DATABASE MANAGEMENT SYSTEMS

MODULE-1

QUESTIONS & ANSWERS

1. What is meant by DBMS and explain applications of DBMS?

A **database** is a collection of data, typically describing the activities of one or more related organizations. For example, a university database might contain information about the following:

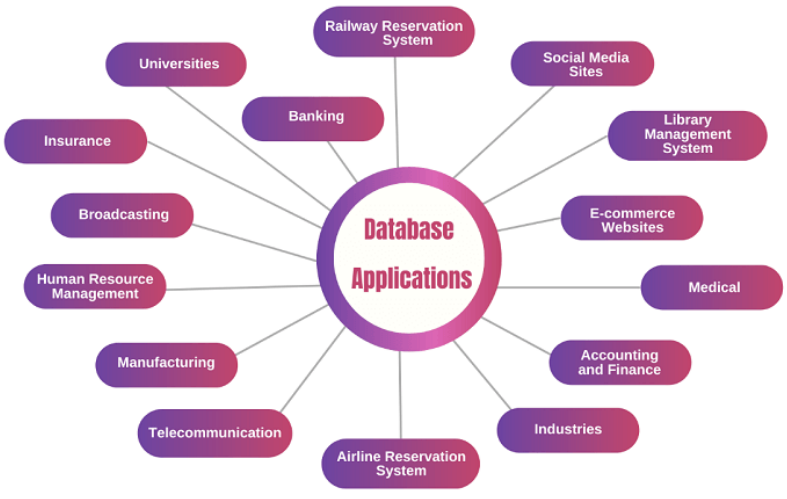
- **Entities** such as students, faculty, courses, and classrooms.
- **Relationships** between entities, such as students' enrollment in courses, faculty teaching courses, and the use of rooms for courses.

A **Database Management System (DBMS)** is a software system that is designed to **manage and organize data in a structured manner.** It allows users to create, modify, and query a database, as well as manage the security and access controls for that database. DBMS provides an environment to store and retrieve data in convenient and efficient manner.

The goal of database management systems is how to **organize information** in a DBMS and to **maintain it** and **retrieve it** efficiently, that is, **how to design a database and use a DBMS effectively.**

DATABASE APPLICATIONS

Nowadays, any business that has small or large amounts of data needs a database to store and manage the information. The database is an easy, reliable, secure, and efficient way to maintain business information. There are many applications where databases are used. Here are some representative applications.



ENTERPRISE INFORMATION

- **Sales:** For customer, product, and purchase information.
- **Accounting:** For payments, receipts, account balances and other accounting information.
- **Human resources:** For information about employees, salaries, and for generation of paychecks.
- **Manufacturing:** For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
- **Online retailers:** For sales data noted above plus online order tracking and maintenance of online product evaluations.

BANKING AND FINANCE

- **Banking:** For customer information, accounts, loans, and banking transactions.
- **Credit card transactions:** For purchases on credit cards and generation of monthly statements.
- **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data.

UNIVERSITIES

For student information, course registrations, and grades (in addition to human resources and accounting).

AIRLINES

For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

TELECOMMUNICATION

For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

---

2. Explain the architecture of Database System with neat diagram?

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. 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 query processor and the storage manager components as shown in the following figure, the structure of database system architecture.

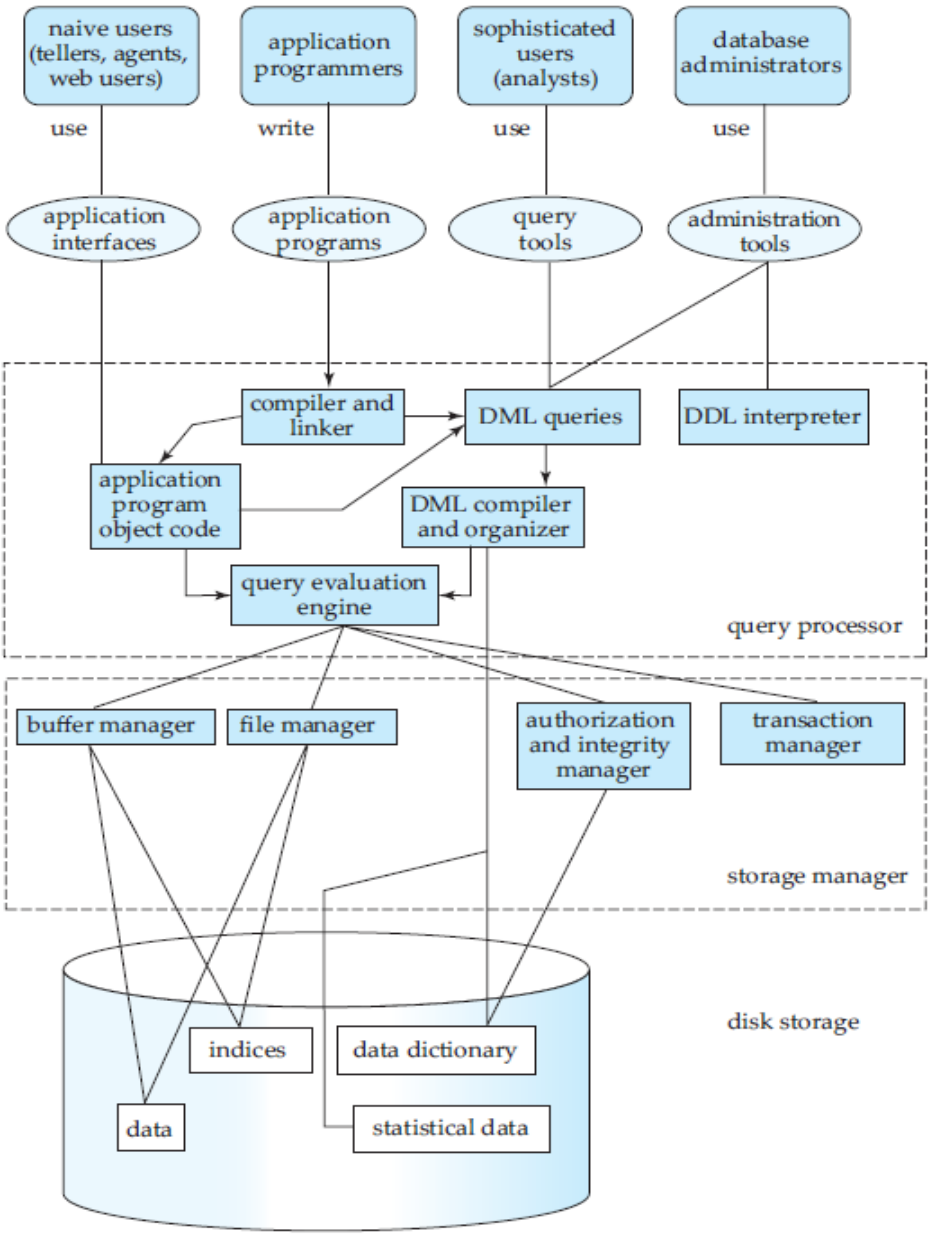


Figure System structure.

Disk Storage

The storage manager implements several data structures as part of the physical system implementation.

- **Data files** are used to store the database itself.
- **Data dictionary** is used to stores metadata about the structure of the database, in particular the schema of the database.
- **Indices**- which can provide fast access to data items. A database index provides pointers to those data items that hold a particular value.

Storage Manager

A storage manager is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system. The storage manager is responsible for the interaction with the file manager.

The storage manager translates the various DML statements into low-level file-system commands. Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.

The Storage Manager Components are:

Authorization and integrity manager: which tests for the satisfaction of integrity constraints and checks the authority of users to access data.

Transaction manager: which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.

File manager: which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.

Buffer manager: which is responsible for fetching data from disk storage into main memory and deciding what data to cache in main memory.

Query Processor

Query Processor Components:

DDL interpreter: It interprets DDL statements and records the definitions in the data dictionary.

DML compiler: It translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.

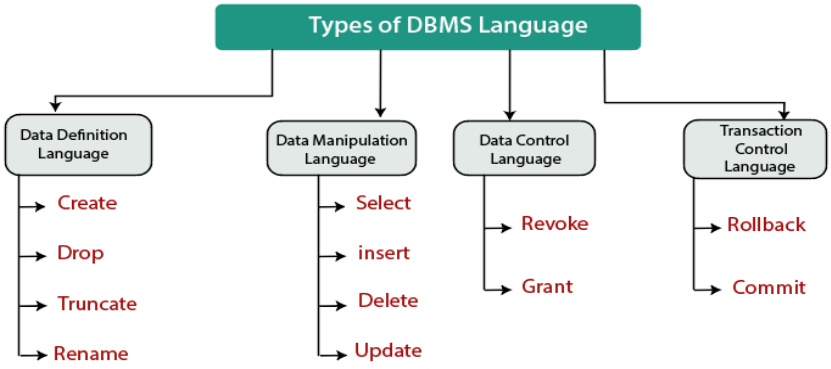
Query evaluation engine: It executes low-level instructions generated by the DML compiler.

---

3. Explain Database languages with examples?

A DBMS has appropriate languages and interfaces to express database queries and updates. And they can be used to read, store and update the data in the database. The different types of DBMS languages are as follows:

- Data Definition Language
- Data Manipulation Language
- Data Control Language
- Transaction Control Language



In practice, the data-definition and data-manipulation languages are not two separate languages; instead, they simply form parts of a single database language, such as the widely used SQL language.

DATA DEFINITION LANGUAGE - DDL

We specify a **database schema** by a set of definitions expressed by a special language called a **Data-Definition language (DDL).** DDLs are used to define the metadata of the database. i.e.; using this, we create schema, tables, constraints, indexes in the database. DDLs are also used to modify Schema, tables index etc. Basically, using DDL statements, we create skeleton of the database.

Create

Create is used to create schema, tables, index, and constraints in the database. The basic syntax to create table is as follows.

Syntax: CREATE TABLE tablename (Column1 DATATYPE, Column2 DATATYPE, … ColumnN DATATYPE);

For example: Create employee table

CREATE TABLE Employee (eid number(5),ename varchar2(20),salary number(6));

Alter

This command is used to modify the structure of the Schema or table. It can even used to add, modify or delete the columns in the table. The syntax for alter statement is as follows.

Syntax:

To add a new column: ALTER TABLE table_name ADD column_name *datatype*;

Ex: ALTER TABLE employee ADD department varchar(50);

To delete a column: ALTER TABLE table_name DROP COLUMN column_name;

Ex: ALTER TABLE Employee DROP COLUMN salary;

To modify a column: ALTER TABLE table_name MODIFY column_name *datatype*;

To rename table: ALTER TABLE table_name RENAME TO new_table_name;

To rename the column: ALTER TABLE table_name RENAME COLUMN old_Column_name to new_Column_name

DATA MANIPULATION LANGUAGE – DML

A Data Manipulation Language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- Retrieval of information stored in the database
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

Select

Select command helps to pull the records from the tables or views in the database. It either pulls the entire data from the table/view or pulls specific records based on the condition. It can even retrieve the data from one or more tables/view in the database.

Syntax: SELECT * FROM table_name;

Insert

Insert statement is used to insert new records into the table. The general syntax for insert is as follows:

Syntax: INSERT INTO TABLE_NAME (col1, col2, col3,...colN) VALUES (value1, value2, value3,...valueN);

Update

Update statement is used to modify the data value in the table.

Syntax:

UPDATE table_name SET column_name1 = value1, column_name2 = value2, ... column_nameN = valueN, [WHERE condition]

Delete

Using Delete statement, we can delete the records in the entire table or specific record by specifying the condition.

Syntax:

DELETE FROM table_name [WHERE condition];

DELETE FROM EMPLOYEE WHERE EMP_ID = 110;

DATA CONTROL LANGUAGE - DCL

DCL languages are used to control the user access to the database, tables, views, procedures, functions and packages. They give different levels of access to the objects in the database.

Grant

GRANT provides the privileges to the users on the database objects. The privileges could be select, delete, update and insert on the tables and views. We can either give all the privileges or any one or more privileges to the objects. The syntax of GRANT is as below:

Syntax:
GRANT privilege_name ON object_name TO **{**user_name | PUBLIC | role_name**}**

[WITH GRANT OPTION**]**;

Revoke

REVOKE removes the privileges given on the database objects. We can remove all the privileges or remove one or more privileges from the objects.

Syntax:
REVOKE privilege_name ON object_name

FROM **{**user_name |PUBLIC |role_name**}**

Ex: REVOKE INSERT ON STUDENT FROM Mathew; -- Removes the INSERT grant from Mathew on STUDENT Table

Transaction Control Language:

The Transaction Control Language commands are used to run the changes made by the DML commands one more thing is TCL can be grouped into a logical transaction. We have two different commands in this category I listed them for reference.

- **Commit:** It is used to save the transaction on the Database. These are very useful in the banking sector.
- **Rollback**: It is used to restore the database to its original state from the last commit. This command also plays an important role in Banking Sectors.

---

4. Explain i) Views of Data ii) Data Abstraction, Instance and schema with example.

VIEWS OF DATA

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

DATA ABSTRACTION

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system.

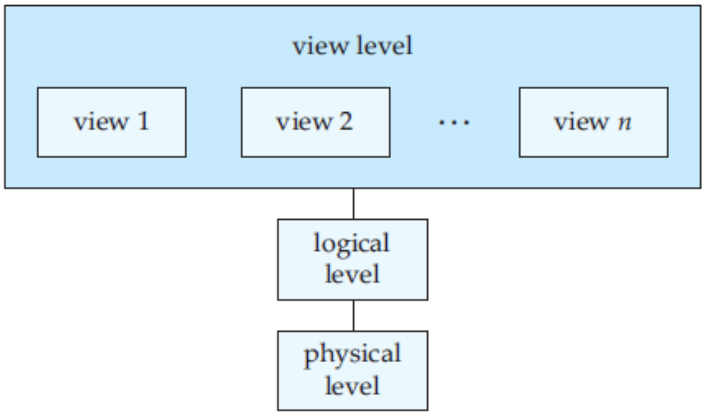


Figure The three levels of data abstraction.

Mainly there are three levels of abstraction for DBMS, which are as follows

- Physical or Internal Level
- Logical or Conceptual Level
- View or External Level

Physical Level

The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail.

Logical level

The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

View level

The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

For example, we may describe a record as follows:

type *instructor* = **record**

ID : **char** (5);

name : **char** (20);

dept name : **char** (20);

salary : **numeric** (8,2);

end;

This code defines a new record type called *instructor* with four fields. Each field has a name and a type associated with it. A university organization may have several such record types, including

• *department*, with fields *dept name*, *building*, and *budget*

• *course*, with fields *course id*, *title*, *dept name*, and *credits*

• *student*, with fields *ID*, *name*, *dept name*, and *tot cred*

At the **physical level**, an *instructor*, *department*, or *student* record can be described as a block of consecutive storage locations. The compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest-level storage details from database

programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the **logical level**, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the **view level**, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database are defined, and a database user sees some or all of these views.

INSTANCE AND SCHEMA

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

Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the **view level**, sometimes called **subschemas**, that describe different views of the database. Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs.

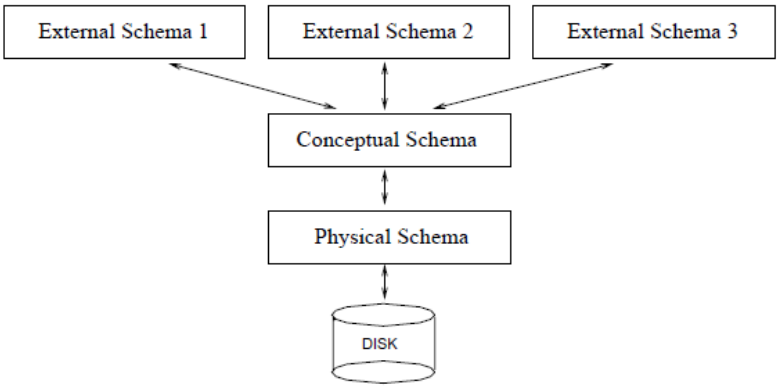
---

5. Describe the three tier architecture. What is the difference between logical and physical data independence.

The database description consists of a schema at each of these three levels of abstraction:

- **The Conceptual Schema**
- **The Physical Schema**
- **The External Schemas**

A data definition language (DDL) is used to define the external and conceptual schemas.



CONCEPTUAL SCHEMA

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

In sample university database, these relations contain information about **entities,** such as **students and faculty**, and about **relationships**, such as students' enrollment in courses. All **student entities** can be described using records in a **Students relation**.

Students(*sid:* `string`, *name:* `string`, *login:* `string`,
age: `integer`, *gpa:* `real`)
Faculty(*fid:* `string`, *fname:* `string`, *sal:* `real`)
Courses(*cid:* `string`, *cname:* `string`, *credits:* `integer`)
Rooms(*rno:* `integer`, *address:* `string`, *capacity:* `integer`)
Enrolled(*sid:* `string`, *cid:* `string`, *grade:* `string`)
Teaches(*fid:* `string`, *cid:* `string`)
Meets_In(*cid:* `string`, *rno:* `integer`, *time:* `string`)

PHYSICAL SCHEMA

The physical schema specifies additional storage details. Essentially, the physical schema summarizes how the relations described in the conceptual schema are actually **stored on secondary storage devices** such as disks and tapes. We must decide what file organizations to use to store the relations, and create **auxiliary data structures called indexes to speed up data retrieval operations.**

A sample physical schema for the university database follows:

- Store all relations as unsorted files of records.
- Create indexes on the first column of the Students, Faculty, and Courses relations, the sal column of Faculty, and the capacity column of Rooms.

EXTERNAL SCHEMA

External schemas, which usually are also in terms of the data model of the DBMS**, allow data access to be customized (and authorized)** at the level of individual users or groups of users. Any given database has **exactly one conceptual schema and one physical schema** because it has just one set of stored relations, but it **may have several external schemas,** each tailored to a particular group of users.

Data independence is achieved through use of the **three levels of data abstraction**; in particular, the **conceptual schema and the external schema** provide distinct benefits in this area. Thus, users can be **shielded from changes in the logical structure of the data**, or changes in the choice of relations to be stored. This property is called **logical data independence.** In turn, the conceptual **schema insulates users from changes in the physical storage of the data.** This property is referred to as **physical data independence.**

6.How to develop an ER diagram? Write steps for developing an ER Diagram with an example?

An **Entity Relationship Diagram (ERD)** is a graphical representation that shows how entities, such as people, objects, or concepts, relate to each other within a system. ER Diagrams are commonly used in database design to visualize the relationships between entities and their attributes. They help in understanding the logical structure of databases by showing entities' connections and relationships using symbols.

Some benefits of drawing ER diagrams for database design are:

1. It helps in understanding the data relationships.
2. ER diagrams are like a blueprint for designing a database.
3. Helps in communicating about databases with database designers, developers, users, etc.
4. They help in describing different relationships and operations within an organization.

Steps to draw an Entity Relation Diagram (ERD)

A step-by-step process to draw an entity relation diagram (ERD) is:

Step 1: Identifying Entities

Determine the main objects you want to represent in the database.

Step 2: Defining Attributes

Identify the properties(attributes) of properties of each entity. These attributes provide more details about an entity.

Step 3: Specifying Relationships

Create relationships between entities to specify how entities interact with each other.

Step 4: Drawing Entities

Draw entities as rectangle and write the name.

Step 5: Adding Attributes

To add attributes of an entity, write attributes inside the rectangle or connect them with lines.

Step 6: Connecting Entities

Draw lines between the related entities to represent their connection.

Step 7: Specifying Cardinality

Indicate the minimum and maximum number of relationship instances associated with an entity using notations like crow's foot.

Step 8: Organizing ER Diagram

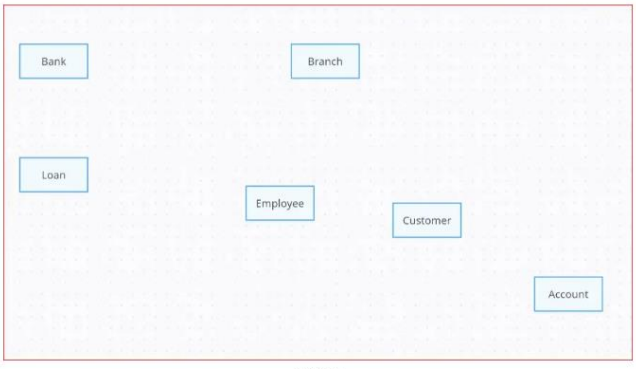
Organize all entities and relationships in a clean way for better readability and understanding.

Example: Entity Relationship Diagram Example for BANK

Defining Entities

Entities for Bank are:

Bank, Branch, Employee, customer, loan, account.



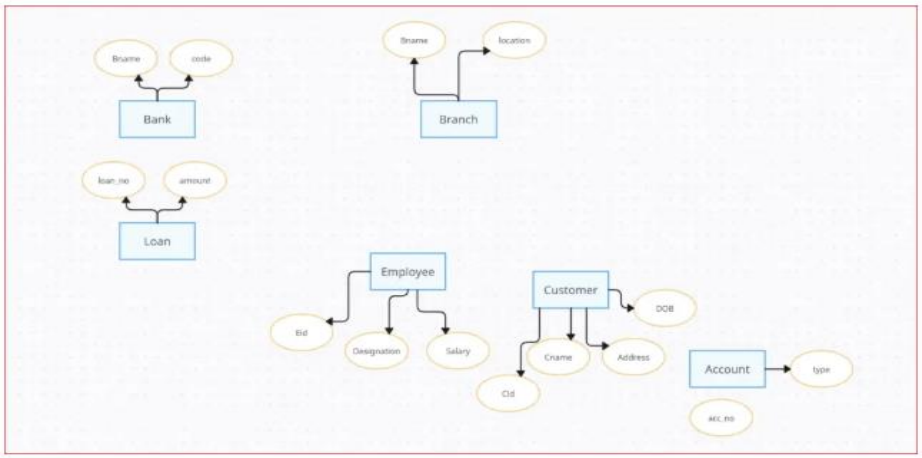
Entities

Step 2: Adding Attributes

Attributes are the kind of properties that describe the entities. They are represented by **ovals**.

Attributes for Bank are:

- For **Bank Entity** the Attributes are **Bname, code.**
- For **Branch Entity** the Attributes are **Blocation, Bname.**
- For **Employee Entity** the Attributes are **Eid, Designation, salary.**
- For **Customer Entity** the Attributes are **Cid, Cname, Address, DOB.**
- For **Loan Entity** the Attributes are **Loan_no, amount, rate.**
- For **Account Entity** the Attributes are **acc_no, type.**



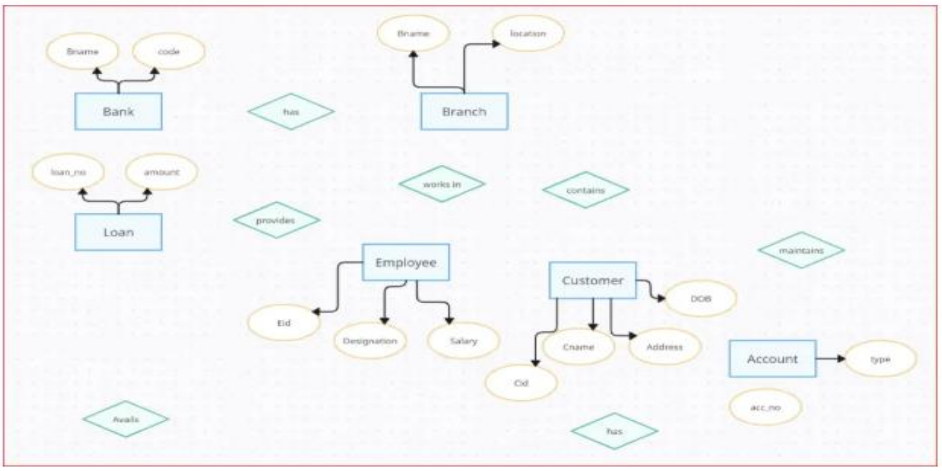
Attributes

Step 3: Establishing Relationships

Entities have some relationships with each other. Relationships define **how entities are associated with each other**.

Let's Establishing Relationships between them are:

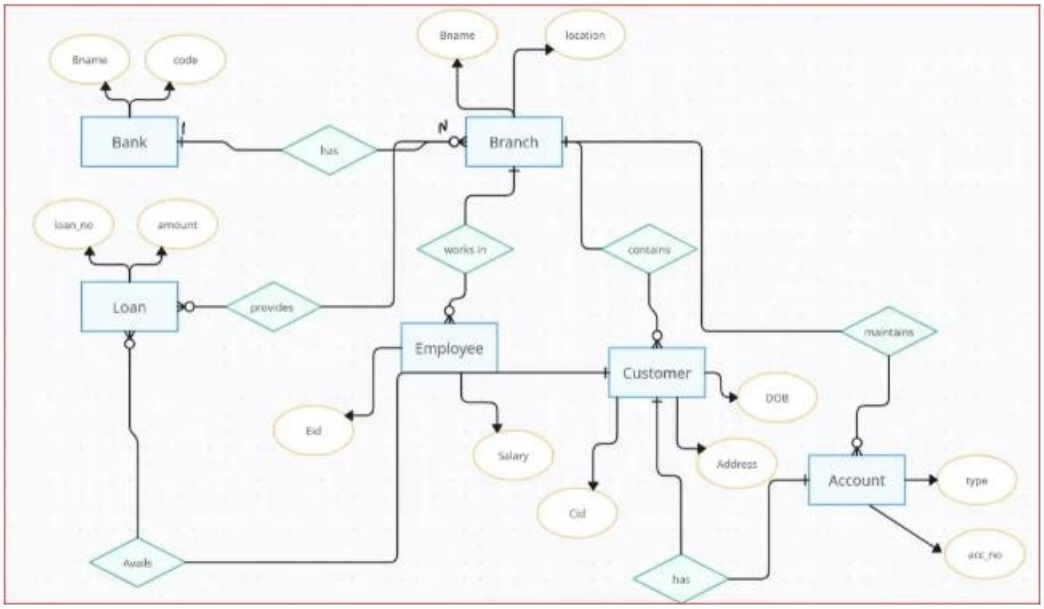
- The **Bank** has **branches**.
- The **Branch** provides **loan**.
- The **Employee** works in **branch**.
- The **Branch** contains **customers**.
- The **Customers** has **account**.
- The **Branch** maintains **account**.
- The **Customer** avails **loan**.



Relationships

Specifying Cardinality

- **Bank** and **branch** have **One to Many** relationships (a bank has multiple branches).
- **Branch** and **loan** have also **One to Many** relationship (a branch can provide many loans).
- **Branch** and **employee** have **One to Many** relationships (one branch has many employees).
- **Branch** and **account** have **One to Many** relationships (one branch has many accounts).
- **Branch** and **customer** have **Many to Many** relationships (multiple branches have multiple customers).
- **Customer** and **account** have **Many to Many** relationships (multiple customers have multiple accounts).
- **Customer** and **loan** have **Many to Many** relationships (multiple customers have multiple loans)



ER Diagram

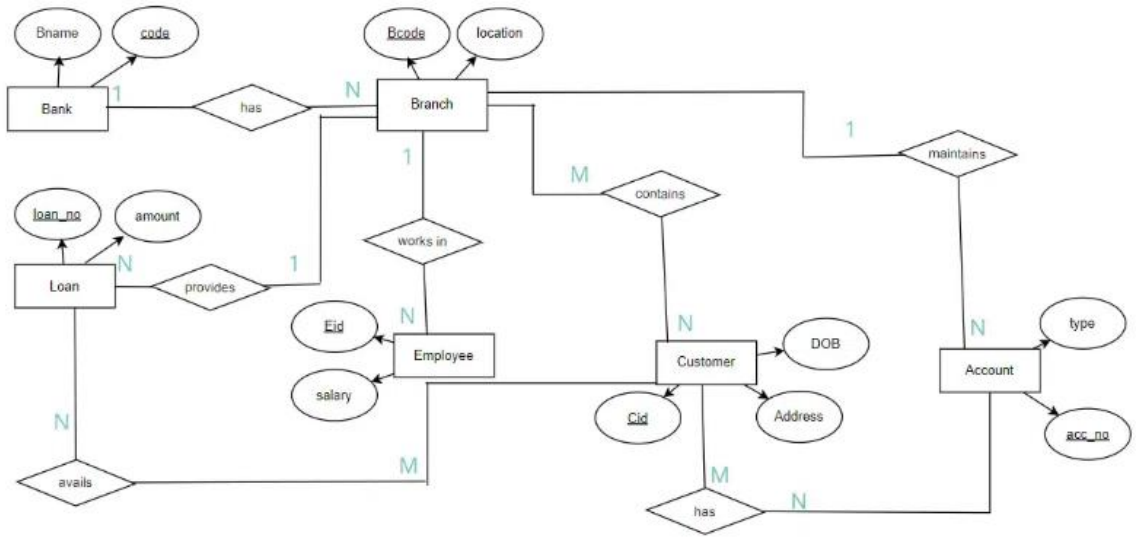
Identify Primary Keys

Primary keys are the **unique** identifier for each record in database table. It is denoted by an underline under the attribute name.

- The Primary key of **Bank** is **code**.
- The Primary key of **Branch** is **branch_code.**
- The Primary key of **Employee** is **Eid.**
- The Primary key of **Customer** is **Cid.**
- The Primary key of **Loan** is **loan_no**.
- The Primary key of **Account** is **acc_no**.

Final ER Diagram

The below diagram is our final entity relationship diagram for bank with all **entities**, their **attributes** and the relationship between them with the PRIMARY KEY and **Cardinality ratio**.



Final ER Diagram

---

7. Write a note on various types of end users who use DBMS. List out the responsibilities of a database administrator.

DBMS users generally access the database; it just specifies the different accounts associated with the user. This may require access for the user, and they are the ones who can take advantage of accessing the database and its data. DBMS, we have different types of users who may have different types of access depending on the requirement.

A primary goal of a database system is to retrieve information from and store new information into the database. People who work with a database can be categorized as **database users** or **database administrators.** There are four different types of database-system users, differentiated by the way they expect to interact with the system.

- **Naive Users**
- **Application Programmers**
- **Sophisticated Users**
- **Specialized Users**

NAIVE USERS

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For examples, Railway's ticket booking users are naive users. Clerks in any bank is a naive user because they don't have any DBMS knowledge but they still use the database and perform their given task.

APPLICATION PROGRAMMERS

Application programmers are computer professionals who write application programs. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports with minimal programming effort. They are free to write their code in any language such as Visual Basic, Developer, C, FORTRAN, COBOL etc. according to requirements.

SOPHISTICATED USERS

Sophisticated users interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Sophisticated users can be engineers, scientists, business analyst, who are familiar with the database.

SPECIALIZED USERS

Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types.

RESPONSIBILITIES OF AN DATABASE ADMINISTRATOR

One of the main reasons for using DBMS is to have central control of both the data and the programs that access those data. A person who has such central control over the system is called a **Database Administrator (DBA).** DBA is a person/team who defines the schema and also controls the 3 levels of database. A DBA is pivotal in ensuring the efficient, secure, and reliable operation of an organization's databases. The scope of their responsibilities is extensive and involves a variety of tasks aimed at maintaining the integrity and performance of database systems.

---

8. Define Entity and Entity set. Explain about Strong Entity Sets and Weak Entity Sets with examples.

The **Entity-Relationship (E-R)** data model was developed to facilitate database design by allowing **specification of an enterprise schema** that represents the **overall logical structure** of a database. The **E-R model** is very useful in **mapping the meanings and interactions** of real-world enterprises onto a **conceptual schema.**

An **Entity** is a **"thing" or "object" in the real world** that is distinguishable from all other objects. For example, each person in a university is an entity. An **entity has a set of properties**, and the values for some **set of properties may uniquely identify an entity.** For instance, a person may have a ***person_id*** property whose value **uniquely identifies** that person.

Example: *If we have a table of a Student (Roll_no, Student_name, Age, Mobile_no) then each student in that table is an entity and can be uniquely identified by their Roll Number i.e Roll_no.*

Student

Roll_no	Student_name	Age	Mobile_no
1	Andrew	18	7089117222
2	Angel	19	8709054568
3	Priya	20	9864257315
4	Analisa	21	9847852156

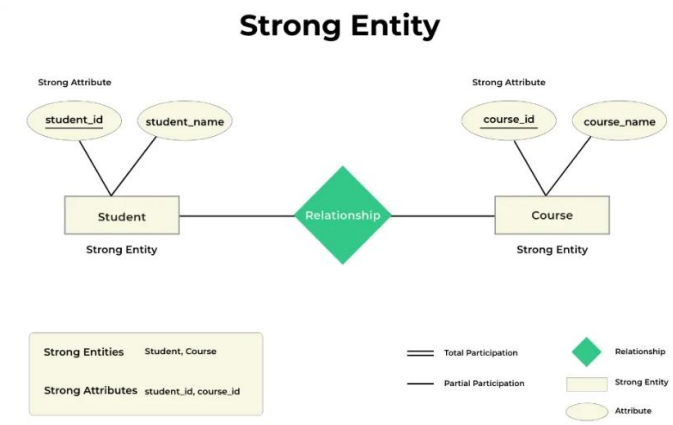
Entity

An **entity set** is a set of entities of the same type that share the same properties, or attributes. The set of all people who are instructors at a given university, for example, can be defined as the entity set instructor. Similarly, the entity set student might represent the set of all students in the university. Types of Entity Sets

- Strong Entity Set
- Weak Entity Set

Strong Entity Set

Strong entity are those entity types which has a key attribute. The primary key helps in identifying each entity uniquely. It is represented by a rectangle. In the above example, Roll_no identifies each element of the table uniquely and hence, we can say that STUDENT is a strong entity type. **The relationship between two strong entities** is represented by a **single diamond.** A Strong entity is nothing but an entity set having a primary key attribute or a table that consists of a primary key column



Weak Entity Set

An entity set that does not have sufficient attributes to form a primary key is termed a **weak entity set.** An entity set that has a primary key is termed a **strong entity set.** A weak entity can be identified uniquely only by **considering some of its attributes in conjunction with the primary key of another entity**, which is called **the identifying owner.**

Example : *If we have two tables of Customer(Customer_id, Name, Mobile_no, Age, Gender) and Address(Locality, Town, State, Customer_id). Here we cannot identify the address uniquely as there can be many customers from the same locality. So, for this, we need an attribute of Strong Entity Type i.e 'Customer' here to uniquely identify entities of 'Address' Entity Type.*