SQL AND DBMS

MOST ASKED INTERVIEW QUESTIONS

MUST SAVE AND SHARE

Curated By- HIMANSHU KUMAR(LINKEDIN)

https://www.linkedin.com/in/himanshukumarmahuri

QUESTIONS COVERED: -

1. What are the Properties of RDBMS?
2. Describe ACID Properties
3. What are Keys in DBMS?
4. Difference between Vertical and Horizontal Scaling
5. What is Sharding?
6. What are SQL Commands? Explain types of SQL Commands
7. What is Indexing in DBMS?
8. Explain Normal Forms in DBMS
9. What are normalization and denormalization and why do we need them?
10. What do you mean by Conflict Serializability in DBMS?
11. Can Primary key contain two entities?
12. What are the Concurrency Control Protocols?
13. What are Nested Queries in SQL?
14. Explain types of JOINs in DBMS.
15. Difference between INNER and OUTER JOIN.
16. Write a SQL query to retrieve furniture from database whose dimensions(Width, Height, Length) match with the given dimension.
17. Write a SQL query to find the 4th maximum element from a table
18. Explain 3-Tier Architecture in DBMS
19. Explain Two tier architecture

What are the Properties of RDBMS?

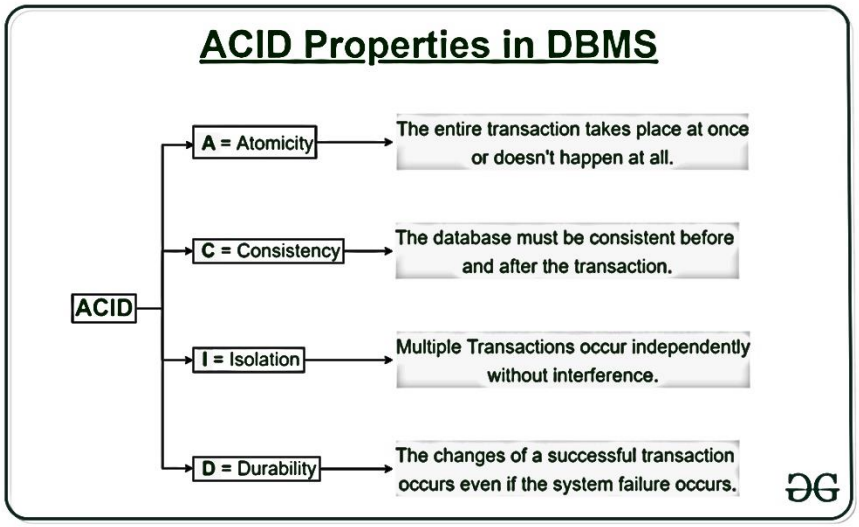
Relational Database Management Systems (RDBMS) are database management systems that maintain data records and indices in tables. Relationships may be created and maintained across and among the data and tables. In a relational database, relationships between data items are expressed by means of tables. Interdependencies among these tables are expressed by data values rather than by pointers. This allows for a high degree of data independence. An RDBMS has the capability to recombine the data items from different files, providing powerful tools for data usage.

Features/ Properties of RDBMS:

- Gives a high level of information security.
- It is quick and precise.
- Provides facility primary key, to exceptionally distinguish the rows.
- The values in RDBMS are atomic
- The values in a column of RDBMS are of the same type
- In RDMS, no two rows have the same data
- The sequence of Rows and Columns in RDBMS does not exist.
- Each column in RDBMS has a unique name.

Describe ACID Properties

A transaction is a single logical unit of work that accesses and possibly modifies the contents of a database. Transactions access data using read and write operations.
In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called **ACID** properties.



Atomicity

By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.

--**Abort**: If a transaction aborts, changes made to the database are not visible.

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

Atomicity is also known as the 'All or nothing rule'.

Consider the following transaction **T** consisting of **T1** and **T2**: Transfer of 100 from account **X** to account **Y**.

Before: X : 500	Y: 200
Transaction T	
T1	**T2**
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of **T1** but before completion of **T2**.(say, after **write(X)** but before **write(Y)**), then the amount has been deducted from **X** but not added to **Y**. This results in an inconsistent database state. Therefore, the transaction must be executed in its entirety in order to ensure the correctness of the database state.

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.

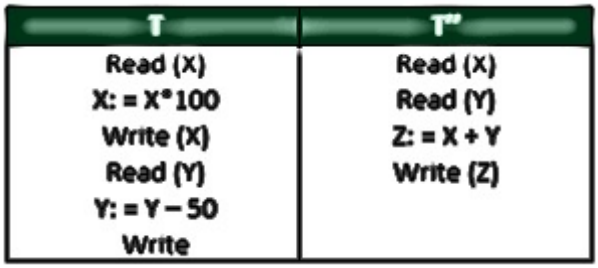
Total **before T** occurs = **500 + 200 = 700**.

Total **after T occurs** = **400 + 300 = 700**.

Therefore, the database is **consistent**. Inconsistency occurs in case **T1** completes but **T2** fails. As a result, T is incomplete.

Isolation This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of the 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. This property ensures that the execution of transactions concurrently will

result in a state that is equivalent to a state achieved these were executed serially in some order.
Let **X**= 500, **Y** = 500.
Consider two transactions **T** and **T''.**



T	T''
Read (X) X: = X*100 Write (X) Read (Y) Y: = Y – 50 Write	Read (X) Read (Y) Z: = X + Y Write (Z)

Suppose **T** has been executed till **Read (Y)** and then **T''** starts. As a result, interleaving of operations takes place due to which **T''** reads the correct value of **X** but the incorrect value of **Y** and sum computed by
T'': (X+Y = 50, 000+500=50, 500)
is thus not consistent with the sum at end of transaction: **T: (X+Y = 50, 000 + 450 = 50, 450)**.
This results in database inconsistency, due to a loss of 50 units. Hence, 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.

The **ACID** properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts as a single unit, produces consistent results, acts in isolation from other operations, and updates that it makes are durably stored.

What are Keys in DBMS?

The attribute which uniquely identifies each entity in the entity set is called a **key**.

For example, Roll_No will be unique for each student.

Different Types of Keys in Relational Model

STUDENT

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	9716271721	Haryana	India	20
2	RAM	9898291281	Punjab	India	19
3	SUJIT	7898291981	Rajsthan	India	18
4	SURESH		Punjab	India	21

Table 1

STUDENT_COURSE

STUD_NO	COURSE_NO	COURSE_NAME
1	C1	DBMS
2	C2	Computer Networks
1	C2	Computer Networks

Table 2

1. **Candidate Key:** The minimal set of an attribute that can uniquely identify a tuple is known as a candidate key. For Example, STUD_NO in STUDENT relation.

    - The value of the Candidate Key is unique and non-null for every tuple.
    - There can be more than one candidate key in a relation. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT.
    - The candidate key can be simple (having only one attribute) or composite as well. For Example, {STUD_NO, COURSE_NO} is a composite candidate key for relation STUDENT_COURSE.

    Note - In SQL Server a unique constraint that has a nullable column, **allows** the value '**null**' in that column **only once**. That's why STUD_PHONE attribute as a candidate here, but can not be 'null' values in the primary key attribute.

2. **Super Key:** The set of attributes that can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME), etc.

    - Adding zero or more attributes to the candidate key generates the super key.
    - A candidate key is a super key but vice versa is not true.

3. **Primary Key:** There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

   Alternate Key: The candidate key other than the primary key is called an alternate key. For Example, STUD_NO, as well as STUD_PHONE both, are candidate keys for relation STUDENT but STUD_PHONE will be an alternate key (only one out of many candidate keys).

4. **Foreign Key:** If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute and the relation which refers to the referenced relation is called referencing relation and the corresponding attribute is called referencing attribute. The referenced attribute of the referenced relation should be the primary key for it. For Example, STUD_NO in STUDENT_COURSE is a foreign key to STUD_NO in STUDENT relation.

Difference between Vertical and Horizontal Scaling

Difference between Horizontal and Vertical Scaling:

Horizontal Scaling	Vertical Scaling
When new server racks are added to the existing system to meet the higher expectation, it is known as horizontal scaling.	When new resources are added in the existing system to meet the expectation, it is known as vertical scaling
It expands the size of the existing system horizontally.	It expands the size of the existing system vertically.
It is difficult to implement	It is easy to implement
It is costlier, as new server racks comprise of a lot of resources	It is cheaper as we need to just add new resources
It takes more time to be done	It takes less time to be done

What is Sharding?

Sharding is a very important concept which helps the system to keep data into different resources according to the sharding process.

The word "**Shard**" means "**a small part of a whole**". Hence, Sharding means dividing a larger part into smaller parts.

In DBMS, Sharding is a type of Database partitioning in which a large Database is divided or partitioned into smaller data, also known as shards. These shards are not only smaller, but also faster and hence easily manageable.

Need for Sharding:

Consider a very large database whose sharding has not been done. For example, let's take a Database of a college in which all the student's records (present and past) in the whole college are maintained in a single database. So, it would contain a very very large number of data, say 100, 000 records.

Now when we need to find a student from this Database, each time around 100, 000 transactions have to be done to find the student, which is very very costly.

Now consider the same college students' records, divided into smaller data shards based on years. Each data shard will have around 1000-5000 student records only. So not only the database became much more manageable, but also the transaction cost of each time also reduces by a huge factor, which is achieved by Sharding.

Hence this is why Sharding is needed.

Features of Sharding:

- Sharding makes the Database smaller
- Sharding makes the Database faster
- Sharding makes the Database much more easily manageable
- Sharding can be a complex operation sometimes
- Sharding reduces the transaction cost of the Database

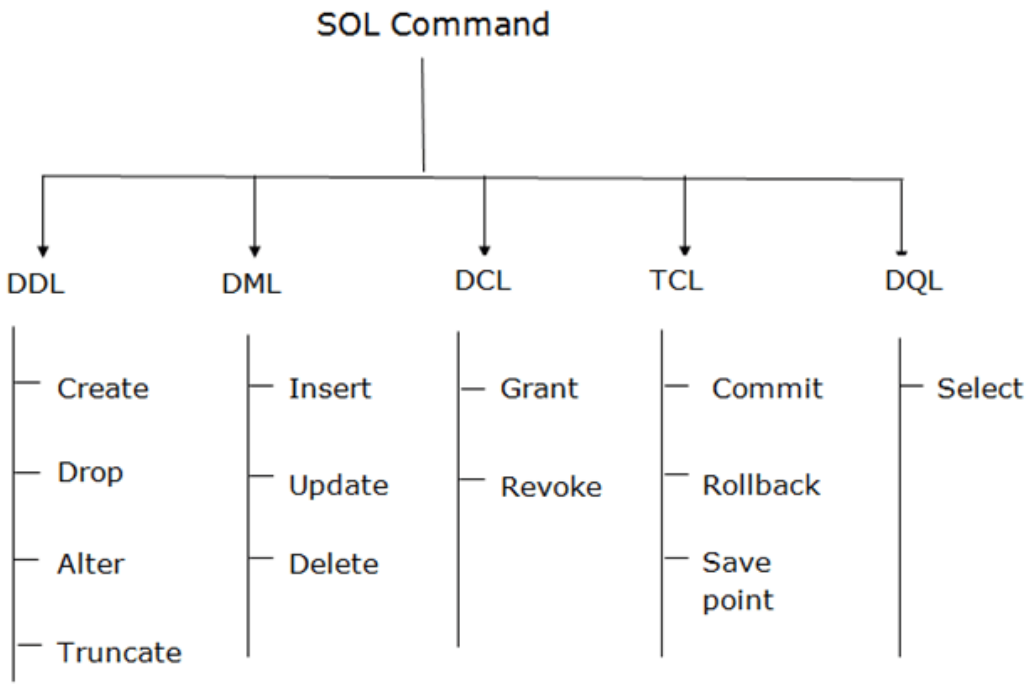
What are SQL Commands? Explain types of SQL Commands

Structured Query Language(SQL) as we all know is the database language by the use of which we can perform certain operations on the existing database, and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert, etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL - Data Definition Language
2. DQl - Data Query Language
3. DML - Data Manipulation Language
4. DCL - Data Control Language

Though many resources claim there to be another category of SQL clauses **TCL - Transaction Control Language**. So we will see in detail about TCL as well.



1. **DDL(Data Definition Language):** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Examples of DDL commands:

- CREATE - is used to create the database or its objects (like table, index, function, views, store procedure, and triggers).
- DROP - is used to delete objects from the database.
- ALTER -is used to alter the structure of the database.
- TRUNCATE–is used to remove all records from a table, including all spaces allocated for the records are removed.
- COMMENT –is used to add comments to the data dictionary.
- RENAME –is used to rename an object existing in the database.

2. **DQL (Data Query Language) :**

DML statements are used for performing queries on the data within schema objects. The purpose of the DQL Command is to get some schema relation based on the query passed to it.

Example of DQL:

- SELECT – is used to retrieve data from the database.

3. **DML(Data Manipulation Language):** The SQL commands that deal with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

Examples of DML:

- INSERT – is used to insert data into a table.
- UPDATE - is used to update existing data within a table.
- DELETE – is used to delete records from a database table.

4. **DCL(Data Control Language) :** DCL includes commands such as GRANT and REVOKE which mainly deal with the rights, permissions, and other controls of the database system.

Examples of DCL commands:

- **GRANT** -gives user's access privileges to the database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

5. **TCL(transaction Control Language):** TCL commands deal with the transaction within the database.

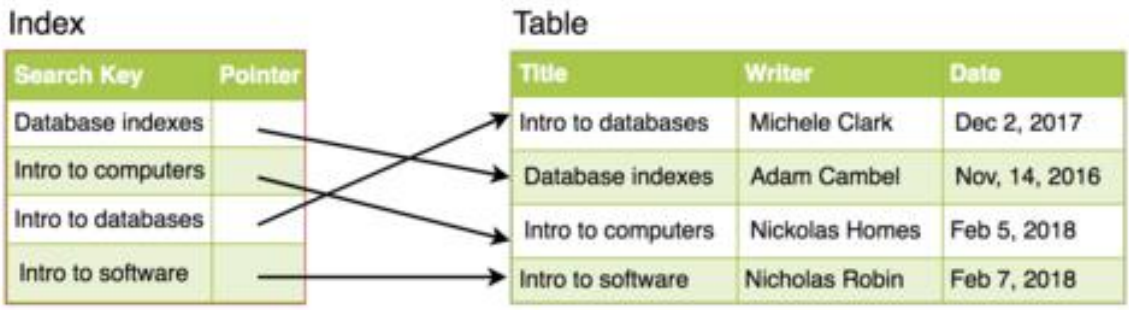
   Examples of TCL commands:
   - **COMMIT**– commits a Transaction.
   - **ROLLBACK**– rollbacks a transaction in case of any error occurs.
   - **SAVEPOINT** –sets a savepoint within a transaction.
   - **SET TRANSACTION** –specify characteristics for the transaction.

What is Indexing in DBMS?

Indexing is a way to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. It is a data structure technique that is used to quickly locate and access the data in a database.

Indexes are created using a few database columns.

- The first column is the **Search key** that contains a copy of the primary key or candidate key of the table. These values are stored in sorted order so that the corresponding data can be accessed quickly.
  Note: The data may or may not be stored in sorted order.
- The second column is the **Data Reference** or **Pointer** which contains a set of pointers holding the address of the disk block where that particular key value can be found.



The indexing has various attributes:

- **Access Types**: This refers to the type of access such as value-based search, range access, etc.
- **Access Time**: It refers to the time needed to find a particular data element or set of elements.

- **Insertion Time**: It refers to the time taken to find the appropriate space and insert new data.
- **Deletion Time**: Time taken to find an item and delete it as well as update the index structure.
- **Space Overhead**: It refers to the additional space required by the index.

Explain Normal Forms in DBMS

Normalization is the process of minimizing **redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion, and updation anomalies. So, it helps to minimize the redundancy in relations. **Normal forms** are used to eliminate or reduce redundancy in database tables.

1. First Normal Form -

If a relation contains a composite or multi-valued attribute, it violates the first normal form or the relation is in the first normal form if it does not contain any composite or multi-valued attribute. A relation is in first normal form if every attribute in that relation is **singled valued attribute**.

- **Example 1 -** Relation STUDENT in table 1 is not in 1NF because of multi-valued attribute STUD_PHONE. Its decomposition into 1NF has been shown in table 2.

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721, 9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 1

Conversion to first normal form

STUD_NO	STUD_NAME	STUD_PHONE	STUD_STATE	STUD_COUNTRY
1	RAM	9716271721	HARYANA	
1	RAM	9871717178	HARYANA	INDIA
2	RAM	9898297281	PUNJAB	INDIA
3	SURESH		PUNJAB	INDIA

Table 2

- **Example 2 -**

```
ID   Name   Courses

-----------------

1    A      c1, c2

2    E      c3

3    M      C2, c3
```

In the above table, Course is a multi-valued attribute so it is not in 1NF.

The below Table is in 1NF as there is no multi-valued attribute

```
ID   Name   Course
------------------
1    A       c1
1    A       c2
2    E       c3
3    M       c2
3    M       c3
```

2. Second Normal Form -

To be in second normal form, a relation must be in first normal form and the relation must not contain any partial dependency. A relation is in 2NF if it has **No Partial Dependency,** i.e., no non-prime attribute (attributes which are not part of any candidate key) is dependent on any proper subset of any candidate key of the table.

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

- **Example 1 -** Consider table-3 as following below.

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

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

Here,
COURSE_FEE cannot alone decide the value of COURSE_NO or STUD_NO;
COURSE_FEE together with STUD_NO cannot decide the value of COURSE_NO;
COURSE_FEE together with COURSE_NO cannot decide the value of STUD_NO;
Hence,
COURSE_FEE would be a non-prime attribute, as it does not belong to the one only candidate key {STUD_NO, COURSE_NO} ;
But, COURSE_NO -> COURSE_FEE, i.e., COURSE_FEE is dependent on COURSE_NO, which is a proper subset of the candidate key. Non-prime attribute COURSE_FEE is dependent on a proper subset of the candidate key, which is a partial dependency and so this relation is not in 2NF.

To convert the above relation to 2NF,
we need to split the table into two tables such as :
Table 1: STUD_NO, COURSE_NO
Table 2: COURSE_NO, COURSE_FEE

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

NOTE: 2NF tries to reduce the redundant data getting stored in memory. For instance, if there are 100 students taking the C1 course, we don't need to store its Fee as 1000 for all the 100 records, instead, once we can store it in the second table as the course fee for C1 is 1000.

- **Example 2 -** Consider following functional dependencies in relation R (A, B , C, D)

```
AB -> C  [A and B together determine C]
```

```
BC -> D  [B and C together determine D]
```

In the above relation, AB is the only candidate key and there is no partial dependency, i.e., any proper subset of AB doesn't determine any non-prime attribute.

3. Third Normal Form -

A relation is in third normal form, if there is **no transitive dependency** for non-prime attributes as well as it is in second normal form.
A relation is in 3NF if **at least one of the following condition holds** in every non-trivial function dependency X --> Y

0. X is a super key.
1. Y is a prime attribute (each element of Y is part of some candidate key).

STUD_NO	STUD_NAME	STUD_STATE	STUD_COUNTRY	STUD_AGE
1	RAM	HARYANA	INDIA	20
2	RAM	PUNJAB	INDIA	19
3	SURESH	PUNJAB	INDIA	21

Table 4

Transitive dependency - If A->B and B->C are two FDs then A->C is called transitive dependency.

- **Example 1 -** In relation STUDENT given in Table 4,

  FD set: {STUD_NO -> STUD_NAME, STUD_NO -> STUD_STATE, STUD_STATE -> STUD_COUNTRY, STUD_NO -> STUD_AGE}
  Candidate Key: {STUD_NO}

  For this relation in table 4, STUD_NO -> STUD_STATE and STUD_STATE -> STUD_COUNTRY are true. So STUD_COUNTRY is transitively dependent on STUD_NO. It violates the third normal form. To convert it in third normal form, we will decompose the relation STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_COUNTRY_STUD_AGE) as:
  STUDENT (STUD_NO, STUD_NAME, STUD_PHONE, STUD_STATE, STUD_AGE)
  STATE_COUNTRY (STATE, COUNTRY)

- **Example 2 -** Consider relation R(A, B, C, D, E)
  A -> BC,
  CD -> E,
  B -> D,
  E -> A
  All possible candidate keys in the above relation are {A, E, CD, BC} All attributes are on the right sides of all functional dependencies are prime.

4. Boyce-Codd Normal Form (BCNF) -

A relation R is in BCNF if R is in Third Normal Form and for every FD, LHS is super key. A relation is in BCNF iff in every non-trivial functional dependency X --> Y, X is a super key.

Example 1 - Find the highest normal form of a relation R(A,B,C,D,E) with FD set as {BC->D, AC->BE, B->E}
Step 1. As we can see, (AC)+ ={A,C,B,E,D} but none of its subset can determine all attribute of relation, So AC will be candidate key. A or C can't be derived from any other attribute of the relation, so there will be only 1 candidate key {AC}.
Step 2. Prime attributes are those attribute which are part of candidate key {A, C} in this example and others will be non-prime {B, D, E} in this example.
Step 3. The relation R is in 1st normal form as a relational DBMS does not allow multi-valued or composite attribute.
The relation is in 2nd normal form because BC->D is in 2nd normal form (BC is not a proper subset of candidate key AC) and AC-> BE is in 2nd normal form (AC is candidate key) and B-> E is in 2nd normal form (B is not a proper subset of candidate key AC).
The relation is not in 3rd normal form because in BC->D (neither BC is a super key nor D is a prime attribute) and in B-> E (neither B is a super key nor E is a prime attribute) but to satisfy 3rd normal for, either LHS of an FD should be super key or RHS should be a prime attribute.
So the highest normal form of relation will be the 2nd Normal form.

Example 2 -For example consider relation R(A, B, C)
A -> BC,
B ->
A and B both are super keys so above relation is in BCNF.

Key Points -

BCNF is free from redundancy.

If a relation is in BCNF, then 3NF is also also satisfied.

If all attributes of relation are prime attribute, then the relation is always in 3NF.

A relation in a Relational Database is always and at least in 1NF form.

Every Binary Relation (a Relation with only 2 attributes) is always in BCNF.

If a Relation has only singleton candidate keys(i.e. every candidate key consists of only 1 attribute), then the Relation is always in 2NF(because no Partial functional dependency possible).

Sometimes going for BCNF form may not preserve functional dependency. In that case go for BCNF only if the lost FD(s) is not required, else normalize till 3NF only.

There are many more Normal forms that exist after BCNF, like 4NF and more. But in real world database systems it's generally not required to go beyond BCNF.

Exercise 1: Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

```
ABC --> D
CD --> AE
```

Important Points for solving above type of question.
1) It is always a good idea to start checking from BCNF, then 3 NF, and so on.
2) If any functional dependency satisfied a normal form then there is no need to check for lower normal form. For example, ABC --> D is in BCNF (Note that ABC is a superkey), so no need to check this dependency for lower normal forms.

Candidate keys in the given relation are {ABC, BCD}

BCNF: ABC -> D is in BCNF. Let us check CD -> AE, the CD is not a super key so this dependency is not in BCNF. So, R is not in BCNF.

3NF: ABC -> D we don't need to check for this dependency as it already satisfied

BCNF. Let us consider CD -> AE. Since E is not a prime attribute, so the relation is not in 3NF.

2NF: In 2NF, we need to check for partial dependency. CD is a proper subset of a candidate key and it determines E, which is a non-prime attribute. So, the given relation is also not in 2 NF. So, the highest normal form is 1 NF.

What are normalization and denormalization and why do we need them?

Normalization: Normalization is the method used in a database to reduce the data redundancy and data inconsistency from the table. It is the technique in which Non-redundancy and consistency data are stored in the set schema. By using normalization the number of tables is increased instead of decreased.

Denormalization: Denormalization is also the method that is used in a database. It is used to add redundancy to execute the query quickly. It is a technique in which data are combined to execute the query quickly. By using denormalization the number of tables is decreased which oppose to the normalization.

Difference between Normalization and Denormalization:

S.NO	Normalization	Denormalization
1.	In normalization, Non-redundancy and consistency data are stored in the set schema.	In denormalization, data are combined to execute the query quickly.
2.	In normalization, Data redundancy and inconsistency are reduced.	In denormalization, redundancy is added for the quick execution of queries.
3.	Data integrity is maintained in normalization.	Data integrity is not maintained in denormalization.
4.	In normalization, redundancy is reduced or eliminated.	In denormalization, redundancy is added instead of reduction or elimination of redundancy.
5.	Number of tables in normalization is increased.	Denormalization, Number of tables in decreased.
6.	Normalization optimizes the uses of disk spaces.	Denormalization does not optimize the disk spaces.

What do you mean by Conflict Serializability in DBMS?

As discussed in Concurrency control, serial schedules have less resource utilization and low throughput. To improve it, two are more transactions are run concurrently. But concurrency of transactions may lead to inconsistency in the database. To avoid this, we need to check whether these concurrent schedules are serializable or not.

Conflict Serializable: A schedule is called conflict serializable if it can be transformed into a serial schedule by swapping non-conflicting operations.

Conflicting operations: Two operations are said to be conflicting if all conditions satisfy:

- They belong to different transactions
- They operate on the same data item
- At Least one of them is a write operation

Example: -

- **Conflicting** operations pair ($R_1(A)$, $W_2(A)$) because they belong to two different transactions on same data item A and one of them is write operation.
- Similarly, ($W_1(A)$, $W_2(A)$) and ($W_1(A)$, $R_2(A)$) pairs are also **conflicting**.
- On the other hand, ($R_1(A)$, $W_2(B)$) pair is **non-conflicting** because they operate on the different data item.
- Similarly, (($W_1(A)$, $W_2(B)$) pair is **non-conflicting.**

Can Primary key contain two entities?

Primary Key is a set of attributes (or attribute) that uniquely identify the tuples in relation or table. The primary key is a minimal super key, so **there is one and only one primary key in any relationship**.

For example,

```
Student{ID, F_name, M_name, L_name, Age}
```

Here only **ID** can be the primary key because the name, age, and address can be the same but ID can't be same.

What are the Concurrency Control Protocols?

Concurrency control is provided in a database to:

- (i) enforce isolation among transactions.
- (ii) preserve database consistency through consistency preserving execution of transactions.
- (iii) resolve read-write and write-read conflicts.

Various concurrency control techniques are:

```
1. Two-phase locking Protocol

2. Time stamp ordering Protocol

3. Multi-version concurrency control

4. Validation concurrency control
```

These are briefly explained below.

1. Two-Phase Locking ProtocolLocking is an operation that secures: permission to read, OR permission to write a data item. Two-phase locking is a process used to gain ownership of shared resources without creating the possibility of deadlock.

   The 3 activities taking place in the two-phase update algorithm are:

```
2. (i). Lock Acquisition

3. (ii). Modification of Data

(iii). Release Lock
```

Two-phase locking prevents deadlock from occurring in distributed systems by releasing all the resources it has acquired if it is not possible to acquire all the resources required without waiting for another process to finish using a lock. This means that no process is ever in a state where it is holding some shared resources, and waiting for another process to release a shared resource that it requires. This means that deadlock cannot occur due to resource contention.

A transaction in the Two-Phase Locking Protocol can assume one of the 2 phases:

- **(i) Growing Phase:** In this phase, a transaction can only acquire locks but cannot release any lock. The point when a transaction acquires all the locks it needs is called the Lock Point.
- **(ii) Shrinking Phase:** In this phase, a transaction can only release locks but cannot acquire any.

Time Stamp Ordering Protocol: A timestamp is a tag that can be attached to any transaction or any data item, which denotes a specific time on which the transaction or the data item had been used in any way. Timestamp can be implemented in 2 ways. One is to directly assign the current value of the clock to the transaction or data item. The other is to attach the value of a logical counter that keeps increment as new timestamps are required.

The timestamp of a data item can be of 2 types:

- **(i) W-timestamp(X):** This means the latest time when the data item X has been written into.

- **(ii) R-timestamp(X):** This means the latest time when the data item X has been read from. These 2 timestamps are updated each time a successful read/write operation is performed on the data item X.

Multiversion Concurrency Control: Multiversion schemes keep old versions of data items to increase concurrency.

Multiversion 2 phase locking: Each successful write results in the creation of a new version of the data item written. Timestamps are used to label the versions. When a read(X) operation is issued, select an appropriate version of X based on the timestamp of the transaction.

Validation Concurrency Control: The optimistic approach is based on the assumption that the majority of the database operations do not conflict. The optimistic approach requires neither locking nor time stamping techniques. Instead, a transaction is executed without restrictions until it is committed. Using an optimistic approach, each transaction moves through 2 or 3 phases, referred to like read, validation, and write.

- **(i)** During the read phase, the transaction reads the database, executes the needed computations, and makes the updates to a private copy of the database values. All update operations of the transactions are recorded in a temporary update file, which is not accessed by the remaining transactions.

- **(ii)** During the validation phase, the transaction is validated to ensure that the changes made will not affect the integrity and consistency of the database. If the validation test is positive, the transaction goes to a write phase. If the validation test is negative, the transaction is restarted and the changes are discarded.

- **(iii)** During the write phase, the changes are permanently applied to the database.

What are Nested Queries in SQL?

In nested queries, a query is written inside a query. The result of the inner query is used in the execution of the outer query. We will use **STUDENT, COURSE, STUDENT_COURSE** tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

There are mainly two types of nested queries:

- **Independent Nested Queries:** In independent nested queries, query execution starts from innermost query to outermost queries. The execution of the inner query is independent of the outer query, but the result of the inner query is used in the execution of the outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

  IN: If we want to find out **S_ID** who are enrolled in **C_NAME** 'DSA' or 'DBMS', we can write it with the help of independent nested query and IN operator. From **COURSE** table, we can find out **C_ID** for **C_NAME** 'DSA' or DBMS' and we can use these **C_ID**s for finding **S_ID**s from **STUDENT_COURSE** TABLE.

  STEP 1: Finding **C_ID** for **C_NAME** ='DSA' or 'DBMS'

  Select **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME** = 'DBMS'

  STEP 2: Using **C_ID** of step 1 for finding **S_ID**
  Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

  (SELECT **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME**='DBMS');

  The inner query will return a set with members C1 and C3 and the outer query will return those **S_ID**s for which **C_ID** is equal to any member of the set (C1 and C3 in this case). So, it will return S1, S2 and S4.

  Note: If we want to find out names of **STUDENT**s who have either enrolled in 'DSA' or 'DBMS', it can be done as:

  Select S_NAME from **STUDENT** where **S_ID** IN

  (Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

  (SELECT **C_ID** from **COURSE** where **C_NAME**='DSA' or **C_NAME**='DBMS'));

  NOT IN: If we want to find out **S_ID**s of **STUDENT**s who have neither enrolled in 'DSA' nor in 'DBMS', it can be done as:

  Select **S_ID** from **STUDENT** where **S_ID** NOT IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**='DSA' or **C_NAME**='DBMS'));

The innermost query will return a set with members C1 and C3. Second inner query will return those **S_ID**s for which **C_ID** is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those **S_ID**s where **S_ID** is not a member of set (S1, S2 and S4). So it will return S3.

Co-related Nested Queries: In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out **S_NAME** of **STUDENT**s who are enrolled in **C_ID** 'C1', it can be done with the help of co-related nested query as:

Select S_NAME from **STUDENT** S where EXISTS

(select * from **STUDENT_COURSE** SC where S.**S_ID**=SC.**S_ID** and SC.**C_ID**='C1');

For each row of **STUDENT** S, it will find the rows from **STUDENT_COURSE** where S.**S_ID** = SC.**S_ID** and SC.**C_ID**='C1'. If for a **S_ID** from **STUDENT** S, atleast a row exists in **STUDENT_COURSE** SC with **C_ID**='C1', then inner query will return true and corresponding **S_ID** will be returned as output.

Explain types of JOINs in DBMS.

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL JOIN

Consider the two tables below:

Student

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	HARSH	DELHI	XXXXXXXXXX	18
2	PRATIK	BIHAR	XXXXXXXXXX	19
3	RIYANKA	SILIGURI	XXXXXXXXXX	20
4	DEEP	RAMNAGAR	XXXXXXXXXX	18
5	SAPTARHI	KOLKATA	XXXXXXXXXX	19
6	DHANRAJ	BARABAJAR	XXXXXXXXXX	20
7	ROHIT	BALURGHAT	XXXXXXXXXX	18
8	NIRAJ	ALIPUR	XXXXXXXXXX	19

StudentCourse

COURSE_ID	ROLL_NO
1	1
2	2
2	3
3	4
1	5
4	9
5	10
4	11

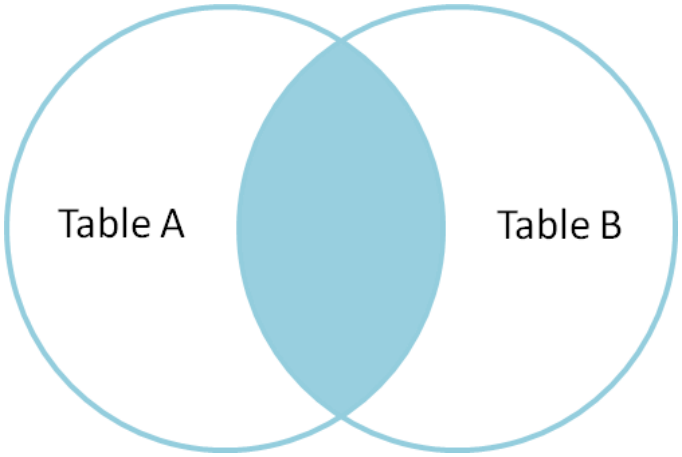
The simplest Join is INNER JOIN.

1. **INNER JOIN:** The INNER JOIN keyword selects all rows from both the tables as long as the condition satisfies. This keyword will create the result-set by combining all rows from both the tables where the condition satisfies i.e., value of the common field will be the same.

Syntax:

```
2. SELECT table1.column1,table1.column2,table2.column1,....

3. FROM table1

4. INNER JOIN table2

5. ON table1.matching_column = table2.matching_column;
6. table1: First table.

7. table2: Second table

8. matching_column: Column common to both the tables.
```

Note: We can also write JOIN instead of INNER JOIN. JOIN is the same as INNER JOIN.



Example Queries(INNER JOIN)

- This query will show the names and age of students enrolled in different courses.

```
SELECT StudentCourse.COURSE_ID, Student.NAME, Student.AGE F
ROM Student

INNER JOIN StudentCourse
```

```
o  ON Student.ROLL_NO = StudentCourse.ROLL_NO;
```

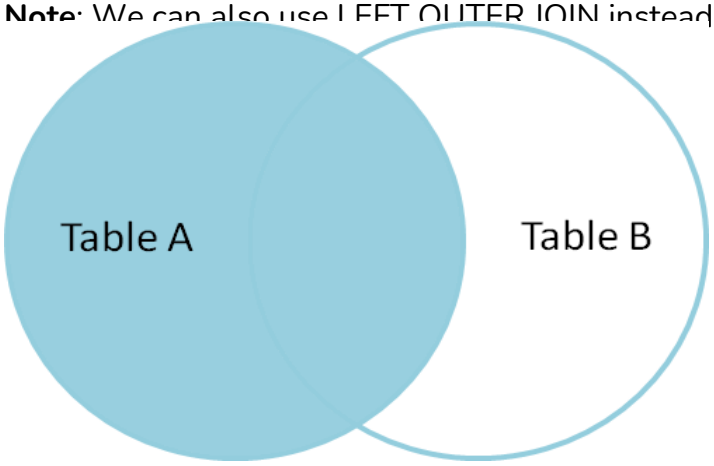
Output:

COURSE_ID	NAME	Age
1	HARSH	18
2	PRATIK	19
2	RIYANKA	20
3	DEEP	18
1	SAPTARHI	19

9. **LEFT JOIN**: This join returns all the rows of the table on the left side of the join and matching rows for the table on the right side of the join. The rows for which there is no matching row on the right side, the result-set will contain *null*. LEFT JOIN is also known as LEFT OUTER JOIN. **Syntax:**

```
10. SELECT table1.column1,table1.column2,table2.column1,....

11. FROM table1

12. LEFT JOIN table2

13. ON table1.matching_column = table2.matching_column;
14. table1: First table.

15. table2: Second table

16. matching_column: Column common to both the tables.
```

Note: We can also use LEFT OUTER JOIN instead of LEFT JOIN, both are same.



Example Queries(LEFT JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

LEFT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*

17. **RIGHT JOIN**: RIGHT JOIN is similar to LEFT JOIN. This join returns all the rows of the table on the right side of the join and matching rows for the table on the left side of join. The rows for which there is no matching row on the

left side, the result-set will contain *null*. RIGHT JOIN is also known as RIGHT OUTER JOIN.**Syntax:**

```
SELECT table1.column1,table1.column2,table2.column1,....

FROM table1

RIGHT JOIN table2

ON table1.matching_column = table2.matching_column;



table1: First table.

table2: Second table

matching_column: Column common to both the tables.
```

Note: We can also use RIGHT OUTER JOIN instead of RIGHT JOIN, both are the same.

Example Queries(RIGHT JOIN):

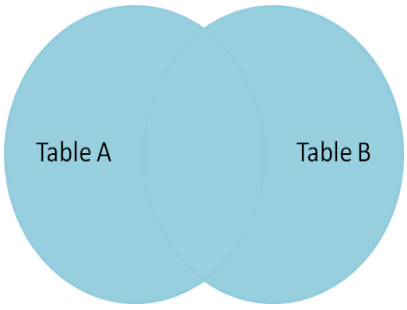
```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

RIGHT JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
NULL	4
NULL	5
NULL	4

Output:

27. **FULL JOIN:** FULL JOIN creates the result-set by combining results of both LEFT JOIN and RIGHT JOIN. The result-set will contain all the rows from both the tables. The rows for which there is no matching, the result-set will contain *NULL* values. **Syntax:**

```
28. SELECT table1.column1,table1.column2,table2.column1,....

29. FROM table1

30. FULL JOIN table2

31. ON table1.matching_column = table2.matching_column;
32. table1: First table.

33. table2: Second table

34. matching_column: Column common to both the tables.
```



Example Queries(FULL JOIN):

```
SELECT Student.NAME,StudentCourse.COURSE_ID

FROM Student

FULL JOIN StudentCourse

ON StudentCourse.ROLL_NO = Student.ROLL_NO;
```

Output:

NAME	COURSE_ID
HARSH	1
PRATIK	2
RIYANKA	2
DEEP	3
SAPTARHI	1
DHANRAJ	*NULL*
ROHIT	*NULL*
NIRAJ	*NULL*
NULL	9
NULL	10
NULL	11

Difference between INNER and OUTER JOIN.

What is Join? An SQL Join is used to combine data from two or more tables, based on a common field between them. For example, consider the following two tables.

Student Table

EnrollNo	StudentName	Address
1001	geek1	geeksquiz1
1002	geek2	geeksquiz2
1003	geek3	geeksquiz3
1004	geek4	geeksquiz4

StudentCourse Table

CourseID	EnrollNo
1	1001
2	1001
3	1001
1	1002
2	1003

Following is a join query that shows names of students enrolled in different courseIDs.

```
SELECT StudentCourse.CourseID,Student.StudentName

FROM Student

INNER JOIN StudentCourse

ON StudentCourse.EnrollNo = Student.EnrollNo

ORDER BY StudentCourse.CourseID
```

Note: INNER is optional above. Simple JOIN is also considered as INNER JOIN

The above query would produce the following result.

CourseID	StudentName
1	geek1
1	geek2
2	geek1
2	geek3
3	geek1

What is the difference between inner join and outer join?

Outer Join is of 3 types

1) Left outer join
2) Right outer join
3) Full Join

1) Left outer join returns all rows of table on left side of join. The rows for which there is no matching row on right side, result contains NULL in the right side.

```
SELECT Student.StudentName,
       StudentCourse.CourseID
FROM Student
LEFT OUTER JOIN StudentCourse
ON StudentCourse.EnrollNo = Student.EnrollNo
ORDER BY StudentCourse.CourseID
```

Note: OUTER is optional above. Simple LEFT JOIN is also considered as LEFT OUTER JOIN

StudentName	CourseID
geek4	*NULL*
geek2	1
geek1	1
geek1	2
geek3	2
geek1	3

2) Right Outer Join is similar to Left Outer Join (Right replaces Left everywhere)

3) Full Outer Join Contains results of both Left and Right outer joins.

Write a SQL query to retrieve furniture from database whose dimensions(Width, Height, Length) match with the given dimension.

```
SELECT *
FROM Furnitures
WHERE Furnitures.Length = GivenLength
  AND Furnitures.Breadth = GivenBreadth
  AND Furnitures.Height = GivenHeight
```

Write a SQL query to find the 4th maximum element from a table

Consider below simple table:

```
Name     Salary
---------------
abc     100000
bcd     1000000
efg     40000
ghi     500000
```

How to find the employee whose salary is second highest. For example, in the above table, "ghi" has the second-highest salary as 500000.

Finding highest element:
Below is a simple query to find the employee whose salary is highest.

```
SELECT name, MAX(salary) as salary
FROM employee
```

Finding Second highest element:

- We can nest the above query to find the second largest salary.

```
SELECT name, MAX(salary) AS salary
FROM employee
WHERE salary < (SELECT MAX(salary)
                   FROM employee);
```

- ```
  SELECT salary
  FROM employee
  ORDER BY salary desc limit 1,1
  ```

```
SELECT name, MAX(salary) AS salary
FROM employee
WHERE salary IN
      (SELECT salary
       FROM employee MINUS
                     SELECT MAX(salary)
                     FROM employee);
```

- ```
  SELECT name, MAX(salary) AS salary
  FROM employee
  WHERE salary (SELECT MAX(salary)
                FROM employee);
  ```

- IN SQL Server using Common Table Expression or **CTE**, we can find the second highest salary:

```
WITH T AS (SELECT *
           DENSE_RANK() OVER (ORDER BY Salary Desc) AS Rnk
           FROM Employees)
SELECT Name
FROM T
WHERE Rnk=2;
```

Finding Third highest element: Simple, we can do one more nesting.

```
SELECT name, MAX(salary) AS salary
  FROM employee
 WHERE salary < (SELECT MAX(salary)
                 FROM employee
                 WHERE salary < (SELECT MAX(salary)
                                 FROM employee)
                   );
```

```
SELECT salary
FROM employee
ORDER BY salary desc limit 2,1
```

Finding Fourth highest element: Simple, we can do one more nesting.

```
SELECT name, MAX(salary) AS salary
  FROM employee
 WHERE salary < (SELECT MAX(salary)
                 FROM employee
                 WHERE salary < (SELECT MAX(salary)
                                 FROM employee
                                 WHERE salary < (SELECT MAX(salary)
                                                 FROM employee)
                   );
```

```
SELECT salary
FROM employee
ORDER BY salary desc limit 3,1
```

Finding Nth highest element:

Note that instead of nesting for second, third, etc largest salary, we can find nth salary using general query like in MySQL:

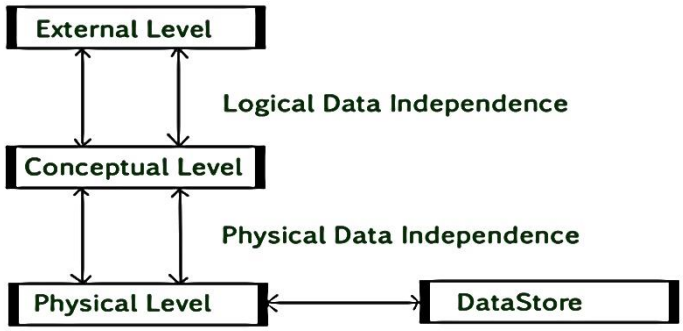
```
SELECT salary
FROM employee
ORDER BY salary desc limit n-1,1
```

```
SELECT name, salary
FROM employee A
```

```
WHERE n-1 = (SELECT count(1)
             FROM employee B
             WHERE B.salary>A.salary)
```

Explain 3-Tier Architecture in DBMS

DBMS 3-tier architecture divides the complete system into three inter-related but independent modules as shown below:



1. **Physical Level:** At the physical level, the information about the location of database objects in the data store is kept. Various users of DBMS are unaware of the locations of these objects.

2. **Conceptual Level:** At a conceptual level, data is represented in the form of various database tables. For Example, the STUDENT database may contain STUDENT and COURSE tables that will be visible to users but users are unaware of their storage.

3. **External Level:** An external level specifies a view of the data in terms of conceptual level tables. Each external level view is used to cater to the needs of a particular category of users. For Example, FACULTY of a university is interested in looking at the course details of students, STUDENTS are interested in looking at all details related to academics, accounts, courses, and hostel details as well. So, different views can be generated for different users.

Data Independence

Data independence means a change of data at one level should not affect another level. Two types of data independence are present in this architecture:

1. **Physical Data Independence:** Any change in the physical location of tables and indexes should not affect the conceptual level or external view of data. This data independence is easy to achieve and implemented by most of the DBMS.

2. **Conceptual Data Independence:** The data at conceptual level schema and external level schema must be independent. This means a change in conceptual schema should not affect external schema. e.g.; Adding or deleting attributes of a table should not affect the user's view of the table. But this type of independence is difficult to achieve as compared to physical data independence because the changes in conceptual schema are reflected in the user's view.

Explain Two tier architecture

The two-tier architecture is similar to a basic **client-server** model. The application at the client end directly communicates with the database at the server-side. API's like ODBC, JDBC are used for this interaction. The server side is responsible for providing query processing and transaction management functionalities. On the client side, the user interfaces and application programs are run. The application on the client-side establishes a connection with the server-side in order to communicate with the DBMS.
An advantage of this type is that maintenance and understanding are easier, compatible with existing systems. However, this model gives poor performance when there are a large number of users.

HIMANSHU KUMAR(LINKEDIN)
https://www.linkedin.com/in/himanshukumarmahuri

CREDITS- INTERNET.

DISCLOSURE- ALL THE DATA AND IMAGES ARE TAKEN FROM GOOGLE AND INTERNET.