DBMS

1. What is meant by DBMS and what is its utility? Explain RDBMS with examples.

As the name suggests DBMS or Database Management System is a set of applications or programs that enable users to create and maintain a database. DBMS provides a tool or an interface for performing various operations such as inserting, deleting, updating, etc. into a database. It is software that enables the storage of data more compactly and securely as compared to a file-based system. A DBMS system helps a user to overcome problems like data inconsistency, data redundancy, etc. in a database and makes it more convenient and organized to use it.

RDBMS stands for Relational Database Management System and was introduced in the 1970s to access and store data more efficiently than DBMS. RDBMS stores data in the form of tables as compared to DBMS which stores data as files. Storing data as rows and columns makes it easier to locate specific values in the database and makes it more efficient as compared to DBMS.

2. Explain different languages present in DBMS.

Following are various languages present in DBMS:

- **DDL(Data Definition Language):** It contains commands which are required to define the database.
  E.g., CREATE, ALTER, DROP, TRUNCATE, RENAME, etc.
- **DML(Data Manipulation Language):** It contains commands which are required to manipulate the data present in the database.
  E.g., SELECT, UPDATE, INSERT, DELETE, etc.
- **DCL(Data Control Language):** It contains commands which are required to deal with the user permissions and controls of the database system.
  E.g., GRANT and REVOKE.
- **TCL(Transaction Control Language):** It contains commands which are required to deal with the transaction of the database.
  E.g., COMMIT, ROLLBACK, and SAVEPOINT.

3. What is meant by ACID properties in DBMS?

ACID stands for Atomicity, Consistency, Isolation, and Durability in a DBMS these are those properties that ensure a safe and secure way of sharing data among multiple users.

Atomicity (all or nothing)

By this, we mean that either the entire transaction takes place at once or doesn't happen at all.

—Abort: If a transaction aborts, changes made to database are not visible.

—Commit: If a transaction commits, changes made are visible.

Consistency

This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,

The total amount before and after the transaction must be maintained.

Total before T occurs = 500 + 200 = 700.

Total after T occurs = 400 + 300 = 700.

Therefore, database is consistent.

Isolation

This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. **Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.**

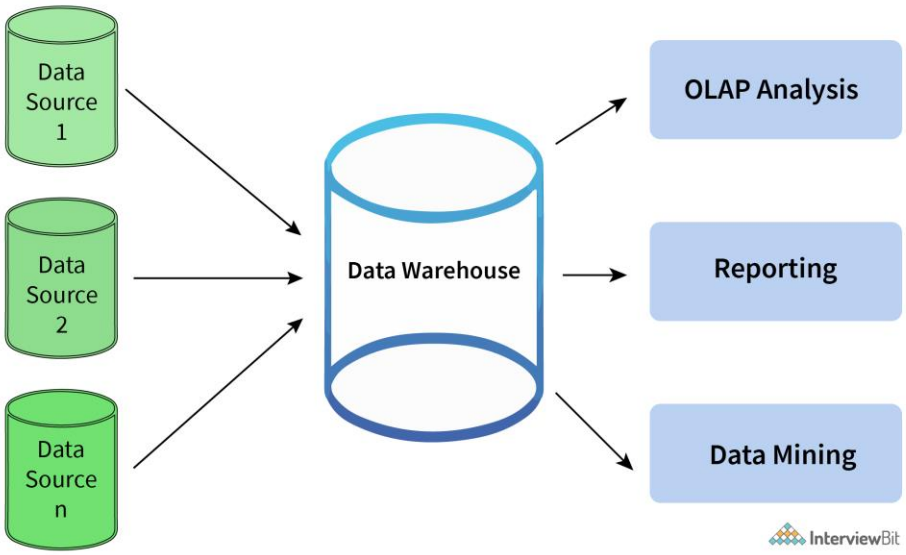
Transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability

This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

4. What is meant by Data Warehousing?

The process of collecting, extracting, transforming, and loading data from multiple sources and storing them into one database is known as data warehousing.



5. Different levels of Data Abstraction in a DBMS.

The process of hiding irrelevant details from users is known as data abstraction. Data abstraction can be divided into 3 levels:

- **Physical Level:** it is the lowest level and is managed by DBMS. This level consists of data storage descriptions and the details of this level are typically hidden from system admins, developers, and users.
- **Conceptual or Logical level:** it is the level on which developers and system admins work and it determines what data is stored in the database and what is the relationship between the data points.
- **External or View level:** it is the level that describes only part of the database and hides the details of the table schema and its physical storage from the users. The result of a query is an example of View level data abstraction. A view is a virtual table created by selecting fields from one or more tables present in the database.

Intension and extension in a database.

Intension: Description of the database.

Extension: Number of rows/tuples present in the database at a given point of time.

6. What is a lock. Explain the major difference between a shared lock and an exclusive lock during a transaction in a database.

A database lock is a mechanism to protect a shared piece of data from getting updated by two or more database users at the same time. When a single database user or session has

acquired a lock then no other database user or session can modify that data until the lock is released.

- **Shared Lock:** A shared lock is required for reading a data item and many transactions may hold a lock on the same data item in a shared lock. Multiple transactions are allowed to read the data items in a shared lock.
- **Exclusive lock:** An exclusive lock is a lock on any transaction that is about to perform a write operation. This type of lock doesn't allow more than one transaction and hence prevents any inconsistency in the database.

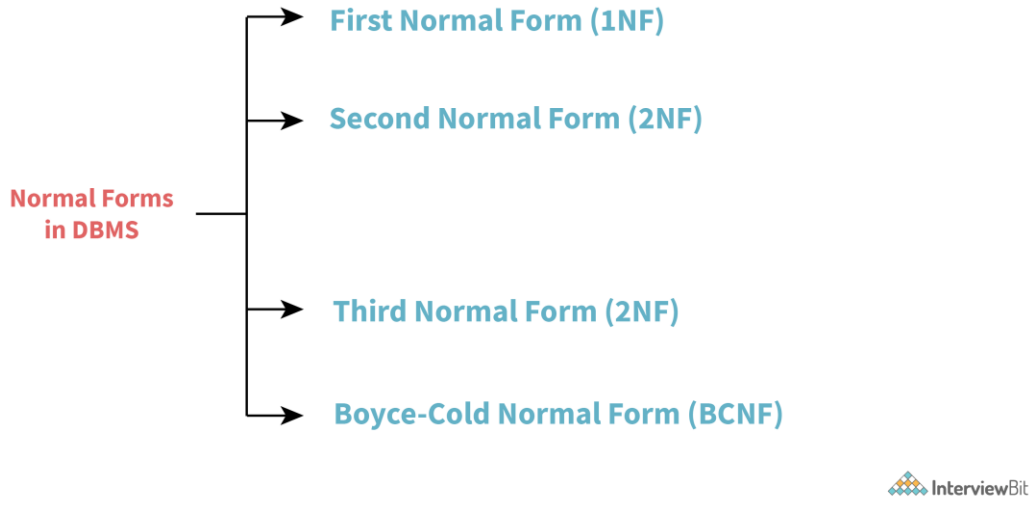
7. What is meant by normalization and denormalization?

Normalization is a process of reducing redundancy by organizing the data into multiple tables. Normalization leads to better usage of disk spaces and makes it easier to maintain the integrity of the database.

Denormalization is the reverse process of normalization as it combines the tables which have been normalized into a single table so that data retrieval becomes faster. JOIN operation allows us to create a denormalized form of the data by reversing the normalization.

Normal forms are used to eliminate or reduce redundancy in database tables.

Different types of Normalization forms in a DBMS.



Full Names	Physical Address	Movies Rented	Salutation
Janet Jones	First street Plot No 4	Pirates. the Caribbean Clash of the Titans	Ms.
Robert Phil	3rd street 34	Forgetting Sarah Marshal Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

InterviewBit

Table 1: Database Table

1NF Conditions:

- Every column must have a single value.
- Duplicate columns must be removed

Full Names	Physical Address	Movies Rented	Salutation
Janet Jones	First street Plot No 4	Pirates. the Caribbean	Ms.
Janet Jones	First street Plot No 4	Clash of the Titans	Ms.
Robert Phil	3rd street 34	Forgetting Sarah Marshal	Mr.
Robert Phil	3rd street 34	Daddy's Little Girls	Mr.
Robert Phil	5th Avenue	Clash of the Titans	Mr.

InterviewBit

Table 2: 1NF Normalized

2NF Conditions:

- Should be in 1NF.
- No Partial Dependency (If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.)
  If AB together is a candidate key and determines C then B alone or A alone cannot determine C. (i.e. if AB->C then A->C or B->C should not be possible)
  [Given that A alone and B alone are not candidate keys & C is a non-prime attribute]

3NF Conditions:

- Should be in 2NF.
- No Transitive Dependency

X->Y->Z (X is a candidate key which determines Y which in a non-prime attribute which determines Z, so indirectly X determines Z)

- No non-prime should determine non-prime.

BCNF(Boyce-Codd Normal form) Conditions:

- Should be in 3NF
- LHS must be Candidate Key or Super Key (i.e. if X->Y then X must be either candidate key or super key)

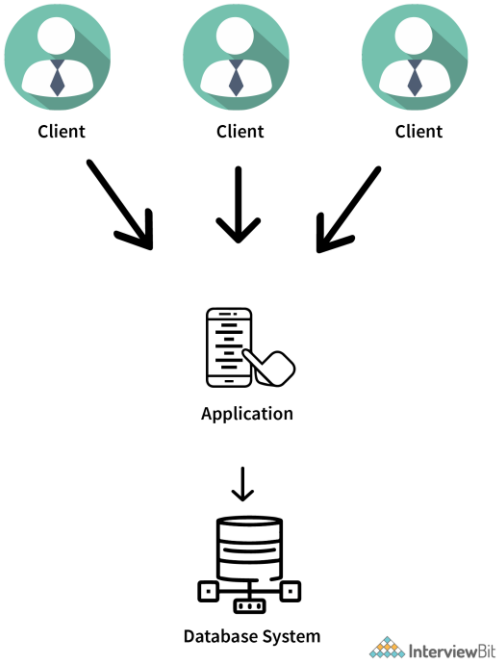
8. Different Types of Keys in a database

There are mainly 7 types of keys in a database:

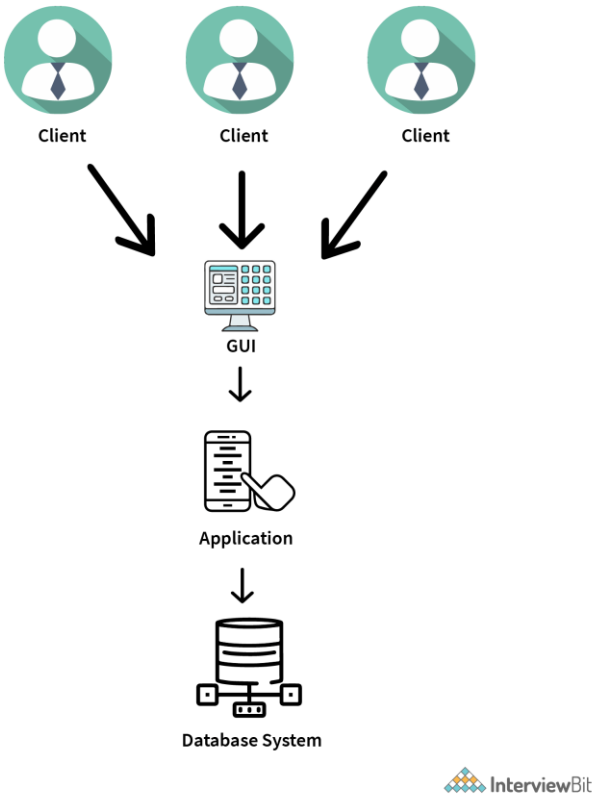
1. **Candidate Key:** The candidate key represents a set of properties that can uniquely identify a table. Each table may have multiple candidate keys. One key amongst all candidate keys can be chosen as a primary key.
2. **Super Key:** The super key defines a set of attributes that can uniquely identify a tuple. Candidate key and primary key are subsets of the super key, in other words, the super key is their superset. If a **Super Key** is made of more than one column it is also a **composite**.
3. **Primary Key:** The primary key defines a set of attributes that are used to uniquely identify every tuple. In the below example studentId and firstName are candidate keys and any one of them can be chosen as a Primary Key.
4. **Unique Key:** Essentially unique keys are primary keys with NULL values.
5. **Alternate Key:** All the candidate keys which are not chosen as primary keys are considered as alternate Keys.
6. **Foreign Key:** The foreign key defines an attribute that can only take the values present in one table common to the attribute present in another table. Foreign Key of one table should be primary key of another table.
7. **Composite Key:** A composite key refers to a combination of two or more columns that can uniquely identify each tuple in a table.

9. Difference between 2-tier and 3-tier architecture in DBMS

The **2-tier architecture** refers to the client-server architecture in which applications at the client end directly communicate with the database at the server end without any middleware involved.



The **3-tier architecture** contains another layer between the client and the server to provide GUI to the users and make the system much more secure and accessible.



SQL

1. Difference between SQL and MYSQL.

SQL is a standard language for retrieving and manipulating structured databases. On the contrary, MySQL is a relational database management system, like SQL Server, Oracle or IBM DB2, that is used to manage SQL databases.

Note: Fields are also known as rows or records or tuples in a table.

2. What are Constraints in SQL?

Constraints are used to specify the rules concerning data in the table. The constraints are:

- NOT NULL - Restricts NULL value from being inserted into a column.
- CHECK - Verifies that all values in a field satisfy a condition.
- DEFAULT - Automatically assigns a default value if no value has been specified for the field.
- UNIQUE - Ensures unique values to be inserted into the field.
- INDEX - Indexes a field providing faster retrieval of records.
- PRIMARY KEY - Uniquely identifies each record in a table.
- FOREIGN KEY - Ensures referential integrity for a record in another table.

Example of adding constraints:

```
CREATE TABLE students(
        id INT NOT NULL,
        name VARCHAR(255),
        branch VARCHAR(255),
        CONSTRAINT pk_id                    // Naming primary key (optional)
        PRIMARY KEY (id, name),   // single or multiple columns as PK
        UNIQUE (branch)
);
ALTER TABLE students ADD PRIMARY KEY (id);          // Alteration
```

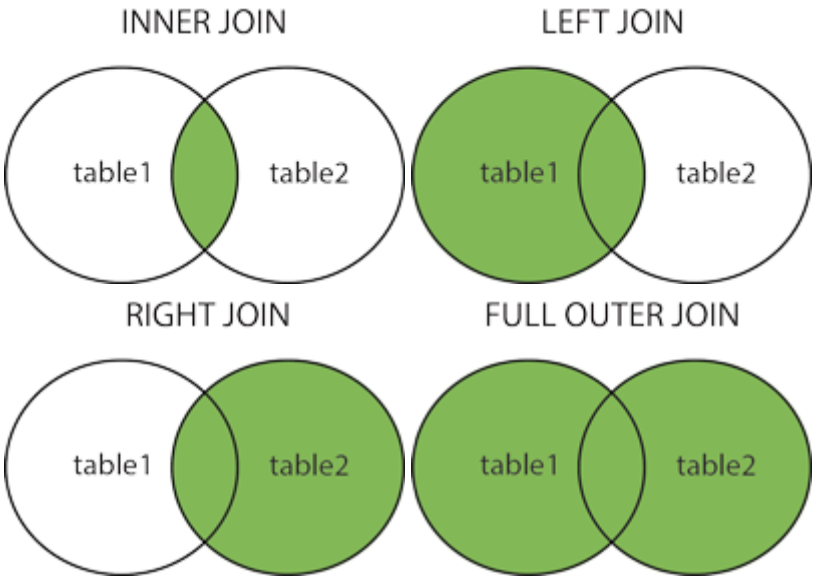
Foreign Key

```
CREATE TABLE students(
```

```
    ID INT NOT NULL PRIMARY KEY,
    Name varchar(255),
    LibraryID INT FOREIGN KEY (LibraryID) REFERENCES Library(LibraryID)
);
```

JOINS

The SQL Join clause is used to combine records (rows) from two or more tables in a SQL database based on a related column between the two.



There are four different types of JOINs in SQL:

INNER JOIN

Returns records that have matching values in both tables

Example:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

LEFT OUTER JOIN

Returns all records from the left table, and the matched records from the right table

Example: table1 is left table so all records are returned from this table and table2 is right table so only matching records are returned from table2.

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name
ORDER BY table1.column_name;
```

RIGHT OUTER JOIN

Returns all records from the right table, and the matched records from the left table.

Example: opposite of left-outer join

FULL OUTER JOIN

Returns all records when there is a match in either left or right table.

Example:

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;
```

SELF JOIN

A self JOIN is a case of regular join where a table is joined to itself based on some relation between its own column(s). Self-join uses the INNER JOIN or LEFT JOIN clause and a table alias is used to assign different names to the table within the query.

Example:

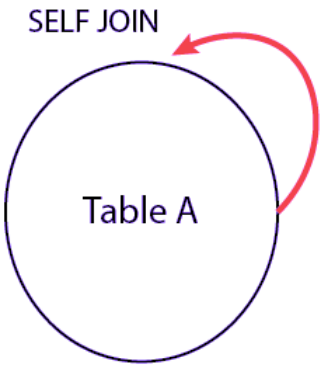
```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2, A.City
FROM Customers A, Customers B
WHERE A.CustomerID = B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

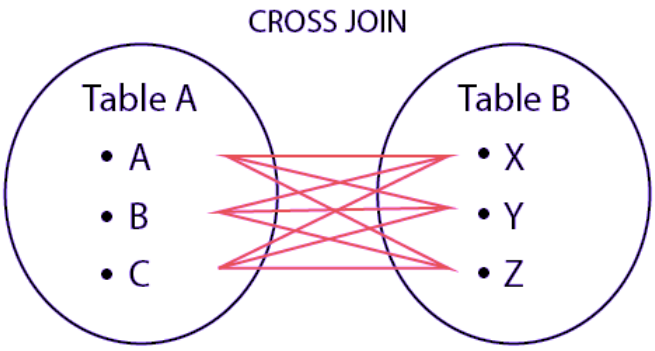
CROSS JOIN

Cartesian product of 2 tables included in the join.

Example:

```
SELECT stu.name, sub.subject
FROM students AS stu
CROSS JOIN subjects AS sub;
```





2. Difference between Clustered and Non-clustered index.

CLUSTERED INDEX	NON-CLUSTERED INDEX
Clustered index is faster.	Non-clustered index is slower.
Clustered index requires less memory for operations.	Non-Clustered index requires more memory for operations.
In clustered index, index is the main data.	In Non-Clustered index, index is the copy of data.
A table can have only one clustered index.	A table can have multiple non-clustered index.
Clustered index has inherent ability of storing data on the disk.	Non-Clustered index does not have inherent ability of storing data on the disk.
Clustered index store pointers to block not data.	Non-Clustered index store both value and a pointer to actual row that holds data.

CLUSTERED INDEX	NON-CLUSTERED INDEX
In Clustered index leaf nodes are actual data itself.	In Non-Clustered index leaf nodes are not the actual data itself rather they only contains included columns.
In Clustered index, Clustered key defines order of data within table.	In Non-Clustered index, index key defines order of data within index.
A Clustered index is a type of index in which table records are physically reordered to match the index.	A Non-Clustered index is a special type of index in which logical order of index does not match physical stored order of the rows on disk.

Select Query

```
SELECT fname, lname
FROM myDb.students
WHERE student_id = 1;
```

Update Query (Action Query)

```
UPDATE myDB.students
SET fname = 'Captain', lname = 'America'
WHERE student_id = 1;
```

Sub-query

```
SELECT name, email, mob, address
FROM myDb.contacts
WHERE roll_no IN (
        SELECT roll_no
        FROM myDb.students
        WHERE subject = 'Maths');
```

Some common SQL clauses used in conjuction with a SELECT query are as follows:

- **WHERE** clause in SQL is used to filter records that are necessary, based on specific conditions.

- **ORDER BY** clause in SQL is used to sort the records based on some field(s) in ascending (ASC) or descending order (DESC).
- **GROUP BY** clause in SQL is used to group records with identical data and can be used in conjuction with some aggregation functions to produce summarized results from the database.
- **HAVING** clause in SQL is used to filter records in combination with the GROUP BY clause. It is different from WHERE, since WHERE clause cannot filter aggregated records.
- Example:
  SELECT COUNT(studentId), country
  FROM myDB.students
  WHERE country != "INDIA"
  GROUP BY country
  HAVING COUNT(studentID) > 5;

AGGREGATE FUNCTIONS IN SQL

An aggregate function performs operations on a collection of values to return a single scalar value. Aggregate functions are often used with the GROUP BY and HAVING clauses of the SELECT statement.

- AVG() - Calculates the mean of a collection of values.
- COUNT() - Counts the total number of records in a specific table or view.
- MIN() - Calculates the minimum of a collection of values.
- MAX() - Calculates the maximum of a collection of values.
- SUM() - Calculates the sum of a collection of values.
- FIRST() - Fetches the first element in a collection of values.
- LAST() - Fetches the last element in a collection of values.
- Note: All aggregate functions described above ignore NULL values except for the COUNT function.

SCALAR FUNCTIONS IN SQL

A scalar function returns a single value based on the input value. Following are the widely used SQL scalar functions:

- LEN() - Calculates the total length of the given field (column).
- UCASE() - Converts a collection of string values to uppercase characters.
- LCASE() - Converts a collection of string values to lowercase characters.
- MID() - Extracts substrings from a collection of string values in a table.
- CONCAT() - Concatenates two or more strings.

- RAND() - Generates a random collection of numbers of given length.
- ROUND() - Calculates the round off integer value for a numeric field (or decimal point values).
- NOW() - Returns the current data & time.
- FORMAT() - Sets the format to display a collection of values.

PATTERN MATCHING IN SQL

SELECT * FROM students

WHERE first_name LIKE 'K%'

SELECT * FROM students

WHERE first_name NOT LIKE 'K%'

Specific position of a keyword (_ = 1 position)

SELECT * FROM students

WHERE first_name LIKE '__K%'

Specific Length

SELECT * FROM students

WHERE first_name LIKE '__K%' **(3 or more letters)**

SELECT * FROM students

WHERE first_name LIKE '___' **(exactly 4 letters)**

UNION, MINUS AND INTERSECT COMMANDS

The UNION operator combines and returns the result-set retrieved by two or more SELECT statements.

The MINUS operator in SQL is used to remove duplicates from the result-set obtained by the second SELECT query from the result-set obtained by the first SELECT query and then return the filtered results from the first.

The INTERSECT clause in SQL combines the result-set fetched by the two SELECT statements where records from one match the other and then returns this intersection of result-sets.

```
SELECT name FROM Students        /* Fetch the union of queries */
UNION
SELECT name FROM Contacts;


SELECT name FROM Students        /* Fetch the union of queries with duplicates*/
UNION ALL
SELECT name FROM Contacts;
```

```
SELECT name FROM Students        /* Fetch names from students */
MINUS                            /* that aren't present in contacts */
SELECT name FROM Contacts;
```

```
SELECT name FROM Students        /* Fetch names from students */
INTERSECT                        /* that are present in contacts as well */
SELECT name FROM Contacts;
```