined as follows:

$$R \bowtie_c S = \sigma_c(R \times S)$$

Thus $\bowtie$ is defined to be a cross-product followed by a selection. Note that the condition c can (and typically *does*) refer to attributes of both R and S.

(*sid*)	*sname*	*rating*	*age*	(*sid*)	*bid*	*day*
22	*Dustin*	*7*	*45.0*	58	*103*	*11/12/96*
31	*Lubber*	*8*	*55.5*	58	*103*	*11/12/96*

Figure 4.12 $S1 \bowtie_{S1.sid<R1.sid} R1$

Equijoin

A common special case of the join operation $R \bowtie S$ is when the *join condition* con-sists solely of equalities (connected by $\wedge$) of the form $R.name1 = S.name2$, that is, equalities between two fields in R and S. In this case, obviously, there is some redun-dancy in retaining both attributes in the result.

Natural Join

A further special case of the join operation $R \bowtie S$ is an equijoin in which equalities are specified on *all* fields having the same name in *R* and *S*. In this case, we can simply omit the join condition; the default is that the join condition is a collection of equalities on all common fields.

Division

The division operator is useful for expressing certain kinds of queries, for example: "Find the names of sailors who have reserved all boats." Understanding how to use the basic operators of the algebra to define division is a useful exercise.

RELATIONAL CALCULUS

- Relational calculus is a non-procedural query language. In the non-procedural query language, the user is concerned with the details of how to obtain the end results.
- The relational calculus tells what to do but never explains how to do.

Types of Relational calculus:

1. Tuple Relational Calculus (TRC)

- The tuple relational calculus is specified to select the tuples in a relation. In TRC, filtering variable uses the tuples of a relation.
- The result of the relation can have one or more tuples.

Notation:

{T | P (T)}   or {T | Condition (T)}

Where

T is the resulting tuples

P(T) is the condition used to fetch T.

For example:

1. { T.name | Author(T) AND T.article = 'database' }

OUTPUT: This query selects the tuples from the AUTHOR relation. It returns a tuple with 'name' from Author who has written an article on 'database'.

TRC (tuple relation calculus) can be quantified. In TRC, we can use Existential (∃) and Universal Quantifiers (∀).

For example:

1. { R| ∃T ∈ Authors(T.article='database' AND R.name=T.name)}

Output: This query will yield the same result as the previous one.

2. Domain Relational Calculus (DRC)

- The second form of relation is known as Domain relational calculus. In domain relational calculus, filtering variable uses the domain of attributes.
- Domain relational calculus uses the same operators as tuple calculus. It uses logical connectives ∧ (and), ∨ (or) and ┓ (not).
- It uses Existential (∃) and Universal Quantifiers (∀) to bind the variable.

Notation:

{ a1, a2, a3, ..., an | P (a1, a2, a3, ... ,an)}

Where

a1, a2 are attributes
P stands for formula built by inner attributes

For example:

{< article, page, subject > | ∈ javatpoint ∧ subject = 'database'}

SQL

- SQL stands for Structured Query Language. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

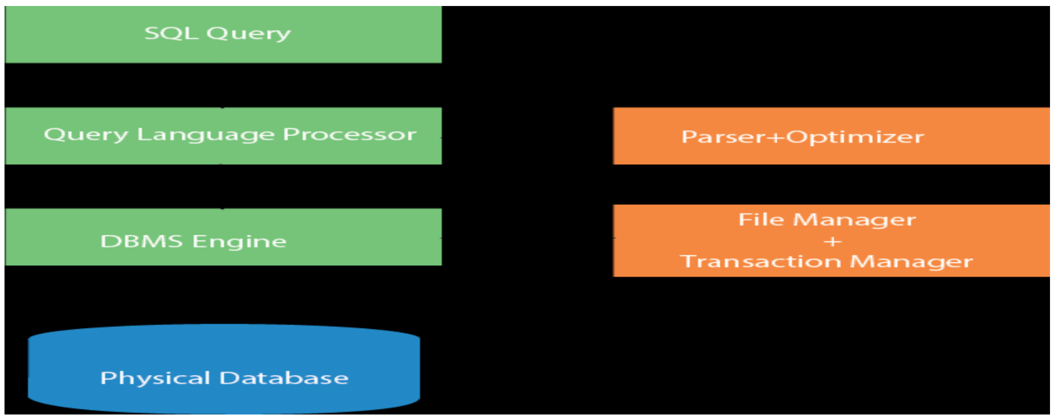
Rules:

SQL follows the following rules:

- Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.
- Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.
- Using the SQL statements, you can perform most of the actions in a database.
- SQL depends on tuple relational calculus and relational algebra.

SQL process:

- When an SQL command is executing for any RDBMS, then the system figure out the best way to carry out the request and the SQL engine determines that how to interpret the task.
- In the process, various components are included. These components can be optimization Engine, Query engine, Query dispatcher, classic, etc.
- All the non-SQL queries are handled by the classic query engine, but SQL query engine won't handle logical files.



1. Basic structure of an SQL expression consists of **select, from** and **where** clauses.
    - **select** clause lists attributes to be copied - corresponds to relational algebra **project**.
    - **from** clause corresponds to Cartesian product - lists relations to be used.
    - **where** clause corresponds to selection predicate in relational algebra.

This is equivalent to the relational algebra expression

- If the where clause is omitted, the predicate *P* is true.
- The list of attributes can be replaced with a * to select all.
- SQL forms the Cartesian product of the relations named, performs a selection using the predicate, then projects the result onto the attributes named.
- The result of an SQL query is a relation.
- SQL may internally convert into more efficient expressions.

Set Operation

The SQL Set operation is used to combine the two or more SQL SELECT statements.

Types of Set Operation

1. Union
2. UnionAll
3. Intersect
4. Minus

1. Union

- The SQL Union operation is used to combine the result of two or more SQL SELECT queries.
- In the union operation, all the number of datatype and columns must be same in both the tables on which UNION operation is being applied.
- The union operation eliminates the duplicate rows from its resultset.

Syntax

```
SELECT column_name FROM table1
UNION
SELECT column_name FROM table2;
```

2. Union All

- Union All operation is equal to the Union operation.
- It returns the set without removing duplication and sorting the data.

Syntax:

```
SELECT column_name FROM table1
UNION ALL
SELECT column_name FROM table2;
```

Example: Using the above First and Second table.

Union All query will be like:

```
SELECT * FROM First
UNION ALL
SELECT * FROM Second;
```

3. Intersect

- It is used to combine two SELECT statements. The Intersect operation returns the common rows from both the SELECT statements.
- In the Intersect operation, the number of datatype and columns must be the same.
- It has no duplicates and it arranges the data in ascending order by default.

Syntax

```
SELECT column_name FROM table1
INTERSECT
SELECT column_name FROM table2;
```

Example:

Using the above First and Second table.

Intersect query will be:

```
SELECT * FROM First
INTERSECT
SELECT * FROM Second;
```

4. Minus

- It combines the result of two SELECT statements. Minus operator is used to display the rows which are present in the first query but absent in the second query.
- It has no duplicates and data arranged in ascending order by default.

Syntax:

```
SELECT column_name FROM table1
MINUS
SELECT column_name FROM table2;
```

Example

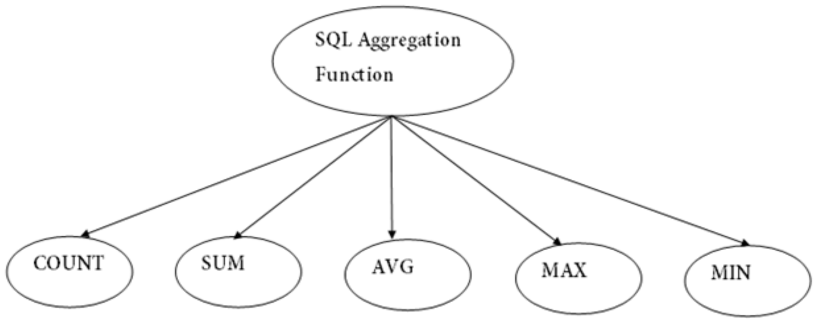
Using the above First and Second table.

Minus query will be:

```
SELECT * FROM First
MINUS
SELECT * FROM Second;
```

Aggregate Functions

- SQL aggregation function is used to perform the calculations on multiple rows of a single column of a table. It returns a single value.
- It is also used to summarize the data.



1. COUNT FUNCTION

- COUNT function is used to Count the number of rows in a database table. It can work on both numeric and non-numeric data types.
- COUNT function uses the COUNT(*) that returns the count of all the rows in a specified table. COUNT(*) considers duplicate and Null.

Syntax

```
COUNT(*)
or
COUNT( [ALL|DISTINCT] expression
)
```

2. SUM Function

Sum function is used to calculate the sum of all selected columns. It works on numeric fields only.

Syntax

```
SUM()
or
SUM( [ALL|DISTINCT] expression )
```

3. AVG function

The AVG function is used to calculate the average value of the numeric type. AVG function returns the average of all non-Null values.

Syntax

```
AVG()
or
AVG( [ALL|DISTINCT] expression )
```

4. MAX Function

MAX function is used to find the maximum value of a certain column. This function determines the largest value of all selected values of a column.

Syntax

```
MAX()
or
MAX( [ALL|DISTINCT] expression )
```

GROUP BY Statement

- The GROUP BY statement groups rows that have the same values into summary rows, like "find the number of customers in each country".
- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

Syntax

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

Examples

The following SQL statement lists the number of customers in each country:

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country;
```

HAVING Clause

- The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Syntax

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

Examples

The following SQL statement lists the number of customers in each country. Only include countries with more than 5 customers:

Example

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
HAVING COUNT(CustomerID) > 5;
```

Nested Queries in SQL

- A subquery can be nested inside other subqueries. SQL has an ability to nest queries within one another. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. SQL executes innermost subquery first, then next level.
- A Subquery or Inner query or a 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 a 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 command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command 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.
- The SELECT list cannot include any references to values that evaluate to a BLOB, ARRAY, CLOB, or NCLOB.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.

Correlated Nested Queries

In the nested queries that we have seen thus far, the inner subquery has been completely independent of the outer query:

(Q1) Find the names of sailors who have reserved boat number 103.

```
SELECT S.sname
FROM    Sailors S
WHERE EXISTS ( SELECT *
               FROM    Reserves R
               WHERE R.bid = 103
                     AND R.sid = S.sid )
```

Set-Comparison Operators

(Q1) Find sailors whose rating is better than some sailor called Horatio.

```
SELECT S.sid
FROM    Sailors S
WHERE S.rating > ANY ( SELECT S2.rating
                       FROM    Sailors S2
                       WHERE S2.sname = 'Horatio' )
```

(Q2) Find the sailors with the highest rating .

```
SELECT S.sid
FROM Sailors S
WHERE        S.rating >= ALL ( SELECT S2.rating FROM
Sailors S2 )
```

More Examples of Nested Queries

(Q1) Find the names of sailors who have reserved both a red and a green boat.

```
SELECT S.sname
FROM    Sailors S, Reserves R, Boats B
WHERE    S.sid = R.sid AND R.bid = B.bid AND B.color = 'red'
        AND S.sid IN ( SELECT S2.sid
                       FROM    Sailors S2, Boats B2, Reserves R2
                       WHERE S2.sid = R2.sid AND R2.bid = B2.bid
                                 AND B2.color = 'green' )
```

*(*Q9) Find the names of sailors who have reserved all boats.

```
SELECT S.sname
FROM    Sailors S
WHERE NOT EXISTS (( SELECT B.bid
                    FROM    Boats B )EXCEPT
                    (SELECT R.bid
                    FROM    Reserves R
                    WHERE R.sid = S.sid ))
```

NULL VALUES

We have assumed that column values in a row are always known. In practice column values can be unknown. For example, when a sailor, say Dan, joins a yacht club, he may not yet have a rating assigned. Since the definition for the Sailors table has a *rating* column, what row should we insert for Dan? What is needed here is a special value that denotes *unknown*.

SQL provides a special column value called *null* to use in such situations. We use *null* when the column value is either *unknown* or *inapplicable*. Using our Sailor table definition, we might enter the row 〈 98, *Dan, null,* 39 〉 to represent Dan. The presence of *null* values complicates many issues, and we consider the impact of *null* values on SQL in this section.

Comparisons Using Null Values

Consider a comparison such as *rating = 8*. If this is applied to the row for Dan, is this condition true or false? Since Dan's rating is unknown, it is reasonable to say that this comparison should evaluate to the value unknown.

SQL also provides a special comparison operator IS NULL to test whether a column value is *null*; for example, we can say *rating* IS NULL, which would evaluate to true on the row representing Dan. We can also say *rating* IS NOT NULL, which would evaluate to false on the row for Dan.

Logical Connectives AND, OR, and NOT

Now, what about boolean expressions such as *rating* = 8 OR *age* < 40 and *rating* = 8 AND *age* < 40? Considering the row for Dan again, because *age* < 40, the first expression evaluates to true regardless of the value of *rating*, but what about the second? We can only say unknown.

INTRODUCTION TO VIEWS

A view is a table whose rows are not explicitly stored in the database but are computed as needed from a view de nition. Consider the Students and Enrolled relations.

```
CREATE VIEW B-Students (name, sid, course)
        AS SELECT S.sname, S.sid, E.cid
           FROM    Students S, Enrolled E
           WHERE S.sid = E.sid AND E.grade = `B'
```

This view can be used just like a base table, or explicitly stored table, in de ning new queries or views.

DESTROYING/ALTERING TABLES AND VIEWS

If we decide that we no longer need a base table and want to destroy it (i.e., delete all the rows *and* remove the table de nition information), we can use the DROP TABLE command. For example, DROP TABLE Students RESTRICT destroys the Students table unless some view or integrity constraint refers to Students; if so, the command fails. If the keyword RESTRICT is replaced by CASCADE, Students is dropped and any referencing views or integrity constraints are (recursively) dropped as well; one of these two keywords must always be speci ed. A view can be dropped using the DROP VIEW command, which is just like DROP TABLE.

ALTER TABLE modifies the structure of an existing table. To add a column called *maiden-name* to Students, for example, we would use the following command:

```
ALTER TABLE Students
      ADD COLUMN maiden-name CHAR(10)
```

TRIGGERS

A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database. A trigger description contains three parts:

1.Event: A change to the database that activates the trigger.

2.Condition: A query or test that is run when the trigger is activated.

3.Action: A procedure that is executed when the trigger is activated and its con-dition is true.

A trigger *action* can examine the answers to the query in the condition part of the trigger, refer to old and new values of tuples modified by the statement activating the trigger, execute new queries, and make changes to the database.

Examples of Triggers in SQL

The examples shown below, written using Oracle 7 Server syntax for defining triggers, illustrate the basic concepts behind triggers. The trigger called *init count* initializes a counter variable before every execution of an INSERT statement that adds tuples to the Students relation. The trigger called *incr count* increments the counter for each inserted tuple that satisfies the condition *age* < 18.

```
CREATE TRIGGER init count BEFORE INSERT ON Students /* Event */
    DECLARE
         count INTEGER;
    BEGIN
         count := 0;                                    /* Action */
    END
```

```
CREATE TRIGGER incr count AFTER INSERT ON Students /* Event */ WHEN
    (new.age < 18)            /* Condition; 'new' is just-inserted tuple */ FOR
    EACH ROW
    BEGIN /* Action; a procedure in Oracle's PL/SQL syntax */ count := count + 1;
    END
```

(identifying the modified table, Students, and the kind of modifying statement, an INSERT), and the third field is the number of inserted Students tuples with *age* < 18. (The above trigger only computes the count; an additional trigger is required to insert the appropriate tuple into the statistics table.)

```
CREATE TRIGGER set count AFTER INSERT ON Students /*     Event      */
REFERENCING NEW TABLE AS InsertedTuples
FOR EACH STATEMENT
        INSERT                                          /* Action */
        INTO  StatisticsTable(ModifiedTable,   ModificationType,   Count)
        SELECT 'Students', 'Insert', COUNT * FROM InsertedTuples I WHERE I.age < 18
```

UNIT-III

DECOMPOSITION

The process of breaking up or dividing a single relation into two or more sub relations is called as decomposition of a relation.

- Lossless Decomposition
- If the information is not lost from the relation that is decomposed, then the decomposition will be lossless.
- The lossless decomposition guarantees that the join of relations will result in the same relation as it was decomposed.
- The relation is said to be lossless decomposition if natural joins of all the decomposition give the original relation.
  Example:
  Let's take 'E' is the Relational Schema, With instance 'e'; is decomposed into: E1, E2, E3, ... En; With instance: e1, e2, e3, ... en, If e1 ⋈ e2 ⋈ e3 . . . . ⋈ en, then it is called as **'Lossless Join Decomposition'.**
- **Dependency Preservation**
- Dependency is an important constraint on the database.
- Every dependency must be satisfied by at least one decomposed table.
- If {A → B} holds, then two sets are functional dependent. And, it becomes more useful for checking the dependency easily if both sets in a same relation.
- This decomposition property can only be done by maintaining the functional dependency.
- In this property, it allows to check the updates without computing the natural join of the database structure.

NORMALIZATION AND TYPES OF NORMALIZATION

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.
- The normal form is used to reduce redundancy from the database table.

Types of normal forms:

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)
- Fourth Normal Form (4NF)
- Fifth Normal Form (5NF)

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.
- Example:
- Sample Employee table, it displays employees are working with multiple departments.

Employee	**Age**	**Department**
Melvin	32	Marketing, Sales
Edward	45	Quality Assurance
Alex	36	Human Resource

Employee table following 1NF:

Employee	**Age**	**Department**
Melvin	32	Marketing
Melvin	32	Sales
Edward	45	Quality Assurance
Alex	36	Human Resource

FUNCTIONAL DEPENDENCY

Functional Dependency (FD) determines the relation of one attribute to another attribute in a database management system (DBMS) system. Functional dependency helps you to maintain the quality of data in the database. A functional dependency is denoted by an arrow →. The functional dependency of X on Y is represented by X → Y. Functional Dependency plays a vital role to find the difference between good and bad database design.

Functional dependency can be written as:

Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id.

Types of Functional Dependencies

- Multivalued dependency:
- Trivial functional dependency**:**
- Non-trivial functional dependency**:**
- Transitive dependency:

➢ **Multivalued dependency:** Multivalued dependency occurs when there are more than one **independent** multivalued attributes in a table.

- **Trivial Functional dependency:**

  - The Trivial dependency is a set of attributes which are called a trivial if the set of attributes are included in that attribute.

  So, X -> Y is a trivial functional dependency if Y is a subset of X.

  - Trivial functional dependency
    - A → B has trivial functional dependency if B is a subset of A.
    - The following dependencies are also trivial like: A → A, B → B

Example:
Consider a table with two columns Employee_Id and Employee_Name.

{Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as

Employee_Id is a subset of {Employee_Id, Employee_Name}.

Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivial dependencies t oo.

- Non-trivial functional dependency
- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

ID → Name,

Name → DOB

SECOND NORMAL FORM (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Or

If a relation second normal form, a relation must be in first normal form and relation must not contain any partial dependency.

A relation is in 2NF if it has **No Partial Dependency,** i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

Partial Dependency – If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.

Example 1 – Consider table-3 as following below.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

{Note that, there are many courses having the same course fee. }

- Here,

  COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;

  COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;

  COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;

  Hence,

  COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;

  But, COURSE_NO -> COURSE_FEE , i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,

we need to split the table into two tables such as :

Table 1: STUD_NO, COURSE_NO

Table 2: COURSE_NO, COURSE_FEE

Table 1		**Table 2**	
STUD_NO	COURSE_NO	COURSE_NO	COURSE_FEE
1	C1	C1	1000
2	C2	C2	1500
1	C4	C3	1000
4	C3	C4	2000
4	C1	C5	2000

Third Normal form (3NF):

A table design is said to be in 3NF if both the following conditions hold:

- Table must be in 2NF
- Transitive functional dependency of non-prime attribute on any super key should be removed.
- An attribute that is not part of any candidate key is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a super key of table
- Y is a prime attribute of table

An attribute that is a part of one of the candidate keys is known as prime attribute.

Example: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

Super keys: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on

Candidate Keys: {emp_id}

Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys. Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

Employee table:

emp_id	emp_name	emp_zip
1001	John	282005

1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

Employee_zip table:

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Dependency-Preserving Decomposition

Consider the Contracts relation with attributes $CSJDPQV$ from Section 15.4.1. The given FDs are C ! $CSJDPQV$, JP ! C, and SD ! P. Because SD is not a key the dependency SD ! P causes a violation of BCNF.

Let R be a relation schema that is decomposed into two schemas with attribute sets X and Y, and let F be a set of FDs over R. The projection of F on X is the set of + FDs in the closure F (not just F !) that involve only attributes in X. We will denote + the projection of F on attributes X as F_X . Note that a dependency U ! V in $F$is in F_X only if *all* the attributes in U and V are in X.

The decomposition of relation schema R with FDs F into schemas with attribute sets X and Y is dependency-preserving if $(F_X [F_Y)^+ = F^+$. That is, if we take the dependencies in F_X and F_Y and compute the closure of their union, we get back all dependencies in the closure of F. Therefore, we need to enforce only the dependencies in F_X and F_Y ; all FDs in F^+ are then sure to be satisfied. To enforce F_X , we need to examine only relation X (on inserts to that relation). To enforce F_Y , we need to examine only relation Y.

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

EMP_ID → EMP_COUNTRY

EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India

264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

EMP_ID $\rightarrow$ EMP_COUNTRY
EMP_DEPT $\rightarrow$ {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

How to find the highest normal form of a relation

Steps to find the highest normal form of a relation:

1. Find all possible candidate keys of the relation.
2. Divide all attributes into two categories: prime attributes and non-prime attributes.

3. Check for 1st normal form then 2nd and so on. If it fails to satisfy nth normal form condition, highest normal form will be n-1.

Example 1. Find the highest normal form of a relation R(A,B,C,D,E) with FD set {A->D, B->A, BC->D, AC->BE}

Step 1. As we can see, $(AC)^+$ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A can be derived from B, so we can replace A in AC by B. So BC will also be a candidate key. So there will be two candidate keys {AC, BC}.

Step 2. Prime attribute are those attribute which are part of candidate key {A,B,C} in this example and others will be non-prime {D,E} in this example.

Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is not in 2nd Normal form because A->D is partial dependency (A which is subset of candidate key AC is determining non-prime attribute D) and 2nd normal form does not allow partial dependency.

So the highest normal form will be 1st Normal Form.

Example 2. Find the highest normal form of a relation **R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}**

Step 1. As we can see, $(AC)^+$ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.

Step 2. Prime attribute are those attribute which are part of candidate key {A,C} in this example and others will be non-prime {B,D,E} in this example.

Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not proper subset of candidate key AC) and AC->BE is in 2nd normal form (AC is candidate key) and B->E is in 2nd normal form (B is not a proper subset of candidate key AC).

The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B->E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be prime attribute.

So the highest normal form of relation will be 2nd Normal form.

Example 3. Find the highest normal form of a relation **R(A,B,C,D,E)** with **FD set {B->A, A->C, BC->D, AC->BE}**

Step 1. As we can see, $(B)^+$ ={B,A,C,D,E}, so B will be candidate key. B can be derived from AC using AC->B (Decomposing AC->BE to AC->B and AC->E). So AC will be super key but $(C)^+$ ={C} and $(A)^+$ ={A,C,B,E,D}. So A (subset of AC) will be candidate key. So there will be two candidate keys {A,B}.

Step 2. Prime attribute are those attribute which are part of candidate key {A,B} in this example and others will be non-prime {C,D,E} in this example.

Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.

The relation is in 2nd normal form because B->A is in 2nd normal form (B is a super key) and A->C is in 2nd normal form (A is super key) and BC->D is in 2nd normal form (BC is a super key) and AC->BE is in 2nd normal form (AC is a super key).

The relation is in 3rd normal form because LHS of all FD's are super keys. The relation is in BCNF as all LHS of all FD's are super keys. So the highest normal form is BCNF.

Multivalued Dependency

- Multivalued dependency occurs when two attributes in a table are independent of each other but, both depend on a third attribute.
- A multivalued dependency consists of at least two attributes that are dependent on a third attribute that's why it always requires at least three attributes.

Example: Suppose there is a bike manufacturer company which produces two colors(white and black) of each model every year.

BIKE_MODEL	**MANUF_YEAR**	**COLOR**
M2011	2008	White
M2001	2008	Black
M3001	2013	White
M3001	2013	Black
M4006	2017	White

M4006	2017	Black

Here columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other.

In this case, these two columns can be called as multivalued dependent on BIKE_MODEL. The representation of these dependencies is shown below:

BIKE_MODEL → → MANUF_YEAR
BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR".

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Join Dependency

- Join decomposition is a further generalization of Multivalued dependencies.
- If the join of R1 and R2 over C is equal to relation R, then we can say that a join dependency (JD) exists.

- Where R1 and R2 are the decompositions R1(A, B, C) and R2(C, D) of a given relations R (A, B, C, D).
- Alternatively, R1 and R2 are a lossless decomposition of R.
- A JD ⋈ {R1, R2,..., Rn} is said to hold over a relation R if R1, R2,....., Rn is a lossless-join decomposition.
- The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R.
- Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

UNIT-4

Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

Example: Suppose an employee of bank transfers Rs 800 from X's account to Y's account. This small transaction contains several low-level tasks:

X's Account

```
Open_Account(X)
Old_Balance = X.balance
New_Balance = Old_Balance - 800
X.balance = New_Balance
Close_Account(X)
```

Y's Account

```
Open_Account(Y)
Old_Balance = Y.balance
New_Balance = Old_Balance + 800
Y.balance = New_Balance
Close_Account(Y)
```

Operations of Transaction:

Following are the main operations of transaction:

Read(X): Read operation is used to read the value of X from the database and stores it in a buffer in main memory.

Write(X): Write operation is used to write the value back to the database from the buffer.

Let's take an example to debit transaction from an account which consists of following operations:

```
R(X);
X = X - 500;
W(X);
```

Let's assume the value of X before starting of the transaction is 4000.

- The first operation reads X's value from database and stores it in a buffer.
- The second operation will decrease the value of X by 500. So buffer will contain 3500.
- The third operation will write the buffer's value to the database. So X's final value will be 3500.

But it may be possible that because of the failure of hardware, software or power, etc. that transaction may fail before finished all the operations in the set.

For example: If in the above transaction, the debit transaction fails after executing operation 2 then X's value will remain 4000 in the database which is not acceptable by the bank.

To solve this problem, we have two important operations:

Commit: It is used to save the work done permanently.

Rollback: It is used to undo the work done.

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability

1.Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:= A-100
Write(A)
	Read(B)
	Y:= Y+100
	Write(B)

- After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.
- If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

2.Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

Total before T occurs = 600+300=900

Total after T occurs=   500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

3.Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

4.Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transaction

In a database, the transaction can be in one of the following states -

1.Active state

- The active state is the first state of every transaction. In this state, the transaction is being executed.
- For example: Insertion or deletion or updating a record is done here. But all the records are still not saved to the database.

2.Partially committed

- In the partially committed state, a transaction executes its final operation, but the data is still not saved to the database.
- In the total mark calculation example, a final display of the total marks step is executed in this state.

3.Committed

- A transaction is said to be in a committed state if it executes all its operations successfully.
- In this state, all the effects are now permanently saved on the database system.

4.Failed state

- If any of the checks made by the database recovery system fails, then the transaction is said to be in the failed state.
- In the example of total mark calculation, if the database is not able to fire a query to fetch the marks, then the transaction will fail to execute.

5.Aborted

- If any of the checks fail and the transaction has reached a failed state then the database recovery system will make sure that the database is in its previous consistent state. If not then it will abort or roll back the transaction to bring the database into a consistent state.

- If the transaction fails in the middle of the transaction then before executing the transaction, all the executed transactions are rolled back to its consistent state.
- After aborting the transaction, the database recovery module will select one of the two operations:
    1. Re-start the transaction
    2. Kill the transaction

Concurrent Execution in DBMS

- In a multi-user system, multiple users can access and use the same database at one time, which is known as the concurrent execution of the database. It means that the same database is executed simultaneously on a multi-user system by different users.
- While working on the database transactions, there occurs the requirement of using the database by multiple users for performing different operations, and in that case, concurrent execution of the database is performed.
- The thing is that the simultaneous execution that is performed should be done in an interleaved manner, and no operation should affect the other executing operations, thus maintaining the consistency of the database. Thus, on making the concurrent execution of the transaction operations, there occur several challenging problems that need to be solved.

Serializability

- When multiple transactions are running concurrently then there is a possibility that the database may be left in an inconsistent state.
- Serializability is a concept that helps us to check which schedules are serializable.
- A serializable schedule is the one that always leaves the database in consistent state.

What is a Serializable Schedule?

- A serializable schedule always leaves the database in consistent state.
- A serial schedule is always a serializable schedule because in serial schedule, a transaction only starts when the other transaction finished execution.
- However a non-serial schedule needs to be checked for Serializability.
- A non-serial schedule of n number of transactions is said to be serializable schedule, if it is equivalent to the serial schedule of those n transactions.
- A serial schedule doesn’t allow concurrency, only one transaction executes at a time and the other starts when the already running transaction finished.

Types of Serializability

1. Conflict Serializability
2. View Serializability

Recoverability in DBMS

A transaction may not execute completely due to hardware failure, system crash or software issues. In that case, we have to roll back the failed transaction. But some other transaction may also have used values produced by the failed transaction. So we have to roll back those transactions as well.

Recoverable Schedules:

- Schedules in which transactions commit only after all transactions whose changes they read commit are called recoverable schedules. In other words, if some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.

  Example 1:
- S1: R1(x), **W1(x)**, R2(x), R1(y), R2(y),

  W2(x), W1(y), **C1**, **C2**;

  Given schedule follows order of **Ti->Tj => C1->C2**. Transaction T1 is executed before T2 hence there is no chances of conflict occur. R1(x) appears before W1(x) and transaction T1 is committed before T2 i.e. completion of first transaction performed first update on data item x, hence given schedule is recoverable.

Testing of Serializability

- Serialization Graph is used to test the Serializability of a schedule.

Example

Assume a schedule S. For S, we construct a graph known as precedence graph. This graph has a pair G = (V, E), where V consists a set of vertices, and E consists a set of edges. The set of vertices is used to contain all the transactions participating in the schedule. The set of edges is used to contain all edges Ti ->Tj for which one of the three conditions holds:

1. Create a node Ti → Tj if Ti executes write (Q) before Tj executes read (Q).
2. Create a node Ti → Tj if Ti executes read (Q) before Tj executes write (Q).
3. Create a node Ti → Tj if Ti executes write (Q) before Tj executes write (Q).

- If a precedence graph contains a single edge Ti → Tj, then all the instructions of Ti are executed before the first instruction of Tj is executed.

- If a precedence graph for schedule S contains a cycle, then S is non-serializable. If the precedence graph has no cycle, then S is known as serializable.

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

There are four types of lock protocols available:

1. Simplistic lock protocol

- It is the simplest way of locking the data while transaction.
- Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it.
- It will unlock the data item after completing the transaction.

2. Pre-claiming Lock Protocol

- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.

3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.

There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.

Timestamp Ordering Protocol

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a **Read (X)** operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a **Write(X)** operation:

- If TS(Ti) < R_TS(X) then the operation is rejected.
- If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where,

TS(TI) denotes the timestamp of the transaction Ti.

R_TS(X) denotes the Read time-stamp of data-item X.

W_TS(X) denotes the Write time-stamp of data-item X.

Validation Based Protocol

Validation phase is also known as optimistic concurrency control technique. In the validation based protocol, the transaction is executed in the following three phases:

1. **Read phase:** In this phase, the transaction T is read and executed. It is used to read the value of various data items and stores them in temporary local variables. It can perform all the write operations on temporary variables without an update to the actual database.
2. **Validation phase:** In this phase, the temporary variable value will be validated against the actual data to see if it violates the serializability.
3. **Write phase:** If the validation of the transaction is validated, then the temporary results are written to the database or system otherwise the transaction is rolled back.

Here each phase has the following different timestamps:

Start(Ti): It contains the time when Ti started its execution.

Validation (T_i): It contains the time when Ti finishes its read phase and starts its validation phase.

Finish(Ti): It contains the time when Ti finishes its write phase.

- This protocol is used to determine the time stamp for the transaction for serialization using the time stamp of the validation phase, as it is the actual phase which determines if the transaction will commit or rollback.
- Hence TS(T) = validation(T).
- The serializability is determined during the validation process. It can't be decided in advance.
- While executing the transaction, it ensures a greater degree of concurrency and also less number of conflicts.

Multiple Granularity

Let's start by understanding the meaning of granularity.

Granularity: It is the size of data item allowed to lock.

Multiple Granularity:

- It can be defined as hierarchically breaking up the database into blocks which can be locked.
- The Multiple Granularity protocol enhances concurrency and reduces lock overhead.
- It maintains the track of what to lock and how to lock.
- It makes easy to decide either to lock a data item or to unlock a data item. This type of hierarchy can be graphically represented as a tree.

For example: Consider a tree which has four levels of nodes.

- The first level or higher level shows the entire database.
- The second level represents a node of type area. The higher level database consists of exactly these areas.
- The area consists of children nodes which are known as files. No file can be present in more than one area.
- Finally, each file contains child nodes known as records. The file has exactly those records that are its child nodes. No records represent in more than one file.
- Hence, the levels of the tree starting from the top level are as follows:
  1. Database
  2. Area
  3. File
  4. Record

UNIT-V

Recovery and Atomicity

When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items. Transactions are made of various operations, which are atomic in nature. But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.

When a DBMS recovers from a crash, it should maintain the following −

- It should check the states of all the transactions, which were being executed.
- A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
- It should check whether the transaction can be completed now or it needs to be rolled back.
- No transactions would be allowed to leave the DBMS in an inconsistent state.

There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction −

- Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
- Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Log-based Recovery

Log is a sequence of records, which maintains the records of actions performed by a transaction. It is important that the logs are written prior to the actual modification and stored on a stable storage media, which is failsafe.

Log-based recovery works as follows −

- The log file is kept on a stable storage media.
- When a transaction enters the system and starts execution, it writes a log about it.

<T_n, Start>

- When the transaction modifies an item X, it write logs as follows −

<T_n, X, V_1, V_2>

It reads T_n has changed the value of X, from V_1 to V_2.

- When the transaction finishes, it logs −

<T_n, commit>

The database can be modified using two approaches −

- **Deferred database modification** − All logs are written on to the stable storage and the database is updated when a transaction commits.
- **Immediate database modification** − Each log follows an actual database modification. That is, the database is modified immediately after every operation.

Recovery with Concurrent Transactions

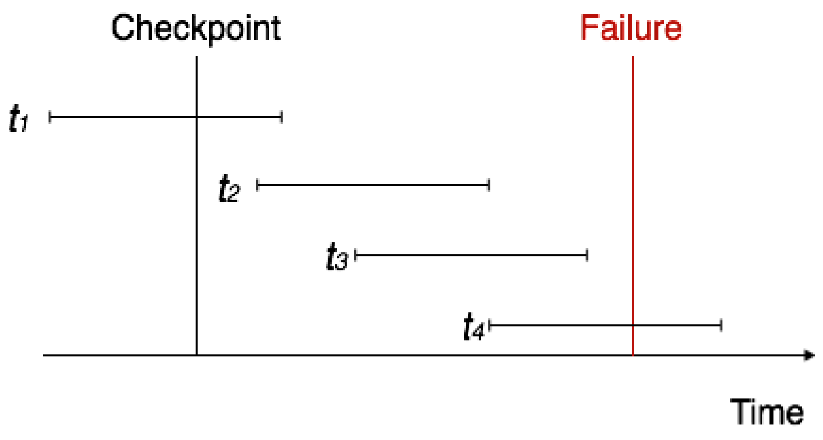
When more than one transaction are being executed in parallel, the logs are interleaved. At the time of recovery, it would become hard for the recovery system to backtrack all logs, and then start recovering. To ease this situation, most modern DBMS use the concept of 'checkpoints'.

Checkpoint

Keeping and maintaining logs in real time and in real environment may fill out all the memory space available in the system. As time passes, the log file may grow too big to be handled at all. Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Recovery

When a system with concurrent transactions crashes and recovers, it behaves in the following manner −



- The recovery system reads the logs backwards from the end to the last checkpoint.
- It maintains two lists, an undo-list and a redo-list.

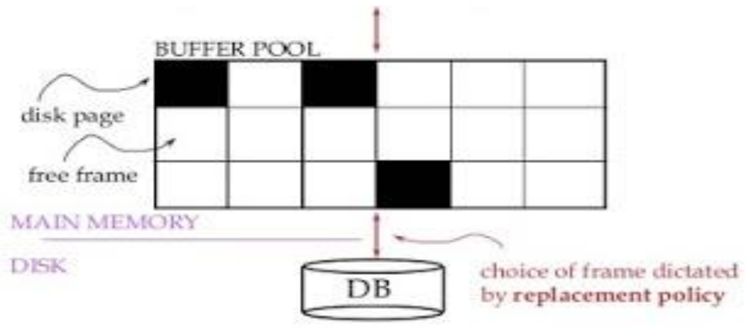
- If the recovery system sees a log with <T_n, Start> and <T_n, Commit> or just <T_n, Commit>, it puts the transaction in the redo-list.
- If the recovery system sees a log with <T_n, Start> but no commit or abort log found, it puts the transaction in undo-list.

All the transactions in the undo-list are then undone and their logs are removed. All the transactions in the redo-list and their previous logs are removed and then redone before saving their logs.

BUFFER MANAGEMENT

The **buffer manager** is the software layer that is responsible for bringing pages from physical disk to main memory as needed. The buffer manages the available main memory by dividing the main memory into a collection of pages, which we called as **buffer pool.** The main memory pages in the buffer pool are called **frames.**

Page Requests from Higher Level



Buffer Management in a DBMS

- Data must be in RAM for DBMS to operate on it!
- Buffer manager hides the fact that not all data is in RAM.

Buffer Manager

- A Buffer Manager is responsible for allocating space to the buffer in order to store data into the buffer.
- If a user request a particular block and the block is available in the buffer, the buffer manager provides the block address in the main memory.
- If the block is not available in the buffer, the buffer manager allocates the block in the buffer.
- If free space is not available, it throws out some existing blocks from the buffer to allocate the required space for the new block.
- The blocks which are thrown are written back to the disk only if they are recently modified when writing on the disk.

- If the user requests such thrown-out blocks, the buffer manager reads the requested block from the disk to the buffer and then passes the address of the requested block to the user in the main memory.
- However, the internal actions of the buffer manager are not visible to the programs that may create any problem in disk-block requests. The buffer manager is just like a virtual machine.

Failure with Loss of Nonvolatile Storage

Until now, we have considered only the case where a failure results in the loss of information residing in volatile storage while the content of the nonvolatile storage remains intact. Although failures in which the content of nonvolatile storage is lost are rare, we nevertheless need to be prepared to deal with this type of failure. In this section, we discuss only disk storage. Our discussions apply as well to other nonvolatile storage types.

The basic scheme is to **dump** the entire content of the database to stable storage periodically—say, once per day. For example, we may dump the database to one or more magnetic tapes. If a failure occurs that results in the loss of physical database blocks, the system uses the most recent dump in restoring the database to a previous consistent state. Once this restoration has been accomplished, the system uses the log to bring the database system to the most recent consistent state.

More precisely, no transaction may be active during the dump procedure, and a procedure similar to checkpointing must take place:

1. Output all log records currently residing in main memory onto stable storage.

2. Output all buffer blocks onto the disk.

3. Copy the contents of the database to stable storage.

4. Output a log record <dump> onto the stable storage.

Database Recovery Techniques in DBMS

Database systems, like any other computer system, are subject to failures but the data stored in it must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.
There are both automatic and non-automatic ways for both, backing up of data and recovery from any failure situations. The techniques used to recover the lost data due to system crash, transaction errors, viruses, catastrophic failure, incorrect commands execution etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred update and immediate update or backing up data can be used.

Recovery techniques are heavily dependent upon the existence of a special file known as a **system log**. It contains information about the start and end of each transaction and any updates which occur in the **transaction**. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

- The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.
- read_item(T, X): This log entry records that transaction T reads the value of database item X.
- write_item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- commit(T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- abort(T): This records that transaction T has been aborted.
- checkpoint: Checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in consistent state, and all the transactions were committed.

Algorithm for Recovery and Isolation Exploiting Semantics (ARIES)

Algorithm for Recovery and Isolation Exploiting Semantics (ARIES) is based on the Write Ahead Log (WAL) protocol. Every update operation writes a log record which is one of the following :

1. **Undo-only log record:**

   Only the before image is logged. Thus, an undo operation can be done to retrieve the old data.
2. **Redo-only log record:**

   Only the after image is logged. Thus, a redo operation can be attempted.
3. **Undo-redo log record:**

   Both before images and after images are logged.

In it, every log record is assigned a unique and monotonically increasing log sequence number (LSN). Every data page has a page LSN field that is set to the LSN of the log record corresponding to the last update on the page. WAL requires that the log record corresponding to an update make it to stable storage before the data page corresponding to that update is written to disk. For performance reasons, each log write is not immediately forced to disk. A log tail is maintained in main memory to buffer log writes. The log tail is flushed to disk when it gets full. A transaction cannot be declared committed until the commit log record makes it to disk.

Once in a while the recovery subsystem writes a checkpoint record to the log. The checkpoint record contains the transaction table and the dirty page table. A master log record is maintained separately, in stable storage, to store the LSN of the latest checkpoint record that made it to disk. On restart, the recovery subsystem reads the master log record to find the checkpoint's LSN, reads the checkpoint record, and starts recovery from there on.

The recovery process actually consists of 3 phases:

1. **Analysis:**
   The recovery subsystem determines the earliest log record from which the next pass must start. It also scans the log forward from the checkpoint record to construct a snapshot of what the system looked like at the instant of the crash.
2. **Redo:**
   Starting at the earliest LSN, the log is read forward and each update redone.
3. **Undo:**
   The log is scanned backward and updates corresponding to loser transactions are undone.

Remote backup definition and strategies.

Remote backup as a part of data protection in companies has a vital importance since it guarantees the organization its operational continuity and functionality in the occurrence of loss or damage to critical data that may affect them, such as:

- Loss of the DB of company personnel
- Corruption of the DB and / or application that control automated production processes
- Loss of historical organizational data, while the company is in the process of auditing or evaluating quality standards
- Loss of information on the balance sheets and management indicators of the different organizational processes
- Theft of a computer or device, even by the people who handle confidential management information

File Organization

- The **File** is a collection of records. Using the primary key, we can access the records. The type and frequency of access can be determined by the type of file organization which was used for a given set of records.
- File organization is a logical relationship among various records. This method defines how file records are mapped onto disk blocks.
- File organization is used to describe the way in which the records are stored in terms of blocks, and the blocks are placed on the storage medium.
- The first approach to map the database to the file is to use the several files and store only one fixed length record in any given file. An alternative approach is to structure our files so that we can contain multiple lengths for records.
- Files of fixed length records are easier to implement than the files of variable length records.

Objective of file organization

- It contains an optimal selection of records, i.e., records can be selected as fast as possible.
- To perform insert, delete or update transaction on the records should be quick and easy.
- The duplicate records cannot be induced as a result of insert, update or delete.

Types of file organization:

File organization contains various methods. These particular methods have pros and cons on the basis of access or selection. In the file organization, the programmer decides the best-suited file organization method according to his requirement.

Types of file organization are as follows:

- Sequential file organization
- Heap file organization
- Hash file organization
- B+ file organization
- Indexed sequential access method (ISAM)
- Cluster file organization

- For the minimal cost of storage, records should be stored efficiently.

What is Indexing?

Indexing is a data structure technique which allows you to quickly retrieve records from a database file. An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.

An index -

- Takes a search key as input
- Efficiently returns a collection of matching records.

Types of Indexing

Indexing in Database is defined based on its indexing attributes. Two main types of indexing methods are:

- Primary Indexing

- Secondary Indexing

1.Primary Indexing

Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.

The primary Indexing in DBMS is also further divided into two types.

- Dense Index
- Sparse Index

(i) Dense Index

  - In a dense index, a record is created for every search key valued in the database.
  - This helps you to search faster but needs more space to store index records.
  - In this Indexing, method records contain search key value and points to the real record on the disk.

(ii) Sparse Index

- It is an index record that appears for only some of the values in the file.
- Sparse Index helps you to resolve the issues of dense Indexing in DBMS.
- In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.
- However, sparse Index stores index records for only some search-key values.
- It needs less space, less maintenance overhead for insertion, and deletions but It is slower compared to the dense Index for locating records.

Below is an database index Example of Sparse Index

2.Secondary Index

- The secondary Index in DBMS can be generated by a field which has a unique value for each record, and it should be a candidate key. It is also known as a non-clustering index.
- This two-level database indexing technique is used to reduce the mapping size of the first level. For the first level, a large range of numbers is selected because of this; the mapping size always remains small.

Example of secondary Indexing

Let's understand secondary indexing with a database index example:

- In a bank account database, data is stored sequentially by acc_no; you may want to find all accounts in of a specific branch of ABC bank.
- Here, you can have a secondary index in DBMS for every search-key. Index record is a record point to a bucket that contains pointers to all the records with their specific search-key value.

Clustering Index

In a clustered index, records themselves are stored in the Index and not pointers. Sometimes the Index is created on non-primary key columns which might not be unique for each record. In such a situation, you can group two or more columns to get the unique values and create an index which is called clustered Index. This also helps you to identify the record faster.

Example:

Let's assume that a company recruited many employees in various departments. In this case, clustering indexing in DBMS should be created for all employees who belong to the same dept.

It is considered in a single cluster, and index points point to the cluster as a whole. Here, Department _no is a non-unique key.

What is Multilevel Index?

Multilevel Indexing in Database is created when a primary index does not fit in memory. In this type of indexing method, you can reduce the number of disk accesses to short any record and kept on a disk as a sequential file and create a sparse base on that file.

B-Tree Index

B-tree index is the widely used data structures for tree based indexing in DBMS. It is a multilevel format of tree based indexing in DBMS technique which has balanced binary search trees. All leaf nodes of the B tree signify actual data pointers.

Moreover, all leaf nodes are interlinked with a link list, which allows a B tree to support both random and sequential access.

- Lead nodes must have between 2 and 4 values.

- Every path from the root to leaf are mostly on an equal length.
- Non-leaf nodes apart from the root node have between 3 and 5 children nodes.
- Every node which is not a root or a leaf has between n/2] and n children.

Advantages of Indexing

Important pros/ advantage of Indexing are:

- It helps you to reduce the total number of I/O operations needed to retrieve that data, so you don't need to access a row in the database from an index structure.
- Offers Faster search and retrieval of data to users.
- Indexing also helps you to reduce tablespace as you don't need to link to a row in a table, as there is no need to store the ROWID in the Index. Thus you will able to reduce the tablespace.
- You can't sort data in the lead nodes as the value of the primary key classifies it.

Disadvantages of Indexing

Important drawbacks/cons of Indexing are:

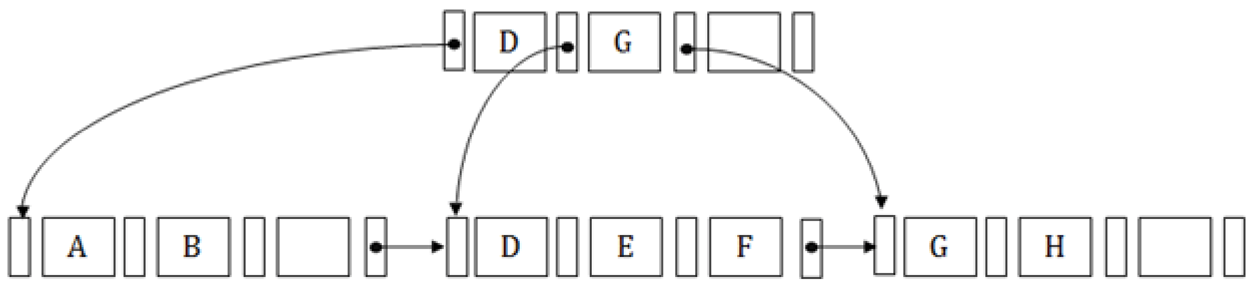
- To perform the indexing database management system, you need a primary key on the table with a unique value.
- You can't perform any other indexes in Database on the Indexed data.
- You are not allowed to partition an index-organized table.
- SQL Indexing Decrease performance in INSERT, DELETE, and UPDATE query.

B+ Tree

- The B+ tree is a balanced binary search tree. It follows a multi-level index format.
- In the B+ tree, leaf nodes denote actual data pointers. B+ tree ensures that all leaf nodes remain at the same height.
- In the B+ tree, the leaf nodes are linked using a link list. Therefore, a B+ tree can support random access as well as sequential access.

Structure of B+ Tree

- In the B+ tree, every leaf node is at equal distance from the root node. The B+ tree is of the order n where n is fixed for every B+ tree.
- It contains an internal node and leaf node.



Internal node

- An internal node of the B+ tree can contain at least n/2 record pointers except the root node.
- At most, an internal node of the tree contains n pointers.

Leaf node

- The leaf node of the B+ tree can contain at least n/2 record pointers and n/2 key values.
- At most, a leaf node contains n record pointer and n key values.
- Every leaf node of the B+ tree contains one block pointer P to point to next leaf node.

Searching a record in B+ Tree

Suppose we have to search 55 in the below B+ tree structure. First, we will fetch for the intermediary node which will direct to the leaf node that can contain a record for 55.

So, in the intermediary node, we will find a branch between 50 and 75 nodes. Then at the end, we will be redirected to the third leaf node. Here DBMS will perform a sequential search to find 55.

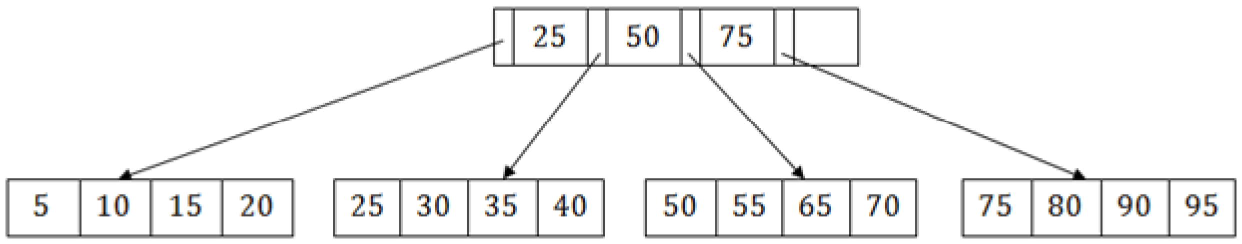
B+ Tree Insertion

- Suppose we want to insert a record 60 in the below structure. It will go to the 3rd leaf node after 55. It is a balanced tree, and a leaf node of this tree is already full, so we cannot insert 60 there.
- In this case, we have to split the leaf node, so that it can be inserted into tree without affecting the fill factor, balance and order.
- The 3rd leaf node has the values (50, 55, 60, 65, 70) and its current root node is 50. We will split the leaf node of the tree in the middle so that its balance is not altered. So we can group (50, 55) and (60, 65, 70) into 2 leaf nodes.
- If these two has to be leaf nodes, the intermediate node cannot branch from 50. It should have 60 added to it, and then we can have pointers to a new leaf node.
- This is how we can insert an entry when there is overflow. In a normal scenario, it is very easy to find the node where it fits and then place it in that leaf node.

B+ Tree Deletion

Suppose we want to delete 60 from the above example. In this case, we have to remove 60 from the intermediate node as well as from the 4th leaf node too. If we remove it from the intermediate node, then the tree will not satisfy the rule of the B+ tree. So we need to modify it to have a balanced tree.

After deleting node 60 from above B+ tree and re-arranging the nodes, it will show as follows:



Query Optimization in Relational Algebra

Query: A query is a request for information from a database.

Query Plans: A query plan (or query execution plan) is an ordered set of steps used to access data in a SQL relational database management system.

Query Optimization: A single query can be executed through different algorithms or re-written in different forms and structures. Hence, the question of query optimization comes into the picture – Which of these forms or pathways is the most optimal? The query optimizer attempts to determine the most efficient way to execute a given query by considering the possible query plans.

Importance: The goal of query optimization is to reduce the system resources required to fulfill a query, and ultimately provide the user with the correct result set faster.

- First, it provides the user with faster results, which makes the application seem faster to the user.
- Secondly, it allows the system to service more queries in the same amount of time, because each request takes less time than unoptimized queries.
- Thirdly, query optimization ultimately reduces the amount of wear on the hardware (e.g. disk drives), and allows the server to run more efficiently (e.g. lower power consumption, less memory usage).