18CSC303J – DATABASE MANAGEMENT SYSTEMS
End Semester Notes

UNIT – 1

Database Management Systems (DBMS):

- Database Management Systems (DBMS) are software systems used to store, retrieve, and run queries on data which is stored in a database.
- A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.
- DBMS manage the data, the database engine, and the database schema, allowing for data to be manipulated or extracted by users and other programs.
- This helps provide data security, data integrity, concurrency, and uniform data administration procedures.

Uses of DBMS:

- To develop software applications in less time.
- Data independence and efficient use of data.
- For uniform data administration.
- For data integrity and security.
- For concurrent access to data, and data recovery from crashes.
- To use user-friendly declarative query language.

Applications of DBMS:

Domain	Usage of DBMS
Banking	Managing customer information, account activities, payments, deposits, loans, etc.
Transportation	Maintain and Manage the Passenger Manifesto, reservations and schedule information.
Universities	Student information, course registrations, colleges, and grades.
Telecommunication	It helps to keep call records, monthly bills, maintaining balances, etc.
Finance	For storing information about stock, sales, and purchases of financial instruments like stocks and bonds.
Sales	To store customer details, product details & sales information.
Manufacturing	It is used for the management of supply chain and for tracking production of items. Inventories status in warehouses.
Social Media	Manage the user accounts, security, data access.

The purpose of a DBMS is to convert raw data into information,
then information to knowledge, then knowledge to action.

File System	DBMS
The file system is software that manages and organizes the files in a storage medium within a computer.	DBMS is software for managing the database.
Redundant data.	No redundant data.
It doesn't provide backup and recovery of data.	It provides backup and recovery of data.
No efficient query processing.	Efficient query processing.
Less data consistency.	More data consistency.
Less complex.	More complex.
Provides less security.	Provides more security.
Less expensive.	Very expensive.
No data independence.	Data independence exists.
Only one user can access data at a time.	Multiple users can access data at a time.
Sharing data is not easy.	Sharing data is easy.
It gives details of storage and representation of data.	It hides the internal details of database.
Integrity constraints are difficult to implement.	Integrity constraints are easy to implement.

Data Abstraction:

To make the easy access of data by the users, the complications are kept hidden and the remaining part of the database is accessible to the them through data abstraction.

Physical level:

Describes how a record (e.g., student) is stored.

Logical level:

Describes data stored in database, and the relationships among the data.

```
type student = record
     student_reg_number : integer;
     student_name : string;
     student_degree : string;
     customer_mobile : integer;
     student_email : string
end;
```

View level:

Application programs hide details of data types. Views can also hide information (such as a student's mobile number /email) for security purposes.

Instances and Schemas:

Instance: The actual content of the database at a particular point in time.

Schema: The logical structure of the database.

- Physical Schema: Datafiles, Control file, Redo log files, Tablespaces, Data Blocks, Segments, Extents.
- Logical Schema: Tables, Views, Synonyms, Indexes, Clusters, Sequences.

Database Users:

Naive Users:

- Those who don't have any knowledge about DBMS.
- They don't have any privileges to modify the database, simply use the application.

Example: Railway booking users.

Application Programmers:

- Users who develop DBMS applications.
- They are backend programmers and use languages such as C++, Java, PHP, Python, etc.

Sophisticated Users:

- Having knowledge about database and DBMS.
- They can create their own applications based on requirements.
- They don't program them, but manage using queries.

Example: Business Analyst.

Native Users:

- These are the users, who use the existing database applications.
- They don't write any codes or queries.

Example: Library Management Systems.

Specialized Users:

- These are also sophisticated users, but they write special database application programs.
- They are the developers who develop the complex programs to the requirement.

Stand-alone Users:

- These users will have a stand-alone database for their personal use.
- These kinds of the database will have readymade database packages which will have menus and graphical interfaces.

Database Administrator (DBA):

- DBA is a person or a group who define and manage the database in all three levels.
- DBA can create / modify /remove the users based on the requirements.
- DBA is the super user having all the privileges of DBMS.

The DBA's responsibilities are as follows:

- Installing the Database
- Upgrading the Database
- Design and Implementation
- Database Tuning and Security
- Migrating the Database
- User Management
- Backup and Recovery

Structured Query Language (SQL):

SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987.

- SQL is a common language for all databases.
- SQL is a fourth generation Language
- SQL is non-procedural.
- SQL is not case sensitive.
- All SQL statements should end with a terminator, the default terminator is semi-colon (;)
- Based on the operation SQL divided into three categories:
    - DDL (Data Definition Language) - To specify the database schema.
    - DML (Data Manipulation Language) - To express the database queries and updates.
    - DCL (Data Control Language) - To manage the database operations.

Data Definition Language (DDL):

DDL consists of the following commands:

- CREATE: Used to create a new object / schema with a defined structure.
  ```
  CREATE TABLE table_name (columnn datatype,.., columnN datatype);
  ```

- ALTER: Alter command used to modify the base table structure.
  ```
  ALTER TABLE table_name ADD / MODIFY column_name datatype;
  ```

- DROP: Used to remove the base table with records from database permanently.
  ```
  DROP TABLE table_name ;
  ```

- TRUNCATE: Used to delete the records from the base table permanently but retains the structure.
  ```
  TRUNCATE TABLE table_name;
  ```

Data Manipulation Language (DML):

DML consists of the following commands:

- INSERT:
    - It relates only with new records.
    - Only one row can be inserted at a time.
    - Multiple rows can be inserted using “&” symbol one by one.
    - Must follow the order of the column specified in the query statement.

  ```
  INSERT INTO <table_name> (column_name_n <datatype>,.., column_name_N
  <datatype>)
  VALUES (value n,.., value N);
  ```

- UPDATE:
    - It works with only existing records.
    - It works only column wise.
    - It is used to modify the column values.

```
UPDATE <table_name> set <field_name> = value [ where <condition>];
```

- DELETE:
  - It works only with existing records.
  - It works only with row wise.
  - It not possible to delete a single column in a row.

```
DELETE from <table_name> [ where <condition>];
```

- SELECT:
  - Works with existing records
  - Works with row wise and column wise
  - Works with multiple tables
  - Never affect / change / update / modification in the data base

```
SELECT column_list FROM table-name
                 [WHERE clause]
                 [GROUP BY clause]
                 [HAVING clause]
                 [ORDER BY clause]
```

NOTE:

- Where clause (Conditional retrieval)
- Order by clause (Retrieval in Ascending or Descending Order)
- Group by clause (Retrieval of distinct values by considering groups)
- Having clause (Followed by Group by clause with COUNT function)

Data Control Language (DCL):

DCL consists of the following commands:

- GRANT: To give access privileges of an object to other users by the owner.

```
GRANT [ ALL / INSERT /UPDATE /DELETE /SELECT]
on <OBJECT_NAME> to <USER_NAME>;
```

- REVOKE: To get back all the privileges from the user who has been granted.

```
REVOKE [ ALL / INSERT /UPDATE /DELETE /SELECT]
on <OBJECT_NAME> from <USER_NAME>;
```

Database System:

The database system is divided into three components:

- Query Processor
- Storage Manager
- Disk Storage

Query Processor:

- It interprets the requests (queries) from user(s) via an application program/interface into instructions.
- It also executes the user request which is received from the DML compiler.

DML Compiler	It processes the DML statements into low level instruction.
DDL Interpreter	It processes the DDL statements into a set of tables containing meta data.
Embedded DML Pre-compiler	It processes DML statements embedded in an application program into procedural calls.
Query Optimizer	It executes the instruction generated by DML Compiler.

Storage Manager:

- It is an interface between the information stored in the database and the requests.
- It is also known as Database Control System ; maintains the consistency and integrity.

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.
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.

Disk Storage:

- Used to store all the information.
- It contains the following components.

Data Files	It stores the data.
Data Dictionary	It contains the information about the structure of any database object. It is the repository of information that governs the metadata.
Indices	It provides faster retrieval of a data item.
Statistical Data	Contains the statistics of all information

Data Independence:

Data Independence is used to achieve the changes in physical level without affecting logical level and vice versa. There are two types of data independence, namely:

Physical Data Independence: It is defined as to make the changes in the structure of the physical level /low level of DBMS without affecting the logical level / view level.

Logical Data Independence: It is defined as to make the changes in the structure of the logical level / view level of DBMS without affecting the physical / low level.

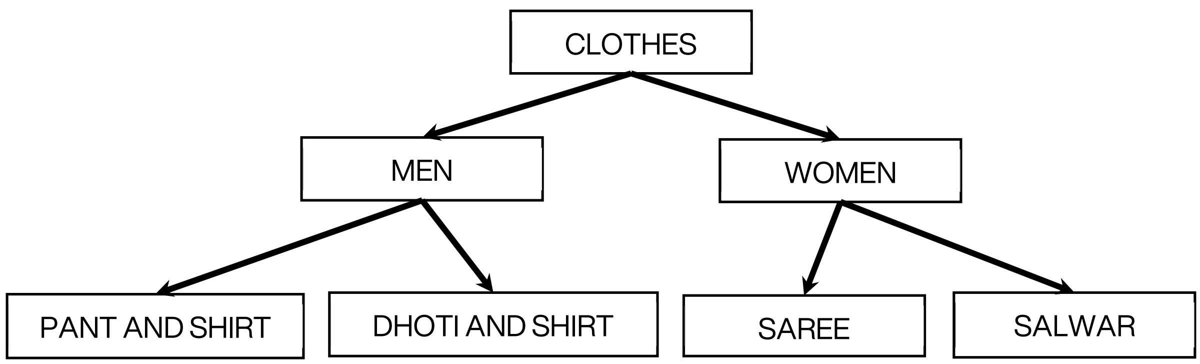
Physical Data Independence	Logical Data Independence
Concerned with the storage of the data.	Concerned with the structure of data definition.
Easy to retrieve.	Difficult to retrieve.
Easy to achieve.	Difficult to achieve.
Concerned with physical schema.	Concerned with logical schema.

Evolution of Data Models:

- Hierarchical Model
- Network Model
- Entity Relationship Model
- Relational Model
- Object Oriented Model

Hierarchical Model:

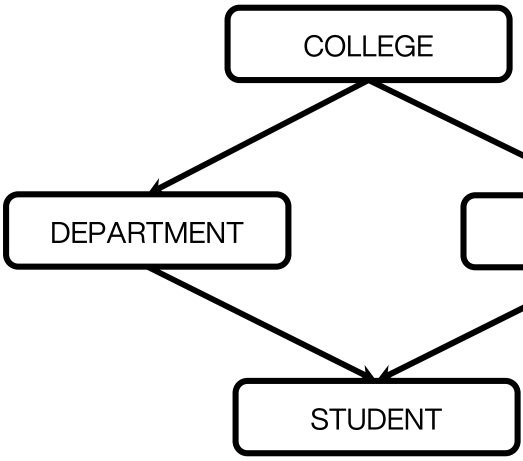
- The first and fore most model of the DBMS.
- This model organizes the data in the hierarchical tree structure.
- This model is easy to understand with real time examples site map of a website.



Advantages	Disadvantages
The design of the hierarchical model is simple.	Implementation is complex.
Data sharing is feasible since the data is stored in a single database.	Maintenance is difficult since changes done in the database may want you to do changes in the entire database structure.

Network Model:

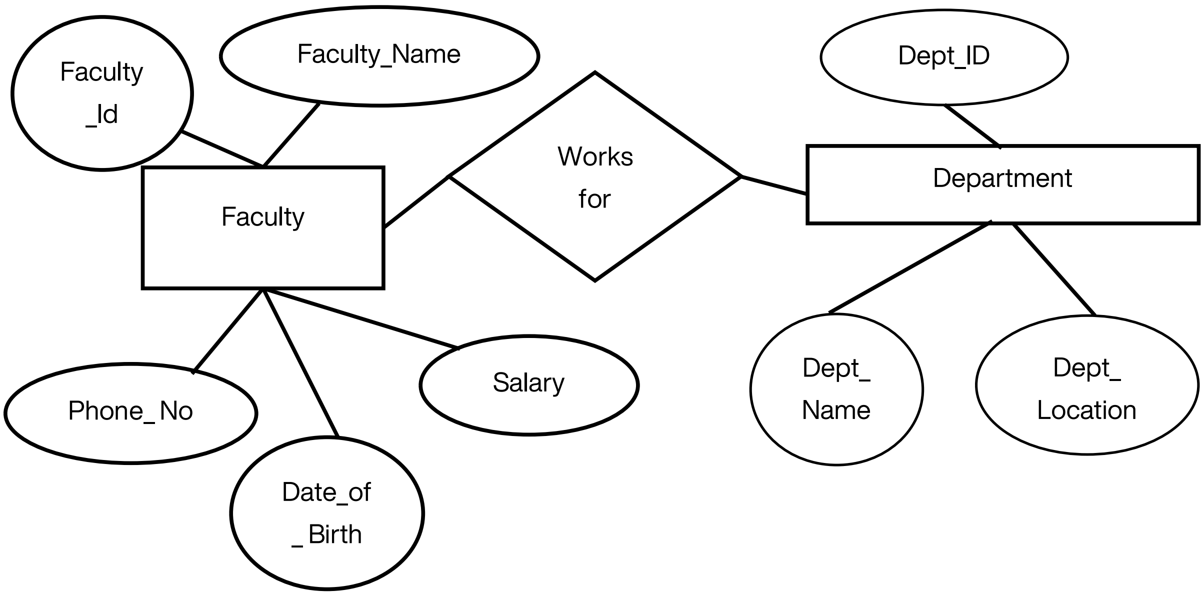
- Network model is an extension of hierarchical model.
- This model was recommended as the best before relationship model.



Advantages	Disadvantages
Easy to design.	Pointers bring complexity since the records are based on pointers and graphs.
It isolates the program from other details.	Changes in the database is not easy that makes it hard to achieve structural independence.

Entity Relationship Model:

- This model is a high-level data model.
- Represents the real – world problem as a pictorial representation.
- Easy to understand by the developers about the specification.



Advantages	Disadvantages
Very Simple.	No industry standard.
Better communication.	Hidden information.

Relational Model:

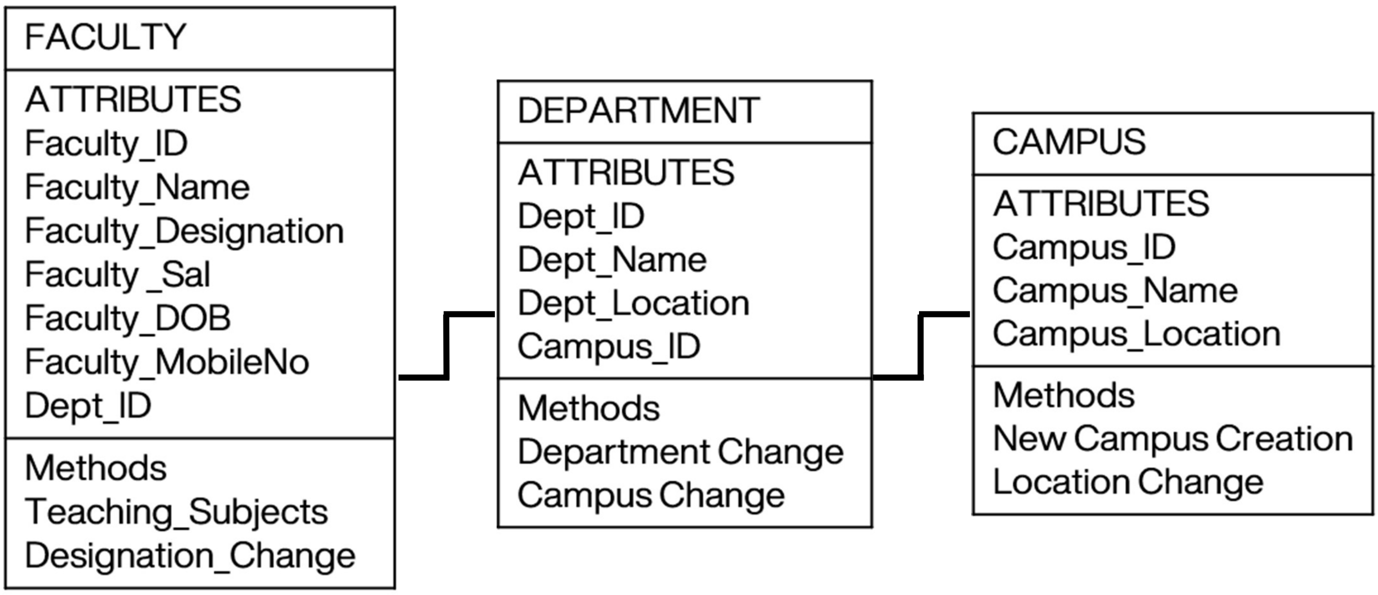
- Widely used model
- Data are represented as row-wise and column-wise (2-Dimensional Array)

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-80	800	-	20
7499	ALLEN	SALESMAN	7698	20-FEB-81	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-81	1250	500	30
7566	JONES	MANAGER	7839	02-APR-81	2975	-	20
7654	MARTIN	SALESMAN	7698	28-SEP-81	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-81	2850	-	30
7782	CLARK	MANAGER	7839	09-JUN-81	2450	-	10
7788	SCOTT	ANALYST	7566	09-DEC-82	3000	-	20
7839	KING	PRESIDENT	-	17-NOV-81	5000	-	10

Advantages	Disadvantages
Simple and scalable.	Hardware overheads.
Structured format.	Have to be updated repetitively.

Object Oriented Model:

- The real- time problems are easily represented through an object.
- In this model, the data and its relationship present in the single structure.
- Complex data like images, audio, videos can be stored easily.



Advantages	Disadvantages
Complex data sets can be saved and retrieved quickly and easily.	Object databases are not widely adopted.
Object IDs are assigned automatically.	In some situations, the high complexity can cause performance problems.

UNIT – 2

Entity Relationship Diagram & its Terminologies:

- An Entity-Relationship (ER) diagram is a visual representation of the relationships among entities in a database.
- It depicts the structure of a database system by showing the entities, their attributes, and the relationships between them.
- ER diagrams are commonly used in database design to model and understand the relationships between different entities in a system.

In an ER diagram, entities are represented as rectangles, attributes are represented as ovals or ellipses, and relationships are represented as diamond shapes. Lines or arrows are used to connect entities and represent the relationships between them.

Here is an easy example to illustrate an ER diagram. Let us consider a simple bookstore database.
In this database, we can identify three main entities: "Book," "Author," and "Publisher."

- The "Book" entity would have attributes like "ISBN," "Title," "Genre," and "Price."
- The "Author" entity would have attributes like "AuthorID," "Name," "Birthdate," and "Nationality."
- The "Publisher" entity would have attributes like "PublisherID," "Name," "Location," and "YearFounded."

Now, the relationships between the entities can be:

- A book is written by one or more authors. This is a many-to-many relationship between "Book" and "Author" entities since a book can have multiple authors, and an author can write multiple books. We represent this relationship with a diamond shape connected to both entities.
- A book is published by one publisher, but a publisher can have multiple books. This is a one-to-many relationship between the "Book" and "Publisher" entities. We represent this relationship with a line connecting the "Book" entity to the "Publisher" entity.

The resulting ER diagram would visually represent these entities, attributes, and relationships, allowing us to understand the structure of the bookstore database at a glance.

Terminology	Description
Entity	A real-world object or concept that has attributes (properties) and can be uniquely identified.
Attribute	A property or characteristic of an entity. It describes the specific details or features of an entity.
Relationship	An association or connection between entities that captures how they are related or interact with each other.
Cardinality	It defines the number of instances of one entity that can be associated with the instances of another entity.
Primary Key	A unique identifier that uniquely identifies each instance of an entity.
Foreign Key	An attribute in one entity that refers to the primary key of another entity. It establishes relationships between entities.

Superclass	A higher-level entity that defines common attributes and relationships shared by multiple subclasses.
Subclass	A specialized entity that inherits attributes and relationships from its superclass.
Generalization	The process of defining a superclass and subclasses to represent a hierarchy of entities.
Aggregation	A relationship between entities where one entity represents the whole (aggregate) and others represent its parts.
Composite Attribute	An attribute that is composed of multiple sub-attributes, each representing a separate piece of data.
Derived Attribute	An attribute whose value can be derived or calculated based on other attributes or entities.

Weak Entity Set vs Strong Entity Set:

Weak Entity Set	Strong Entity Set
An entity set that depends on another entity set for its existence. It cannot exist without the relationship with the identifying entity set.	An entity set that can exist independently and does not depend on any other entity set for its existence.
It is identified by a partial key, which includes the primary key of the identifying entity set along with its own attributes.	It is identified by its own unique attributes or primary key.
It cannot exist without a relationship with the identifying entity set.	It can exist independently without any dependency on other entity sets.
It participates in a one-to-many relationship with the identifying entity set (strong entity set).	It can participate in various types of relationships with other entity sets.
Consider an "OrderItem" entity set that represents individual items in an order. It depends on the "Order" entity set for its existence.	Consider a "Customer" entity set that represents individual customers. It can exist independently without any dependency.

Mapping Cardinalities:

Cardinality	Description	Example
(1:1)	Each instance in one entity set is associated with exactly one instance in another entity set.	A person has one unique social security number, and a social security number is assigned to only one person.
(1:M)	Each instance in one entity set can be associated with zero or more instances in another entity set.	A customer can place multiple orders, but each order is associated with only one customer.
(M:1)	Each instance in one entity set is associated with at most one instance in another entity set.	Multiple students can belong to the same class, but a class can have only one teacher.
(M:M)	Each instance in one entity set can be associated with zero or more instances in another entity set, and vice versa.	Students can enroll in multiple courses, and each course can have multiple students.

High Level Data Model:

A high-level data model, also known as a conceptual data model, is an abstraction of the database that focuses on the overall structure, relationships, and key entities without concerning itself with implementation details.

- It provides a broad view of the data requirements and serves as a foundation for database design and development.
- A high-level data model typically represents the entities, their attributes, and the relationships between them in a simplified and intuitive manner.

A high-level data model provides the database designer with a conceptual frame work which includes:

  - What kind of data required by the database users?
  - How the database to be designed to fulfil the requirements?

- Database designer should choose the appropriate data model and translate these requirements into a conceptual schema.
- The schema developed at this conceptual-design phase provides a detailed overview of the enterprise.
- The designer reviews the schema to confirm that all data requirements.
- The designer can review the design to remove the redundant features
- The focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.

Data Model	Description
Entity-Relationship (ER) Model	Represents entities as objects, attributes as properties of the objects, and relationships as associations between objects.
Unified Modelling Language (UML)	A general-purpose modelling language used in software engineering. The class diagram in UML represents classes and their relationships.
Object-Oriented Data Model (OODM)	Based on the concept of objects, representing entities as objects with attributes and methods, and relationships as associations.
Network Data Model	Organizes data using a network structure, with entities connected through links.
Hierarchical Data Model	Organizes data in a tree-like structure with parent-child relationships.

(REFER UNIT – 1 NOTES FOR A BETTER DESCRIPTION)

UNIT – 3

PL/SQL Triggers:

A PL/SQL trigger is a named program unit that is automatically executed in response to a specific event occurring in a database. It is associated with a table, view, schema, or database and is triggered by data manipulation language (DML) statements like INSERT, UPDATE, or DELETE.

Syntax:

```
CREATE OR REPLACE TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF} {INSERT | UPDATE | DELETE}
[OR {INSERT | UPDATE | DELETE}] ON table_name
[FOR EACH ROW]
[DECLARE
  -- Variable declarations ]
BEGIN
  -- Trigger logic
END;
```

Example:

Let us consider a scenario where we have a table called "Employees" with columns "emp_id," "emp_name," and "salary." We want to create a trigger that automatically updates the "last_updated" column with the current timestamp whenever a row in the "Employees" table is updated.

```
CREATE OR REPLACE TRIGGER employees_trigger
BEFORE UPDATE ON Employees
FOR EACH ROW
BEGIN
  :NEW.last_updated := SYSTIMESTAMP;
END;
/
```

Advantages	Disadvantages
Provides automatic execution of code.	Can introduce performance overhead.
Helps enforce data integrity and consistency.	Triggers can be complex to write and maintain.
Allows customization and business logic implementation.	Triggers can impact the application scalability of the given data.
Enables auditing and logging of database events.	Can cause cascading triggers and potential infinite loops.
Facilitates data validation and error handling.	Triggers may introduce dependencies and tight coupling.
Enhances security by controlling access to data.	Trigger logic can be difficult to debug and troubleshoot.
Offers a way to enforce complex business rules.	Triggers may impact transaction management and locking.
Supports event-driven programming paradigm.	Inappropriate use of triggers can lead to unexpected behaviour.

PL/SQL Cursors:

A PL/SQL cursor is a mechanism that allows you to retrieve and manipulate data from a result set returned by a SELECT statement. It provides a way to traverse and process individual rows returned by a query.

Syntax:

```
DECLARE
  cursor_name [parameters] RETURN datatype;
  cursor_variable cursor_name%ROWTYPE;
BEGIN
  OPEN cursor_variable;
  FETCH cursor_variable INTO target_variables;
  -- Processing logic
  CLOSE cursor_variable;
END;
```

Example:

Let's consider a scenario where we want to retrieve the employee names and salaries from the "Employees" table and display them using a cursor.

```
DECLARE CURSOR emp_cursor IS
   SELECT emp_name, salary FROM Employees;
   emp_rec emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;
  LOOP
    FETCH emp_cursor INTO emp_rec;
    EXIT WHEN emp_cursor%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE('Employee Name: ' || emp_rec.emp_name || ', Salary: '
|| emp_rec.salary);
  END LOOP;
  CLOSE emp_cursor;
END;
/
```

Advantages	Disadvantages
Allows processing of multiple rows in a controlled manner.	Can consume more memory and resources compared to other techniques.
Enables data retrieval and manipulation from result sets.	Cursor operations can be slower than set-based operations.
Supports explicit control over fetching, updating, and deleting rows.	Requires careful handling to avoid cursor-related errors like "cursor is closed".
Facilitates sequential access and navigation through query results.	Cursors can introduce complexity and reduce code readability.
Enables transactional control with open, fetch, and close operations.	Cursors may cause performance issues when used inefficiently.

High-Level Query Processing:

High-level Query Processing includes translations on high level queries into low level expressions that can be used at physical level of file system, query optimization and actual execution of query to get the actual result. High-level query processing involves multiple stages and tasks to execute queries efficiently.

Query Parsing:

Syntax Analysis	The query parser checks the query's syntax to ensure it adheres to the grammar rules of the query language (e.g., SQL). It verifies that the query is well-formed and properly structured.
Semantic Analysis	The parser performs semantic analysis to ensure the query's correctness and meaningfulness in terms of table & column names, data types, and other semantic rules.

Query Optimization:

Query Rewriting	The optimizer may transform the original query into an equivalent, but more efficient, representation. It can rewrite the query to exploit certain query patterns or properties that enable more efficient execution.
Cost Estimation	The optimizer estimates the cost of different query execution plans to determine their relative efficiency. It considers factors such as disk I/O, CPU usage, and memory consumption to estimate the execution time for each plan.
Plan Generation	The optimizer generates alternative query execution plans based on optimization techniques. Each plan represents a different approach to retrieve the query result.
Plan Selection	The optimizer evaluates the generated plans based on their estimated costs and selects the plan with the lowest cost as the optimal execution plan. The chosen plan is the one expected to deliver the query result with the least resource consumption.

Query Execution:

Plan Execution	The execution engine follows the steps defined in the chosen execution plan. It retrieves the necessary data from disk or memory, performs operations such as joins, filters, and aggregations, and produces intermediate results.
Data Access	The execution engine interacts with the storage layer to access the required data efficiently. It leverages indexing structures, caching mechanisms, and disk I/O optimizations to minimize data retrieval overhead.
Join Processing	If the query involves joins between multiple tables, the execution engine employs various join algorithms to combine the data from different tables efficiently.
Aggregation and Filtering	The execution engine performs aggregations (e.g., sum, count, average) and applies filters (e.g., WHERE clauses) to narrow down the result set based on the query criteria.
Result Presentation	Finally, the execution engine formats and presents the query result to the user or application in the desired format (e.g., table, JSON, XML).

Throughout the high-level query processing, the focus is on optimizing the query execution plan and minimizing resource usage, such as disk I/O, CPU utilization, and memory consumption. The goal is to achieve efficient query processing and deliver accurate results in a timely manner. This process is critical for maintaining good performance in database systems handling large volumes of data and complex queries.

UNIT - 4

Database Normalization:

- Database Normalization is the technique of organizing data into more than one table in the database.
- It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristic like insertion, updation and deletion anomalies.
- It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Anomalies:

Insertion Anomaly	Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
Deletion Anomaly	The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
Updation Anomaly	The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

Here's an example...

S_id	S_name	S_address	Subject
101	Adam	Noida	Biology
102	Alex	Panipat	Maths
103	Stuart	Jammu	Maths
104	Adam	Noida	Physics

Insertion Anomaly	To update address of student to occurs twice (or) more than twice in a table we have to update address column in all the rows else the data will come inconsistent.
Deletion Anomaly	Suppose for a student new admission, we have s_id, s_name, s_address of a student but the student has not opted for any subject then we have to insert null for the subject which leads to insertion anomalies.
Updation Anomaly	If s_id = 401 has only one subject and temporarily drop it when we delete that row entire student record will be deleted along with it.

Relational Databases:

- A relational database is a type of database that stores and organizes data in a tabular form, with rows representing individual records and columns representing fields or attributes of those records.
- The relationships between tables are defined using keys, which allow data to be linked across different tables.
- Relational databases are widely used in a variety of applications, including financial systems, human resources management, inventory management, and e-commerce platforms. Some examples of popular relational databases include MySQL, Oracle, Microsoft SQL Server, and PostgreSQL.

Data Integrity	Relational databases ensure data integrity by enforcing constraints such as primary keys, unique keys, and foreign keys, which help maintain consistency and accuracy of data.
Scalability	Relational databases are designed to scale horizontally and vertically, making them suitable for use in applications with high data volumes.
Flexibility	Relational databases are highly flexible and can be easily adapted to meet changing business requirements.
Security	Relational databases provide robust security mechanisms such as user authentication, authorization, and encryption to protect sensitive data.
Querying and Reporting	Relational databases provide powerful querying and reporting capabilities, allowing users to retrieve and analyze data in a variety of ways.

Functional Dependency:

- A functional dependency is a constraint that specifies the relationship between two sets of attributes where one set can accurately determine the value of other sets.
- It is denoted as X → Y, where X is a set of attributes that can determine the value of Y.
- The attribute set on the left side of the arrow, X is called Determinant, while on the right side, Y is called the Dependent.

Trivial Functional Dependency:

A 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.

Emp_id	Emp_name
AS555	Harry
AS811	George
AS999	Kevin

{Emp_id, Emp_name} -> Emp_id is a trivial functional dependency as Emp_id is a subset of {Emp_id,Emp_name}.

Non-Trivial Functional Dependency:

A nontrivial dependency occurs when A->B holds true where B is not a subset of A. In a relationship, if attribute B is not a subset of attribute A, then it is considered as a non-trivial dependency.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Apple	Tim Cook	57

(Company} -> {CEO} (if we know the Company, we know the CEO name)
But CEO is not a subset of Company, and hence it's non-trivial functional dependency.

Transitive Dependency:
A transitive dependency is a type of functional dependency which happens when “t” is indirectly formed by two functional dependencies.

Company	CEO	Age
Microsoft	Satya Nadella	51
Google	Sundar Pichai	46
Alibaba	Jack Ma	54

{Company} -> {CEO} (if we know the company, we know its CEO’s name)
{CEO} -> {Age} If we know the CEO, we know the Age.

Multivalued Functional Dependency:
Multivalued dependency occurs in the situation where there are multiple independent multivalued attributes in a single table. A multivalued dependency is a complete constraint between two sets of attributes in a relation.

Car_model	Maf_year	Color
H001	2017	Metallic
H001	2017	Green
H005	2018	Metallic
H005	2018	Blue
H010	2015	Metallic
H033	2012	Gray

maf_year and color are independent of each other but dependent on car_model. In this example, these two columns are said to be multi-value dependent on car_model.
This dependence can be represented like this: car_model -> maf_year , car_model-> colour.

Normal Forms:
Database normalization is the process of organizing data in a relational database to reduce data redundancy and improve data integrity. The normalization process involves breaking down a database into smaller, more manageable tables and defining relationships between them.

First Normal Form (1NF)	This level requires that all data be atomic, meaning that each value in a field should be indivisible.
Second Normal Form (2NF)	In addition to meeting the requirements of 1NF, this level requires that all non-key fields be fully dependent on the primary key.
Third Normal Form (3NF)	In addition to meeting the requirements of 2NF, this level requires that all non-key fields be independent of each other.
Boyce-Codd Normal Form	BCNF is a higher level of normalization than 3NF and applies to tables with multiple candidate keys. A table is in BCNF if every determinant of the table is a candidate key.
Fourth Normal Form (4NF)	4NF is a higher level of normalization than BCNF and applies to tables with multi-valued dependencies.

UNIT - 5

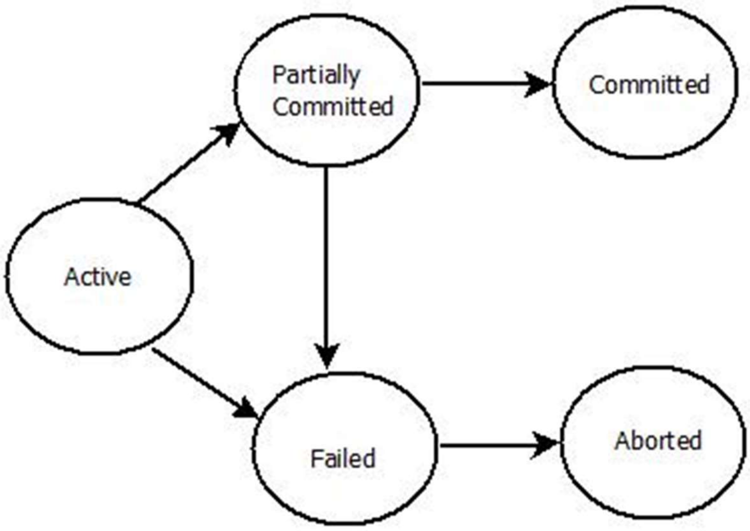
Transaction:

A transaction is a unit of program execution that accesses and possibly updates various data items. To preserve the integrity of data the database system must ensure:

Atomicity	Either all operations of the transaction are properly reflected in the database or none are.
Consistency	Execution of a transaction in isolation preserves the consistency of the database.
Isolation	Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions.
Durability	After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

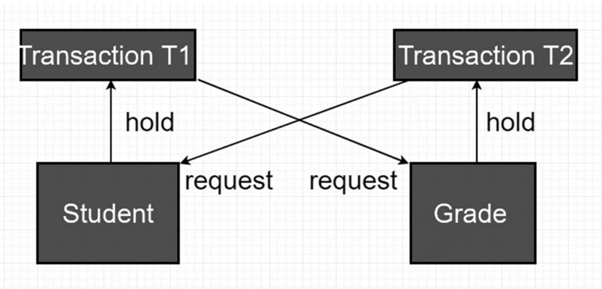
Transition States:

Active	The initial state; the transaction stays in this state while it is executing.
Partially committed	After the final statement has been executed.
Failed	After the discovery that normal execution can no longer proceed.
Aborted	After the transaction has been rolled back and the database restored to its state prior to the start of the transaction. You can either restart or kill the transaction later.
Committed	After successful completion of the transaction.



Deadlock:

- In database management systems (DBMS), a deadlock occurs when two or more transactions are blocked, each waiting for the other to release a resource that it needs in order to proceed. As a result, the transactions are unable to complete and the system becomes stuck, or deadlocked.
- Deadlocks can occur in any multi-user system where concurrent transactions are executed, including DBMS. Deadlocks can be difficult to detect and resolve, and can cause significant performance problems and data integrity issues.
- To prevent deadlocks, most DBMS use locking mechanisms to control access to shared resources. However, improper use of locking or insufficient concurrency control can lead to deadlocks.



Deadlock Avoidance	When a database is stuck in a deadlock, it is always better to avoid the deadlock rather than restarting or aborting the database. The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.
Deadlock Detection	When a transaction waits indefinitely to obtain a lock, The DBMS should detect whether the transaction is involved in a deadlock or not.
Deadlock Prevention	A deadlock can be prevented if the resources are allocated in such a way that a deadlock never occurs. The DBMS analyses the operations whether they can create a deadlock situation or not, if they do, that transaction is never allowed to be executed.

WAIT – DIE	WOUND -WAIT
It is based on a non-preemptive technique.	It is based on a preemptive technique.
In this, older transactions must wait for the younger one to release its data items.	In this, older transactions never wait for younger transactions.
The number of aborts and rollbacks is higher in these techniques.	In this, the number of aborts and rollback is lesser.

Features of a deadlock:

Mutual Exclusion	Each resource can be held by only one transaction at a time, and other transactions must wait for it to be released.
No Preemption	Resources cannot be taken away from a transaction forcibly, and the transaction must release them voluntarily.
Circular Wait	Transactions are waiting for resources in a circular chain, where each transaction is waiting for a resource held by the next transaction in the chain.
Indefinite Blocking	Transactions are blocked indefinitely, waiting for resources to become available, and no transaction can proceed.
Inconsistent Data	Deadlock can lead to inconsistent data if transactions are unable to complete and leave the database in an intermediate state.

Log-Based Recovery:

- Log-based recovery is a technique used a DBMS to ensure that data remains consistent even in the event of a system failure or crash.
- The basic idea behind log-based recovery is to use a log file to record all changes to the database, and then use this log file to reconstruct the database to a consistent state after a failure.
- In log-based recovery, all changes made to the database are first recorded in a log file, which is typically stored on disk separately from the database itself.
- The log file contains a record of all updates, inserts, and deletes made to the database, along with information about the transaction that made the change, such as the start time and the commit time.
- When a failure occurs, the DBMS can use the log file to recover the database to a consistent state.
- The recovery process involves rolling back any incomplete transactions that were in progress at the time of the failure, and then rolling forward any completed transactions that were recorded in the log file but not yet written to the database.
- Log-based recovery is an important technique for ensuring data consistency and durability in DBMS. By recording all changes to the database in a log file, DBMS can recover from failures and ensure that the database remains in a consistent state, even in the event of a system crash or other type of failure.