

Final Examination Preparation for Database Management System

UNIT 1: Introduction to DBMS

1. Database-System Introduction

- **Database System** is defined as a collection of application programs that interact with the **database** along with the DBMS and **database** itself.
- **Data**: the raw information or information processed by a computer. This information may be in the form of text documents, images, audio clips, software programs, or other types of data. Computer data may be processed by the computer's [CPU](#) and is stored in [files](#) and [folders](#) on the computer's [hard disk](#).
- Form of Data
 - Type: 1,2, 3, a, b, c, “abs”.
 - Format: text, image, audio, video, etc.
- **Database**: is a collection of related data, collections of tables.
- **DBMS (Database Management System)**: is a software system for creating and managing databases. The DBMS essentially serves as an interface between databases and end users or application programs, ensuring that data is consistently organized and remains easily accessible.
- **Data Warehouse and Mining**: store in one place, to analyze, OLAP, the collection of database and Mining is use to control over Database.
 - OLTP (Online transaction processing): the information system mainly used to 5 handle the transactional data (huge transactional data). Ex. ODOO, KIT point system.
 - OLAP (Online Analyze processing): the information system used for analytical purpose, used to perform complex query and find out issues and for business purposes. It can be the cube or data warehouse kind of system. Ex. A2A KPI dashboard.

2. DB Applications

Databases touch all the aspects of our lives

- Banking: all transactions
- Airlines: reservations, schedules
- Universities: registration, grade.
- Sales: customers, products, purchases
- Manufacturing: productions, inventory, orders, supply chain.
- Human resources: employee records, salaries, tax deductions.

3. Purpose and Advantages of Database Systems

- Database applications were built on top of the file system.
- Drawbacks of using file systems to store data
 - **Data redundancy and inconsistency**: manipulate file formats, duplication of information in different files

- *Difficulty in accessing data*: need to write a new program to carry out each task.
- *Data isolation*: manipulate files and formats
- *Integrity problems*: data information change constraints.
- *Atomicity of updates*: inconsistent state with partial updates from failures.
- *Concurrent access by multiple users*: inconsistencies uncontrolled concurrent access.
- *Security problems*.
- According to all those mentioned problems above, DBMS was created to offer all the solutions.

4. *View or Level of Database Abstraction*

Levels of Database abstraction have been divided into 3 parts differently:

- a. *Physical Level*
 - The internal level or the actual physical storage structure and access paths. The internal schema is a very low-level representation of the entire database.
 - It contains multiple occurrences of multiple types of internal record.
- b. *Logical level or the conceptual level*
 - Describes the Database structure of the whole database for the community of users.
 - This schema hides information about the physical storage structures and focuses on describing data types, entities, relationships, etc.
 - This logical level comes between the user level and physical storage view.
- c. *View or External level*
 - An external schema describes the part of the database which specific user is interested in. It hides the unrelated details of the database from the users.
 - Each external view is defined using an external schema, which consists of definitions of various types of external record of that specific view.

5. *Database Languages*

A DBMS has appropriate languages and interfaces to express database queries and updates. DBMS languages are divided into 4 types

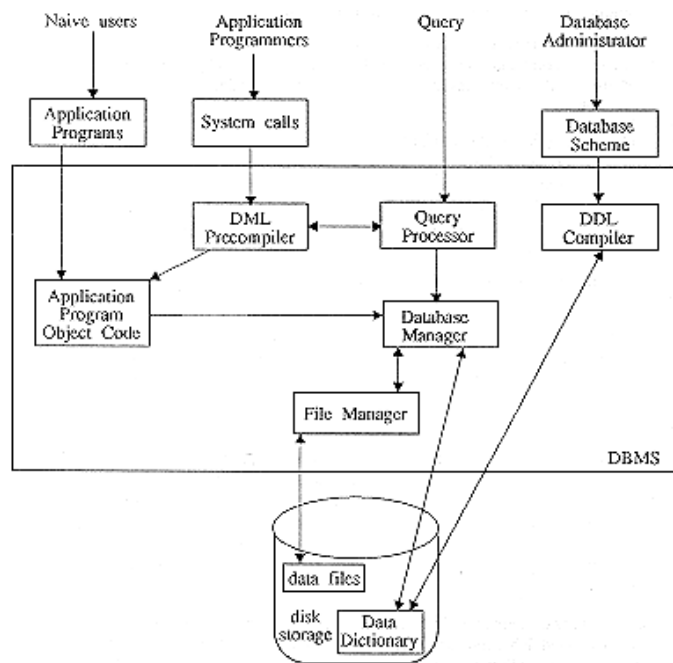
- *Data Definition Language (DDL)*
 - It is used to define database structure or pattern. DDL statements create the skeleton of the database.
 - DDL is used to store the information of the metadata like the number of tables and schemas, names, indexes, columns in each table, constraints, etc.
 - Tasks under DDL
 - CREATE: to create a database and its objects like (table, index, views, store procedure, function, and triggers)
 - ALTER: alters the structure of the existing database
 - DROP: delete objects from the database
 - TRUNCATE: remove all records from a table, including all spaces allocated for the records are removed
 - COMMENT: add comments to the data dictionary

- RENAME: rename an object
- **Data Manipulation Language (DML)**
 - It is used to access and manipulate the data in the database. It handles the user request. It is also known as query language.
 - Tasks done under DML
 - SELECT: retrieve data from a database
 - INSERT: insert data into a table
 - UPDATE: updates existing data within a table
 - DELETE: Delete all records from a database table
 - MERGE: UPSERT operation (insert or update)
 - CALL: call a PL/SQL or Java subprogram
 - EXPLAIN PLAN: interpretation of the data access path
 - LOCK TABLE: concurrency Control
 - There are 2 classes of languages
 - Procedural- user specifies what is required and how to get those data.
 - Non-procedural-- user specifies what data is required without specifying how to get those data.
- **Data Control Language (DCL)**
 - It is used to retrieve the stored or saved data. The DCL execution is transactional. It also has rollback parameters.
 - Tasks done under DCL
 - Grant: It is used to give access privileges to database.
 - Revoke: It is used to take back permissions from the users. They are connected, insert, usage, execute, delete, update and select.
- **Transaction Control Management**
 - TCL is used to run the changes made by DML statement.
 - Tasks done by TC
 - Commit: to save the transaction on the database.
 - Rollback: to restore the database to original since the last commit.

6. Relational Databases

- In RDMBS, the relationship between data is relational and data is stored in tabular form of columns and rows. Each column if a table represents an attribute and each row in a table represents a record. Each field in a table represents a data value.
- SQL is the language used to query a RDBMS including inserting, updating, deleting, and searching records.
- Relational databases work on each table has a key field that uniquely indicates each row, and that these key fields can be used to connect one table of data to another.

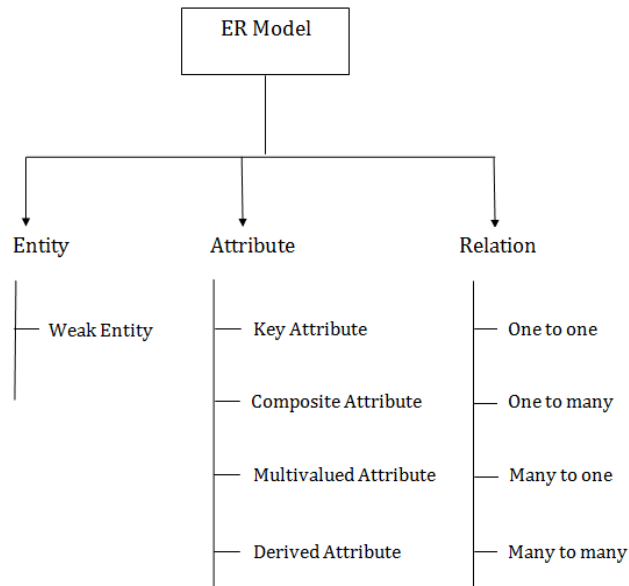
7. Database Architecture



8. ER Diagram

a. ER Model

- Entity-Relationship Model is a high data model to define the data elements and relationship for a specified system.
- It is developed a very simple and easy for a conceptual design for the database.
- In ER modeling, the database structure is portrayed as a diagram called as an entity-relationship diagram.
- Component of ER Diagram: **Entities set, Attributes and Relationships.**

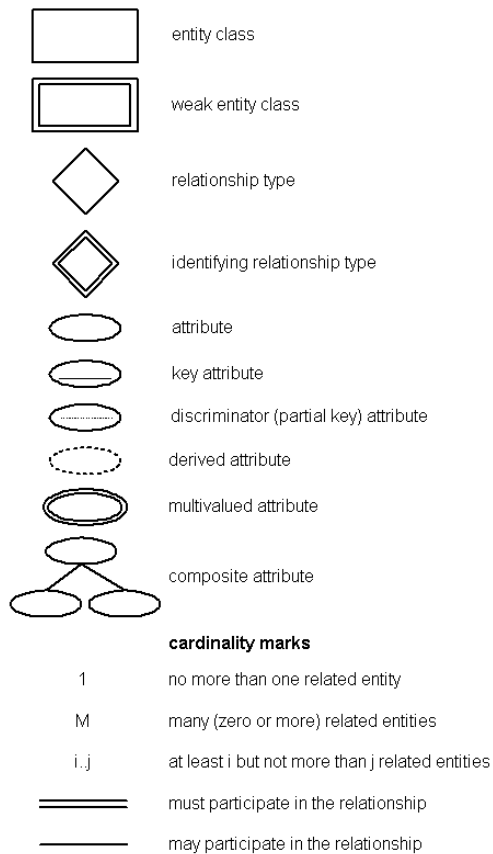


b. *Building blocks of E-R Diagram*

- The basic building blocks of all data models are **entities, attributes, relationships, and constraints**.
- **Entity** is anything (a person, a place, a thing, or an event) about which data are to be collected and stored. An entity represents a particular type of object in the real world.
- **An attribute** is a characteristic of an entity. For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone, customer address, and customer credit limit. Attributes are the equivalent of fields in file systems.
- **A relationship** describes an association among entities.
- **Constraints** enforce integrity: referential integrity, data integrity, domain integrity and so on. Each domain will have its own class of constraints (e.g. referential integrity is maintained by PRIMARY/FOREIGN KEY constraint; data integrity might have CHECK, NULL and DEFAULT constraints, etc.)

c. *Classification of Entity sets*

- Entity Type: is an object and set of all entities is called as entity set. It is also divided into entity and weak entity.
- Type of Entity:
 - **Entity**: any object that is relevant to a given system. It is represented in the *rectangle* symbol.
 - **Weak entity**: an entity that depends on the existence of another entity of another entity. It is represented as *double rectangle* symbol.
- ER Diagram entity symbol



d. Attribute Classification

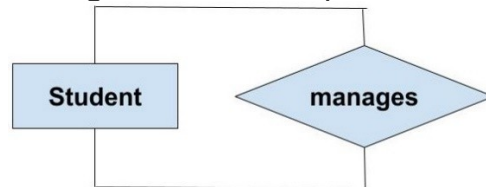
There are 14 types of attributes

- **Simple Attribute**: there is only attribute which we can't divided further.
Ex. Roll_no, age, etc.
- **Composite Attribute**: are those attributes which are composed of many other simple attributes.
Ex. Address (street, city, country...).
- **Single Valued**: attributes which can take only one value for a given entity from an entity set.
- **Multi valued**: attributes which can take more than one value for a given entity from an entity set.
Ex. Mobile_no, email_id, etc.
- **Derived Attribute**: to derive attribute from the existing attribute.
Ex. Age from DOB
- **Key Attribute**: are attributes which can identify an entity uniquely in an entity set.
- **Component Attribute**: the component of composite attribute.
- **Stored Attribute**: to store all the related attribute.
Ex. DOB...
- **Simple Single Valued**: you aren't able to divide and unique.
Ex. Passport, username...
- **Simple Multi Valued**: The value can't be divided with many values.
Ex. Phone numbers.

- *Composite Single Valued*: One attribute which can be divided into other attributes.
Ex. DOB-> Date, Month, Year...
- *Composite Multi Valued*: Many Attribute which can be divided into many more attributes.
Ex. Address: 2 different addresses with province, city, road,
- *Null Valued Attribute*: Haven't defined the attribute value.
Ex. Address--> ()
- *Descriptive Attribute*: Describe the relationship between the attributes.
Ex. Student: Listener, Lecturer: giving lectures.

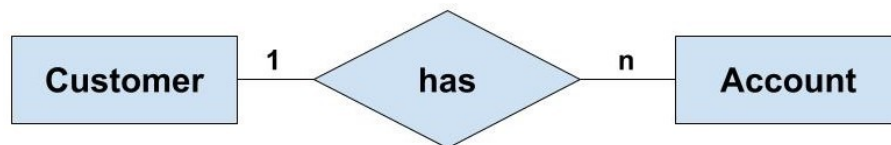
e. *Relationship Degree*

- The degree of a relationship is the number of entity types that participate (associate) in a relationship.
- Based on the number of entity types that are connected we have the following degree of relationships:
 - *Unary (degree 1)*: A unary relationship exists when both the participating entity type are the same. When such a relationship is present, we say that the degree of relationship is 1.



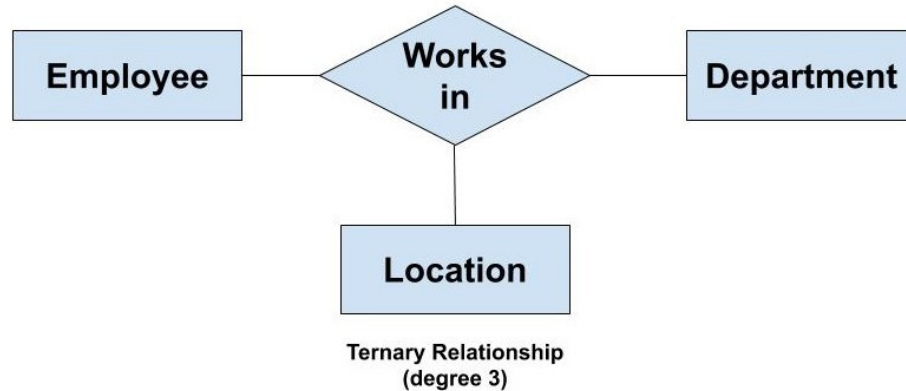
Unary Relationship
(degree 1)

- *Binary (degree 2)*: A binary relationship exists when exactly two entity type participates. It is easy to deal with such relationship as these can be easily converted into relational tables.

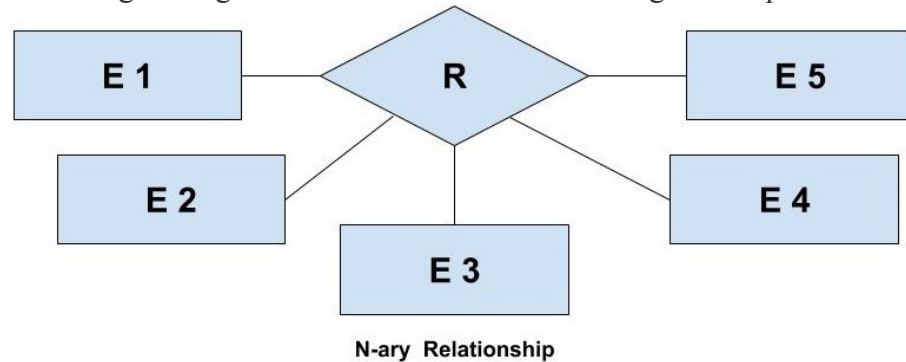


Binary Relationship
(degree 2)

- *Ternary (degree 3)*: A ternary relationship exists when exactly three entity type participates. As the number of entities increases in the relationship, it becomes complex to convert them into relational tables.



- *N-ary (n-degree)*: An N-ary relationship exists when 'n' number of entities are participating. So, any number of entities can participate in a relationship. But, relations with a higher degree are not common because the conversion of higher degree relations to relational tables gets complex.



f. Relationship Classification

- A relationship describes how entities interact. It is represented as diamonds symbol.
- *Cardinality Constraints*: express by drawing either a directed line (\rightarrow), signifying "one," or an undirected line ($—$), signifying "many," between the relationship set and the entity set.
- Participation type:
 - *Total Participation (double line)*: every entity in the entity set participates in at least one relationship in the relationship set.
 - *Partial participation*: some entities may not participate in any relationship in the relationship set.
- Mapping relationship
 - a. One to One
 - b. One to Many
 - c. Many to One
 - d. Many to many

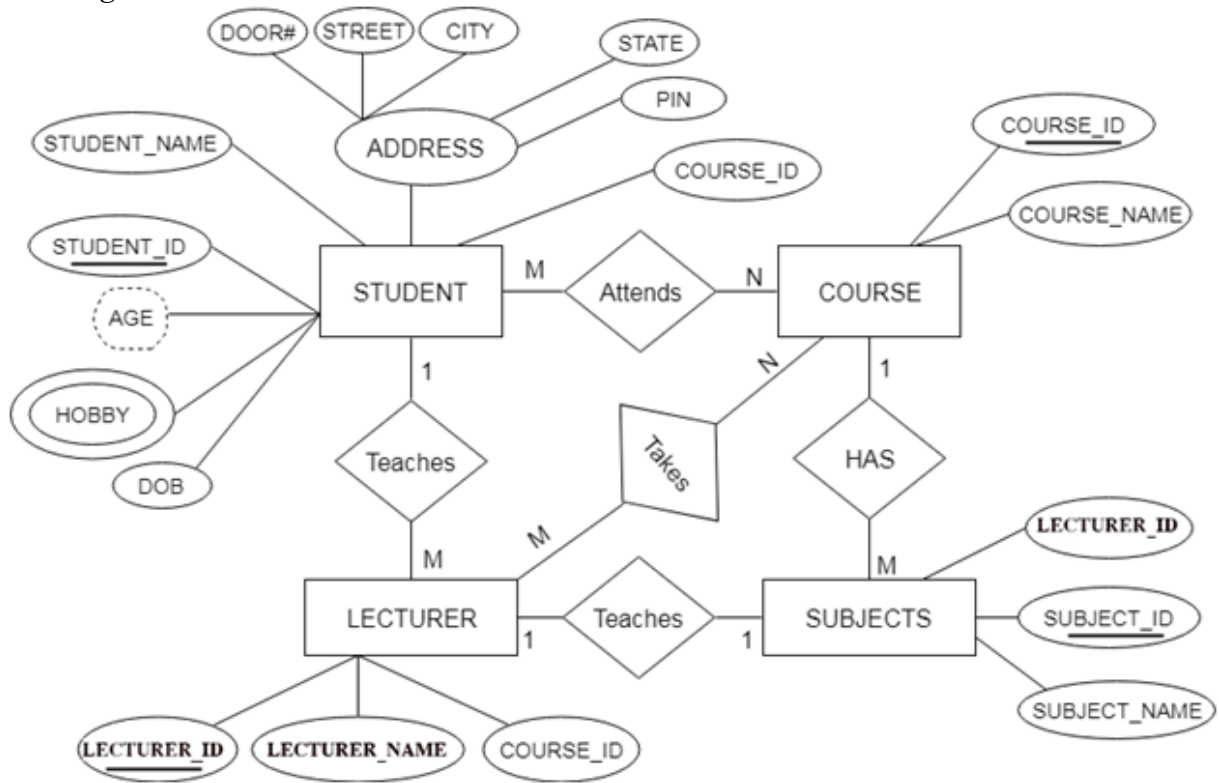
g. Reducing ER Diagram to Tables

There are some points for converting the ER diagram to the table:

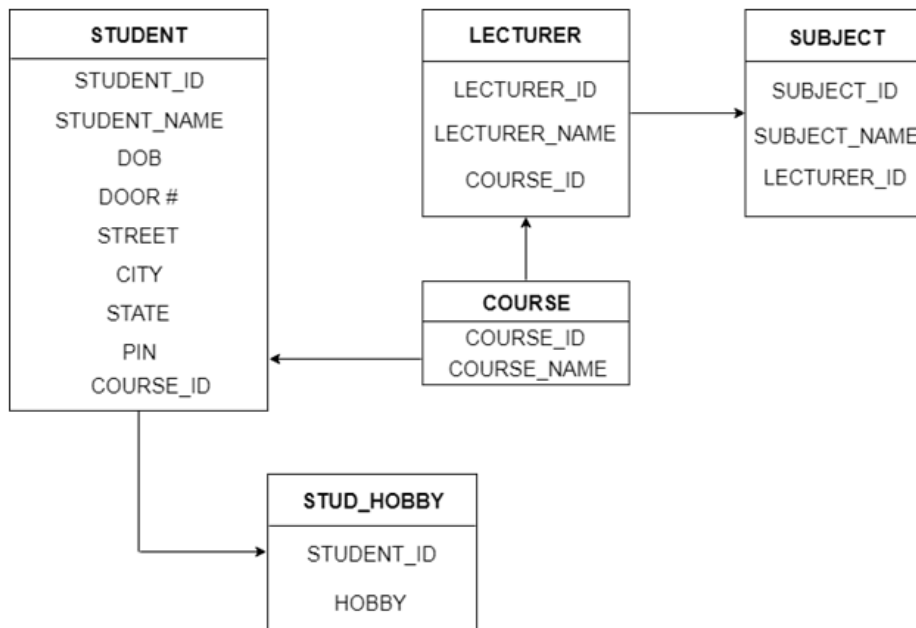
- Entity type becomes a table
- All single-valued attribute becomes a column for the table
- A key attribute of the entity type represented by the primary key
- The multivalued attribute is represented by a separate table

- Composite attribute represented by components.
- Derived attributes are not considered in the table.

ER Diagram



Class Diagram/ Table



h. *Advantages of ER Modeling*

- *Conceptually is very simple*: ER model is very simple because if we know relationship between entities and attributes, then we can easily draw an ER diagram.
- *Better visual representation*: ER model is a diagrammatic representation of any logical structure of database. By seeing ER diagram, we can easily understand relationship among entities and relationship.
- *Effective Communication tool*: It is an effective communication tool for database designer.
- *Highly integrated with relational model*: ER model can be easily converted into relational model by simply converting ER model into tables.
- *Easy conversion to any data model*: ER model can be easily converted into another data model like hierarchical data model, network data model and so on.





UNIT 2: Database Design

1. Introduction to Relational Model

- RDBMS stands for Relational Database Management System.
- A relational model can be represented as a table of rows and columns.
- A relational database has following major components:
 1. Table
 2. Record or Tuple
 3. Field or Column name or Attribute
 4. Domain
 5. Instance
 6. Schema
 7. Keys

2. CODD's Rules

These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

- *Rule 0* - RDBMS: the system must qualify as relational, as a database, and as a management system.
- *Rule 1* - Information Rule: Everything in a database must be stored in a table format.
- *Rule 2* - Guaranteed Access Rule: Every single data(value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value).
- *Rule 3* - Systematic Treatment of NULL values: The DBMS must allow each field to remain null (or empty). The null values in a database must be given a systematic and uniform treatment.
- *Rule 4* - Active Online Catalog based on the relational model: the structure description of the entire database must be stored in an online catalog, known as data dictionary.
- *Rule 5* - Comprehensive Data Sub-Language Rule: A database can only be accessed using a language having linear syntax that supports data definition, data manipulation, and transaction management operations.
- *Rule 6* - View Updating Rule: All the views of a database, which can theoretically be updated, must also be updatable by the system.
- *Rule 7* - High Level Insert, Update, and Delete Rule: A database must support high-level insertion, updating, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.
- *Rule 8* - Physical Data Independence: Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.
 - How we organize and structure the data
-  *Rule 9* - Logical Data Independence: The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it.
 - How we implement our data
-  *Rule 10* - Integrity Independence: All its integrity constraints can be independently modified without the need of any change in the application. This rule makes a database independent of the front-end application and its interface.
 - How we give the security to the data
 - Avoid duplication to the data
-  *Rule 11* - Distribution Independence: Users should always get the impression that the data is located at one site only. It has been regarded as the foundation of distributed database systems.
 - How to distribute the table to other different places with security
-  *Rule 12* - Subversion Rule: If a system has an interface that provides access to low-level records, then the interface must not be able to subvert the system and bypass security and integrity constraints.
 - Support the low level and all the system
 - Should the integration and security

3. Concept of Key

- A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.
- Why we need key?
 - o Keys help you to identify any row of data in a table.
 - o Keys ensure that you can uniquely identify a table record despite these challenges.
 - o Allows you to establish a relationship between and identify the relation between tables
 - o Help you to enforce identity and integrity in the relationship.
- Various Keys type in DBMS
 - o **Primary key**: a set of one or more attribute of the table that uniquely identify a record in database table. Rules to define the primary key:
 - Two rows can't have the same primary key value
 - It must for every row to have a primary key.
 - The primary key field cannot be **null**.
 - The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.
 - o **Alternate key**: All the keys which are not primary key are called an alternate key. It is a candidate key which is currently not the primary key.
 - o Ex. Since Roll_no is the primary key, StuID or email should be alternative key.
 - o **Super Key**: a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.
 - o **Candidate Key**: A super key with no repeated attribute is called candidate key. Properties of candidate keys are:
 - constraint: Value in one relation must appear in another
 - It must contain a unique value.
 - It may have multiple attributes.
 - Must not a null value.
 - Should contain minimum fields to ensure uniqueness.
 - Uniquely identify each record in a table.
 - o **Foreign key**: a field key in the database that is primary in another table.
 - o **Compound Key**: Compound key has many fields which allow you to uniquely recognize a specific record. Any part of the compound key can be a foreign key.
 - o **Composite key**: A key which has multiple attributes to uniquely identify rows in a table is called a composite key. The composite key may or maybe not a part of the foreign key.
 - o **Surrogate key**: An artificial key which aims to uniquely identify each record is called a surrogate key. These kinds of key are unique because they are created when you don't have any natural primary key.

4. Relational Algebra

Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

The fundamental operations of relational algebra are as follows –

➤ Select

- 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 propositional logic formula which may use connectors like **and**, **or**, and **not**. These terms may use relational operators like $=$, \neq , \geq , $<$, $>$, \leq .
Example: Selects tuples from books where subject is 'database'.
Output: $\sigma_{\text{subject} = \text{"database"}}(\text{Books})$

➤ Project

- It projects column(s) that satisfy a given predicate.
- Notation – $\Pi_A(r)$ where A is attribute name of relation r .
- Duplicate rows are automatically eliminated, as relation is a set.
Example: Selects and projects columns named as subject and author from the relation Books.
Output: $\Pi_{\text{subject, author}}(\text{Books})$

➤ Union

- It performs binary union between two given relations.
- **Notation** – $r \cup s$ 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.
Example: Projects the names of the authors who have either written a book or an article or both.
Output: $\Pi_{\text{author}}(\text{Books}) \cup \Pi_{\text{author}}(\text{Articles})$

➤ Set different

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.
- Notation: $r - s$, meaning to find all the tuples that are present in r but not in s .
Example: Provides the name of authors who have written books but not articles.
Output: $\Pi_{\text{author}}(\text{Books}) - \Pi_{\text{author}}(\text{Articles})$

➤ Cartesian product

- Combines information of two different relations into one.
- **Notation** – $r \times s$, where r and s are relations and their output will be defined as – $r \times s = \{q \mid q \in r \text{ and } t \in s\}$.
Example: Yields a relation, which shows all the books and articles written by tutorialspoint.
Output: $\sigma_{\text{author} = \text{"tutorialspoint"}}(\text{Books} \times \text{Articles})$

➤ Rename

- 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 ρ .

- **Notation** – $\rho_x(E)$, where the result of expression **E** is saved with name of **x**.

Additional operations are – Set intersection, Assignment and Natural join.

5. Relation Calculus

Relational Calculus is a non-procedural query language, that is, it tells what to do but never explains how to do it. Relational calculus exists in two forms:

- **Tuple Relational Calculus (TRC)**

- Filtering variable ranges over tuples
- **Notation** – $\{T \mid \text{Condition}\}$, returns all tuples **T** that satisfies a condition.
Example: $\{T.\text{name} \mid \text{Author}(T) \text{ AND } T.\text{article} = \text{'database'}\}$
Output – Returns tuples with 'name' from Author who has written article on 'database'.
- It can be quantified. We can use Existential (\exists) and Universal Quantifiers (\forall).
Example: $\{R \mid \exists T \in \text{Authors} (T.\text{article} = \text{'database'} \text{ AND } R.\text{name} = T.\text{name})\}$
Output – The above query will yield the same result as the previous one.

- **Domain Relational Calculus (DRC)**

- In DRC, the filtering variable uses the domain of attributes instead of entire tuple values (as done in TRC, mentioned above).
- **Notation** – $\{a_1, a_2, a_3, \dots, a_n \mid P(a_1, a_2, a_3, \dots, a_n)\}$, Where a_1, a_2 are attributes and **P** stands for formulae built by inner attributes.
Example: $\{ \langle \text{article}, \text{page}, \text{subject} \rangle \mid \in \text{TutorialsPoint} \wedge \text{subject} = \text{'database'} \}$
Output – Yields Article, Page, and Subject from the relation TutorialsPoint, where subject is database.

6. Aggregation functions

Aggregation function takes a collection of values and returns a single value as a result.

There are 5 types of aggregate function:

- **avg: average value**
 - 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)**
- **min: minimum value**
 - MIN function is used to find the minimum value of a certain column. This function determines the smallest value of all selected values of a column.
 - Syntax: **MIN () or MIN([ALL|DISTINCT] expression)**
- **max: maximum value**
 - 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)**
- **sum: sum of values**
 - 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)**
- **count: number of values**
 - 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. It uses the COUNT (*) that

returns the count of all the rows in a specified table. COUNT (*) considers duplicate and Null.

- o Syntax: **COUNT (*) or COUNT([ALL|DISTINCT] expression)**

UNIT 3: SQL

1. Create database

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

- Syntax: **CREATE DATABASE DatabaseName;**
- Example: CREATE DATABASE testDB;

2. Drop database

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

- Syntax: **DROP DATABASE DatabaseName;**
- Example: DROP DATABASE testDB;

3. Use database

The SQL USE statement is used to select any existing database in SQL schema. When you have multiple databases in your SQL Schema, then before starting your operation, you would need to select a database where all the operations would be performed.

- Syntax: **USE DatabaseName;**
- Example: USE testDB;

4. Create table

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

Syntax: **CREATE TABLE table_name (**
 column1 datatype,
 column2 datatype,
 columnN datatype);

5. Create table with another table

The statement is used to create a new copy of an existing table.

- Syntax: **CREATE TABLE NewTableName**
 AS SELECT [column1, column2, ..., columnN]
 FROM ExistingTableName [where];
- Example: CREATE TABLE SALARY AS
 SELECT ID, SALARY
 FROM CUSTOMERS;

6. Select

SQL SELECT Statement is used to fetch the data from a database table which returns data in the form of result table. These result tables are called result-sets.

- Syntax: **SELECT column1, column2...columnN FROM table_name;**

7. Where

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

- Syntax: **SELECT column1, column2...columnN FROM table_name WHERE**
 [condition]

8. Truncate table

The TRUNCATE TABLE command deletes the data inside a table, but not the table itself.

- Syntax: **TRUNCATE TABLE TableName;**

9. Alter

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

- Syntax: **ALTER TABLE TableName {ADD|DROP|MODIFY} columnName {data_type};**

10. Rename

The RENAME TABLE statement is used to rename a table.

- Syntax: **RENAME TABLE TableName TO NewTableName;**
- Example: ALTER TABLE table_name RENAME TO new_table_name;

11. Describe

The DESC statement is used to describe the structure of the table.

- Syntax: **DESC TableName;**

12. Delete or drop

The DROP TABLE command deletes a table in the database.

- Syntax: **DROP TABLE TableName;**
- Example: DROP TABLE CUSTOMERS;

13. Insert

The INSERT INTO statement is used to insert new records in a table.

- Syntax: **INSERT INTO table_name (column1, column2...columnN) VALUES (value1, value2...valueN);**

14. Update table

The UPDATE statement is used to modify the existing records in a table.

- Syntax: **UPDATE table_name
SET column1 = value1, column2 = value2...columnN = valueN
WHERE [condition];**

15. Expression

An expression is a combination of one or more values, operators, and SQL functions that evaluate to a value. There are 3 types of expressions:

- Boolean Expressions
- Numeric Expression
- Date Expressions

Syntax: **SELECT column1, column2, columnN
FROM table_name
WHERE [CONDITION|EXPRESSION];**

Unit 4 Relational Database Design

1. Database Design

- There are some approaches to follow in order to design the best database. They are:
 - Security
 - Integrity

- *Database Design methods*
- *The level of table design*
- *Rules*
- Important of good database design:
 - Responsive applications
 - Enable you to extend database with ease
 - Sharing database for many applications
 - Integrity of data
 - Security of data
- Database Primary design methods are:
 - ***Bottom-up design method***
Steps that you should follow for this design approach are as follows:
 - Step-1: Identify attributes for which you need to store data
 - Step-2: Group the attributes into relations based on associations between the attributes through the process of normalization.
 - ***Top-down design method***
Steps that you should follow for this design approach are as follows:
 - Step-1: Create an Entity-Relationship Model
 - Step-2: Start with Entity Types of the Entity- Relationship Model as initial relations, apply normalization procedure to come up with the relations.
- Database Design Phase
Following are the three phases of database design:
 - Conceptual Database Design
 - Input: Output of Systems Analysis – Data-Flow Diagrams, Software requirement specifications, etc.
 - Output: Conceptual Data Model (ER-Model Diagram)
 - Goal: To get a model capturing complete user data requirements, understand and represent relationships among entity types
 - Logical Database Design
 - Input: Conceptual Data Model
 - Output: A DBMS specific Data Model such as relation data model, hierarchical data model and network data model.
 - Physical Database Design
 - Input: DBMS Specific Data Model
 - Output: Database schema with implementation details such as physical organization of database objects, indexes to be used and location of the objects on secondary storage.

2. ***DB Design Rules***

There are 9 rules as following in order to design the best Database.

- 1) ***Determine the purpose of the database*** - This helps prepare for the remaining steps.
- 2) ***Find and organize the information required*** - Gather all of the types of information to record in the database, such as product name and order number

- 3) ***Divide the information into tables*** - Divide information items into major entities or subjects, such as *Products* or *Orders*. Each subject then becomes a table
- 4) ***Turn information items into columns*** - Decide what information needs to be stored in each table.
- 5) ***Specify primary keys*** - Choose each table's primary key.
- 6) ***Set up the table relationships*** - Look at each table and decide how the data in one table is related to the data in other tables.
- 7) ***Refine the design*** - Analyze the design for errors. Create tables and add a few records of sample data.
- 8) ***Apply the normalization rules*** - Apply the data normalization rules to see if tables are structured correctly and make adjustments to the tables.

3. ***Redundancy and Anomaly***

Data redundancy is repetition of multiple data in different places will can cause many problems.

Anomalies are caused when there is too much redundancy in the database's information. It can often be caused when the tables that make up the database suffer from poor construction.

There are three types of Data Anomalies:

- Update Anomalies: happen when modifying the current record.
- Insertion Anomalies: happen when inserting vital data into the database is not possible because other data is not already there.
- Deletion Anomalies: happen when the deletion of unwanted causes desired to be deleted as well.

4. ***Functional Dependency***

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

$X \rightarrow Y$

X is known as a determinant, and Y production is known as a dependent.

There are 2 types of Functional Dependency

- Trivial Dependency
 - o $A \rightarrow B$ has trivial functional dependency if B is a subset of A.
 - o The following dependencies are also trivial like: $A \rightarrow A$, $B \rightarrow B$
- Non-Trivial Dependency
 - o $A \rightarrow B$ has a non-trivial functional dependency if B is not a subset of A.
 - o When $A \cap B$ is NULL, then $A \rightarrow B$ is called as complete non-trivial.

5. ***Normalization – 1NF, 2 NF, 3NF, BCNF, 4NF***

- Normalization is a technique of organizing the data into multiple related tables, to minimize *data redundancy*. Data redundancy is repetition of multiple data in different places will can cause many problems.
- Type of normalization
 - o 1st Normal Form

- 2nd Normal Form
- 3rd Normal Form
- BCNF

1st Normal Form

- Step 1 of the normalization, the basic requirement to design the database.
- The 1st normalization has to follow some following rules:
 - Each column should contain atomic values.
 - A column should contain values that are the same type.
 - Each column should have a unique name.
 - Order in which data is stored doesn't matter.

2nd Normal Form (2NF)

- Criteria for a table to be a 2nd normal form
 - It should be in the 1st normal form
 - NO partial Dependencies
- Partial Dependency: 2 or more attribute in table are dependent on each other.
- Solution
 - Try to remove those partial dependencies attributes to the new table with some primary keys.
 - If there are more than one primary keys, you have to combine the two keys together as a *composite key*.

3rd Normal Form (3NF)

- Criteria for a table to be a 3rd normal form
 - It should be in the 3rd normal form
 - It should not have transitive Dependency
- Transitive Dependency: when an attribute is depending another attribute, which is not a primary key.
- Solution: storing those transitive dependencies attributes in another table.

BCNF- Boyce-Codd Normal Form (3.5 NF)

- Upgraded of 3NF
- A table to be in BCNF should satisfy:
 - It should be in the 3NF
 - NO Transitive Dependency (A: Non-prime \rightarrow B: Prime attribute)

4th Normal Form (4NF)

- A 4th formal normalization must satisfy:
 - It should be in BCNF
 - It should not have Multi-valued dependency
- Multi-valued dependency: b
 - $A \twoheadrightarrow B$, for a single value of A, more than one value of B exists.
 - Table should have at least 3 columns.
 - For this table with A, B, C columns, B and C should be independent.

5th Normal Form (5NF)

- Known As PJNF- Project Normal form
- A table to be 5th normal form must:
 - Should be in 4NF

- It should not have Join Dependency
- A Join Dependency: an attribute which can be divided into other small attributes.
- To understand the 5NF, there are two ways:
 - Understand the JOIN dependency
 - Understand the basic concept of breaking down a table

6. *Problems Using Normal forms.*

A normalized database is not as advantageous under conditions where an application is read-intensive. Here are some of the disadvantages of normalization:

1. **Since data is not duplicated, table joins are required.** This makes queries more complicated, and thus read times are slower.
2. **Since joins are required, indexing does not work as efficiently.** Again, this makes read times slower because the joins don't typically work well with indexing.

UNIT 5: Database Administration

1. *Transaction Management*

A transaction is a sequence of operations performed as a single logical unit of work. A transaction is a logical unit of work that contains one or more SQL statements.

2. *ACID Property*

ACID properties are using in database transaction to ensure the success during data transaction.

- Atomicity: prefer to the entire transaction take place at once or don't happen at all. It Checks either transaction execute 0% or 100%. It induces the following operations:
 - Abort: If the transaction aborts, changes made to the database are not visible.
 - Commit: If the transaction commit, changes made are visible.
- Consistency: Ensuring that the database must remain in a consistent state after any transaction.
- Isolation: Returning the intermediate transaction results must be hidden from other concurrently executed transactions.
- Durability: Once a transaction completed successfully, the changes it has made into the database should be permanent.

3. *Scheduling*

- A schedule is a process of grouping the transactions into one and executing them in a predefined order. It is created to execute the transactions.
- A schedule is the chronological (sequential) order in which instructions are executed in a system.
- A schedule is required in a database because when some transactions execute in parallel, they may affect the result of the transaction. Meaning if one transaction is updating the

values which the other transaction is accessing, then the order of these two transactions will change the result of another transaction.

- There are 3 types of schedule
 - A serial schedule is one in which ***no transaction starts until a running transaction has ended.*** Transactions are executed one after another, as transactions are executed in a serial manner.
 - Interleaved Schedule is the schedule that ***interleaves the execution of different transactions,*** meaning that the second transaction is started before the first one could end and execution can switch between the transactions back and forth.
 - Equivalent Schedule is the one which ***two schedules produce the same result*** after execution, they may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

4. ***Serializability***

- They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.
- In the serial schedule, only one transaction is allowed to execute at a time i.e. no concurrency is allowed.
- Whereas in serializable schedules, multiple transactions execute simultaneously i.e. concurrency is allowed.
- If a given ***schedule can be converted into a serial schedule by swapping its non-conflicting operations,*** then it is called as a ***conflict serializable schedule.***
- Forms of Serializability
 1. Conflict Serializability: If a given schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a conflict serializable schedule.
 2. View Serializability: is a process to find out that a given schedule is view serializable or not
- To check whether a given schedule is view serializable, we need to check whether the given schedule is View Equivalent to its serial schedule. Three conditions of View Equivalent:
 - Initial Read - Initial read of each data item in transactions must match in both schedules.
Here, initial read means the first read operation on a data item
 - Update Read - If in schedule S1, the transaction T1 is reading a data item updated by T2 then in schedule S2, T1 should read the value after the write operation of T2 on the same data item.
 - Final Write - Final write operations on each data item must match in both the schedules.

5. ***Concurrency Control***

- Concurrency is the ability of a database to allow multiple (more than one) users to access data at the same time. Three problems due to concurrency

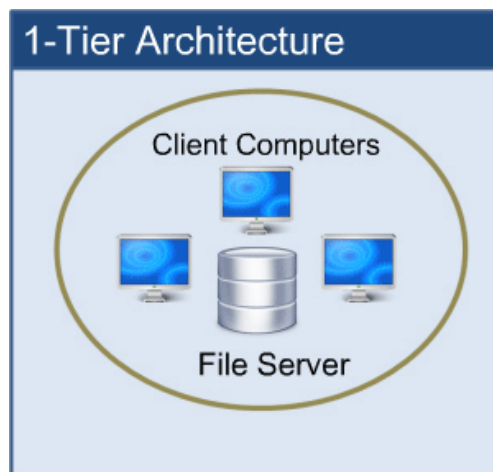
1. Lost update problem
 - happens when two transactions access and update the same data simultaneously, and the result of one is overwritten by another.
 - How to avoid: A transaction, **T2 must not update the data item (X) until the transaction T1 can commit** data item (X).
2. Dirty read problem
 - The dirty read arises when one transaction updates some item and then fails due to some reason. This updated item is retrieved transaction by another before it is changed back to the original value.
 - How to avoid: a transaction **T1 must not read the data item (X) until the transaction T2 can commit data item (X).**
3. Incorrect retrieval problem
 - The inconsistent retrieval problem arises when **one transaction retrieves data to use in some operation but before it can use this data another transaction updates that data and commits.**
 - Through this change will be hidden from first transaction and it will continue to use previous retrieved data. This problem is also known as inconsistent analysis problem.
 - **How to avoid:** A transaction **T2 must not read or update data item (X) until the transaction T1 can commit** data item (X).
- Concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories:
 - Lock based protocols
 - Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds –
 - **Binary Locks** – A lock on a data item can be in two states; it is either locked or unlocked.
 - **Shared/exclusive** – This type of locking mechanism differentiates the locks based on their uses.
 - There are four types of lock protocols available –
 - Simplistic Lock Protocol: allows transactions to obtain a lock on every object before a 'write' operation is performed.
 - Pre-claiming Lock Protocol: evaluates their operations and create a list of data items on which they need locks.
 - Two-Phase Locking 2PL: This locking protocol divides the execution phase of a transaction into three parts.
 - In first part, when the transaction starts executing, it seeks permission for the locks it requires.
 - The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts.
 - In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.

- Strict Two-Phase Locking:
 - The first phase of Strict-2PL is same as 2PL.
 - Strict-2PL holds all the locks until the commit point and releases all the locks at a time.
- Time stamp-based protocols: timestamp-based protocols start working as soon as a transaction is created. Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction.

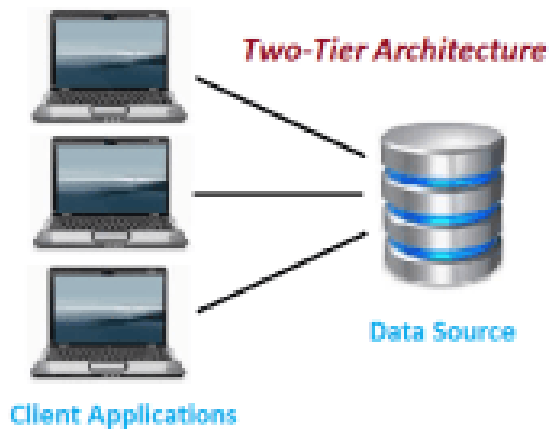
7. *Client server DB*

- Client server database consists of two logical components. One is “Client” and the other one is “Server”.
- Clients are those who send the request to perform a specific task to the server.
- Servers are normally receiving the command sent by the clients, perform the task and send the appropriate result back to the client.
- Normally, there are three types of client server architecture available in database management system. These are listed below.

(a) ***Single tier client server computing model/ One-tier architecture:*** involves putting all of the required components for a software application or technology on a single server or platform.



(b) ***Two tier client server computing model/ Two-tier architecture***



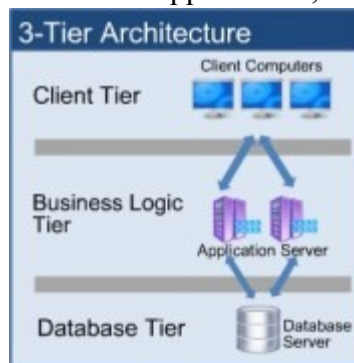
Ex. Client programs using ODBC/JDBC to communicate with a database.

(c) Three tier client server computing model/ Three-tier architecture

Three tier architecture is the improvement over two-tier architecture. Three tier architecture has three layers.

- The first layer is the user interface which runs on client system.
- The second layer is called the application server. It is used for business logic and data processing. The third layer is known as database server.
- It is a database management system which stores the data as needed by the middle layer.

ex. Web-based applications, and applications built using "middleware".



8. Distributed DB

A **distributed database** is a collection of multiple interconnected databases, which are spread physically across various locations that communicate via a computer network.

Features

- Databases in the collection are logically interrelated with each other. Often, they represent a single logical database.
- Data is physically stored across multiple sites. Data in each site can be managed by a DBMS independent of the other sites.
- The processors in the sites are connected via a network. They do not have any multiprocessor configuration.
- A distributed database is not a loosely connected file system.

- A distributed database incorporates transaction processing, but it is not synonymous with a transaction processing system.

Make it better, do the best!!!