**Questions related to DBMS**

**Q1. What is data and information?**

**Ans🡪**

**Data** is the raw material that can be processed for any computing machine. For example − Employee name, Product name, Name of the student, Marks of the student, Mobile number, Image etc.

**Information** is the data that has been converted into a more useful or intelligent form. For example: Report card sheet.

The information is needed for the following reasons −

* To gain knowledge about the surroundings.
* To keep the system up to date.
* To know about the rules and regulations of the society.

**Knowledge**

The human mind purposefully organizes information and evaluates it to produce knowledge.

A student secures 450 marks. Here 450 is **data**, the marks of the student are the **information** and the hard work required to get the marks is **knowledge**.

**Q2. What is a Database?**

**Ans.** A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS).

**Q3. What is DBMS?**

**Ans.** A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

It is used to perform different operations, like **addition**, **access**, **updating**, and **deletion** of the data.

**Q4. Why do we need DBMS when we already have file systems?**

**Ans.**

1. The absence of indexing in a traditional file-based system leaves us with the only option of scanning the full page and hence making the access of content tedious and super slow.
2. The other issue is redundancy and inconsistency as files have many duplicate and redundant data and changing one of them makes all of them inconsistent.
3. Accessing data is harder in traditional file-based systems because data is unorganized in them.
4. No security features to ensure guarded access of data.
5. Concurrent-access anomalies🡪 in case where there is a debit of money from an account from more than one place at a time.

**Q5. Explain abstraction in DBMS.**

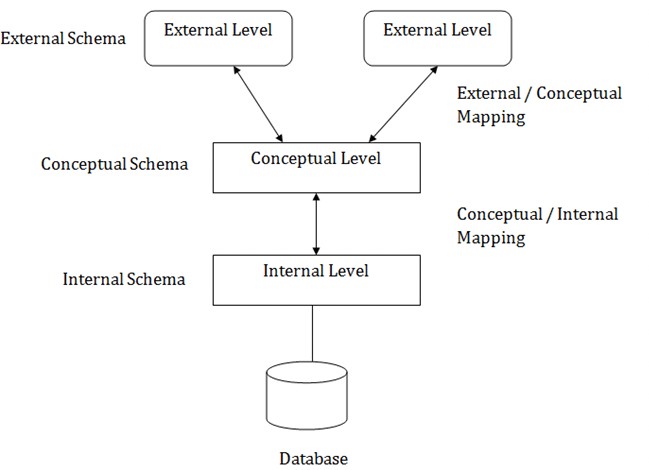
**Ans.** The major purpose of DBMS is to provide users with an abstract view of the data. That is, the system hides certain details of how the data is stored and maintained.

DBMS provides an abstract view to the user. For example, in an organization, teachers have data on marks and name while the account section has account-related data and don’t need to know other details.

**Q6. What are the different levels of DBMS architecture?**

**Ans.**

1. Physical level / Internal level
2. The lowest level of abstraction describes how the data are stored.
3. Low-level data structures used.
4. It has Physical schema which describes physical storage structure of DB.
5. Talks about: Storage allocation (N-ary tree etc), Data compression & encryption etc.
6. Goal: We must define algorithms that allow efficient access to data.
7. Logical level / Conceptual level:
8. The conceptual schema describes the design of a database at the conceptual level, describes what data are stored in DB, and what relationships exist among those data.
9. User at logical level does not need to be aware about physical-level structures.
10. DBA, who must decide what information to keep in the DB use the logical level of abstraction.
11. Goal: ease to use.
12. View level / External level:
13. The highest level of abstraction aims to simplify users’ interaction with the system by providing different view to different end-user.
14. Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group.
15. At the external level, a database contains several schemas that are sometimes called as subschema. The subschema is used to describe the different view of the database.
16. At views also provide a security mechanism to prevent users from accessing certain parts of DB.



**Q7. What is a DBA(Database Administrator) and its roles?**

**Ans.** A person who has central control of both the data and the programs that access those data.

Duties of DBA 🡪

1. Schema Definition
2. Storage structure and access methods.
3. Schema and physical organization modifications.
4. Authorization control.
5. Routine maintenance 1. Periodic backups. 2. Security patches. 3. Any upgrades.

**Q8. What are schemas and instances?**

**Ans.** The collection of information stored in the DB at a particular moment is called an **instance** of DB. The overall design of the DB is called the DB **schema.**

**Q9. What are data models?**

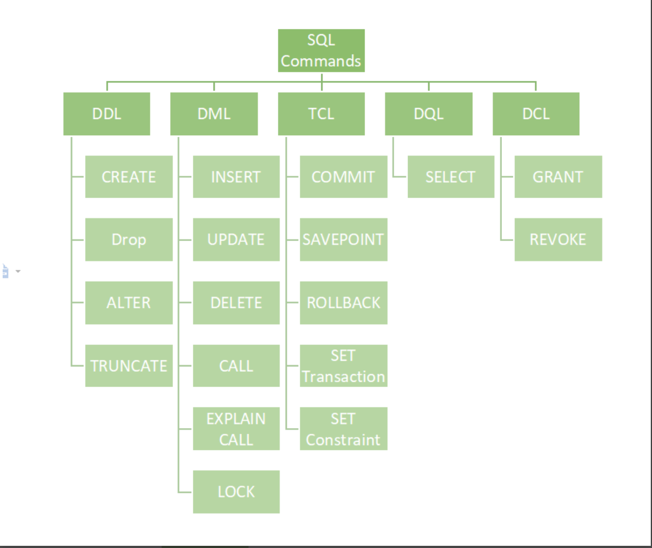
**Ans.**

1. Provides a way to describe the design of a DB at logical level.
2. Underlying the structure of the DB is the Data Model; a collection of conceptual tools for describing data, data relationships, data semantics & consistency constraints.
3. E.g., ER model, Relational Model, object-oriented model, object-relational data model etc.

**Q10. What are Database languages>**

**Ans.**

1. Data definition language (DDL)
2. To specify the database schema.
3. We specify consistency constraints, which must be checked, every time DB is updated.
4. CREATE table students (student\_ID int, name varchar(50), phone int);
5. Data manipulation language (DML)
6. To express database queries and updates.
7. Retrieval of information stored in DB.
8. Insertion of new information into DB.
9. Deletion of information from the DB.
10. Updating existing information stored in DB.
11. SELECT \* from students;

Practically, both language features are present in a single DB language, e.g., SQL language.

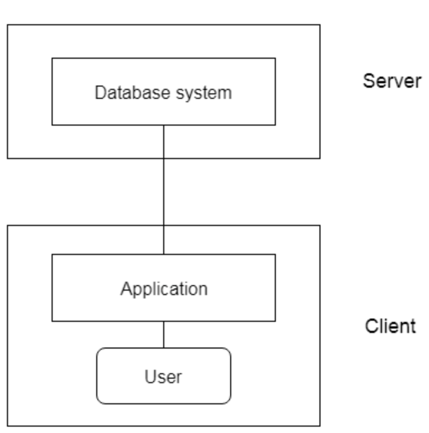
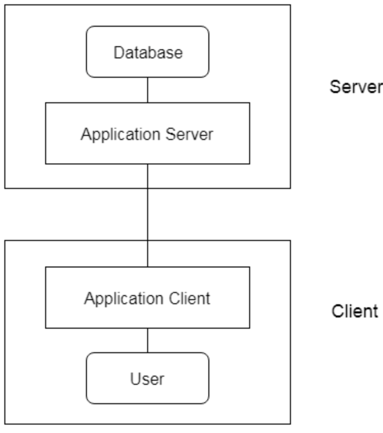
**Q11. How does the application made using C/C++ or Java interacts with the DBMS?**

**Ans.**

1. These languages are called host languages in this case.
2. API is provided to send DML/DDL statements to DB and retrieve the results.
3. Open Database Connectivity (ODBC), Microsoft “C”.
4. Java Database Connectivity (JDBC), Java.

**Q12. What are DBMS Application Architectures?**

**Ans.** These are Client machines, on which remote DB users work, and server machines on which the DB system runs.

1. Tier1 Architecture 🡪
2. The client, server & DB are all present on the same machine.
3. Tier2 Architecture 🡪
4. App is partitioned into 2-components.
5. Client machine, which invokes DB system functionality at server end through query language statements.
6. API standards like ODBC & JDBC are used to interact between client and server.
7. Tier3 Architecture 🡪
8. App is partitioned into 3 logical components.
9. Client machine is just a frontend and doesn’t contain any direct DB calls.
10. Client machine communicates with App server, and App server communicated with DB system to access data.
11. Business logic, what action to take at that condition is in App server itself. v. T3 architecture are best for WWW Applications.
12. Advantages 🡪 **Scalability** due to distributed application servers, **Data integrity**, App server acts as a middle layer between client and DB, which minimizes the chances of data corruption, Security, client can’t directly access DB, hence it is more secure.

**Q13. What are entities and entity set?**

**Ans.** An Entity is a “**thing**” or “**object**” in the real world that is **distinguishable** from all other objects. Eg. Each student in the college is an entity and student are an entity set of college database.

The entities in DBMS 🡪

1. has physical existence.
2. Can be uniquely identified (Primary Key)
3. Strong entity is which that can be uniquely identified.
4. Weak entity cannot be uniquely identified hence depends on some strong entity. It doesn’t have sufficient attributes.

**Q14. What are Attributes?**

**Ans**.

1. An entity is represented by a set of attributes.
2. There is a set of permitted values called domain of that attribute.
3. For a student attribute can be name, id , roll etc.

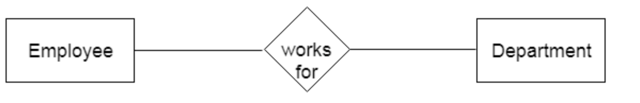
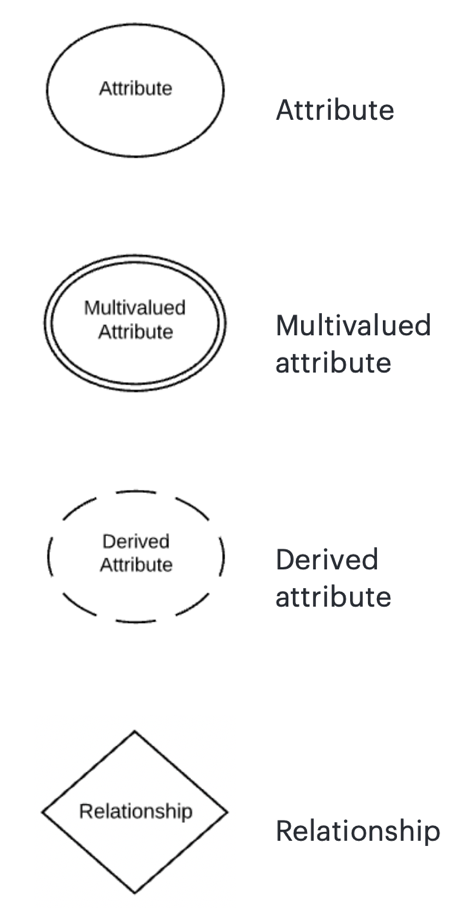
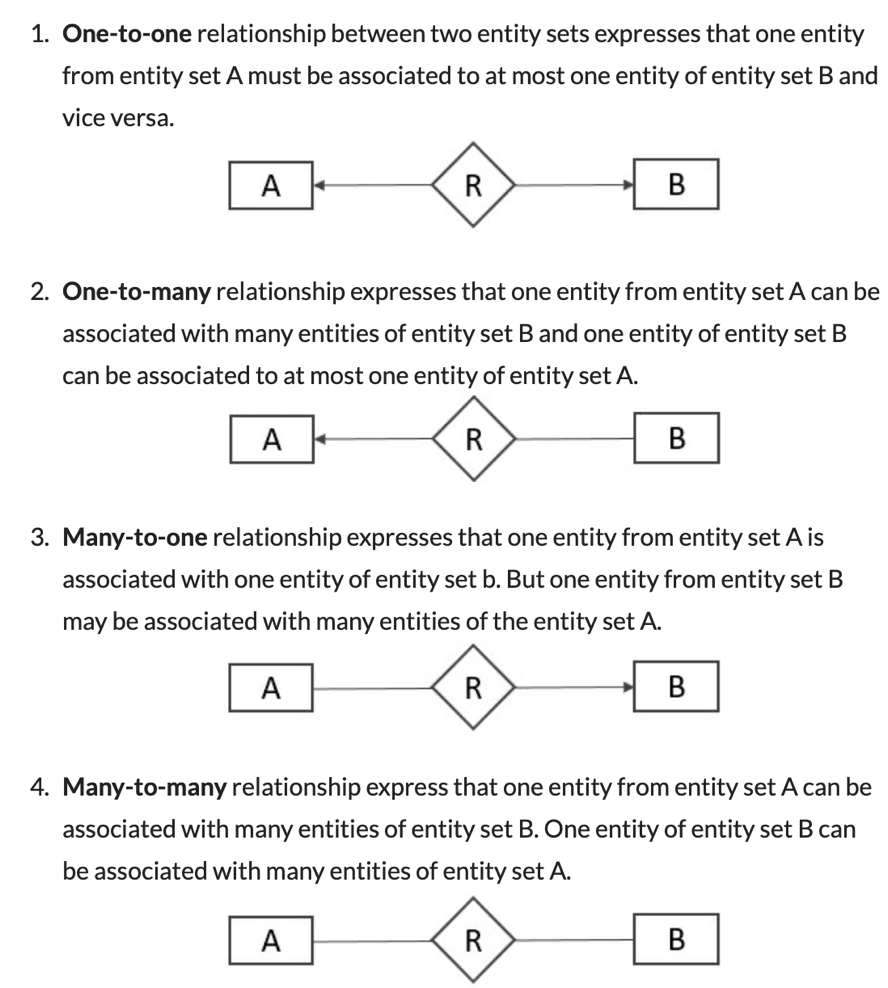
**Q15. What are relationships?**

**Ans**. It’s the association between two or more entities.  For example, a mother feeds her baby. Here "feeds" is a relationship, and this is the relationship between a child and the mother. These are categorised as

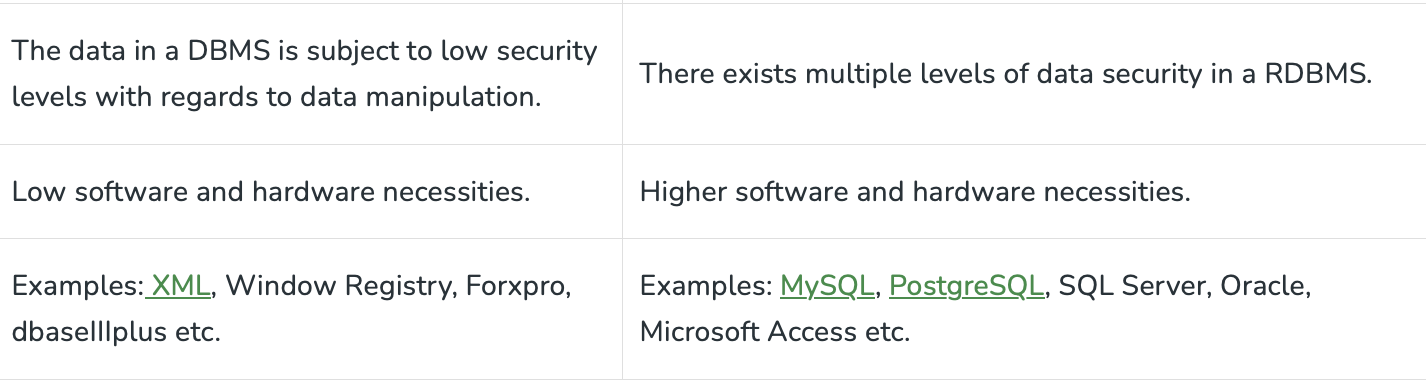
1. One to one 🡪 Every citizen has Aadhar card.
2. Many to one or one to many 🡪courses taken by prof or citizen has car
3. Many to many 🡪 customer buys a product.

Strong relationships are those established between two or more independent entities.

**Q16. What are different ER symbols?  
Ans.**

1. Entity 🡪 strong entity is represented by rectangles and weak entities are represented by double rectangles.
2. Attribute 🡪 represented by eclipse, primary attribute is represented by eclipse with underline, multi-valued attribute is represented by double eclipse, weak key attribute is represented by eclipse with dotted underlined text and derived attribute is represented by dotted eclipse.

**Q17. Difference between DBMS and RDBMS.**

**Ans.**

**Q18. What is specialization in ER models?  
Ans.** Specialisation is splitting up the entity set into further sub entity sets on the basis of their functionalities, specialities and features. It is a top to down approach.

Person entity set can be divided into customer, student, employee. Person is superclass and other specialised entity sets are subclasses.

Why Specialisation?

1. Certain attributes may only be applicable to a few entities of the parent entity set.
2. DB designer can show the distinctive features of the sub entities.
3. To group such entities we apply Specialisation, to overall refine the DB blueprint

\*\*Both Specialisation and Generalisation, has attribute inheritance and the attributes of higher level entity sets are inherited by lower level entity sets.

**Q19. What are Integrity Constraints?**

**Ans.** Integrity constraints are a set of rules. It is used to maintain the quality of information. integrity constraints ensure that the data insertion, updating, and other processes have to be performed in such a way that data integrity is not affected. Thus, integrity constraint is used to guard against accidental damage to the database.

1. Domain constraints 🡪 The data type of domain includes string, character, integer, time, date, currency, etc. The value of the attribute must be available in the corresponding domain.
2. Entity integrity constraints🡪 The entity integrity constraint states that primary key value can't be null.
3. Referential Integrity Constraints 🡪 A referential integrity constraint is specified between two tables. In the Referential integrity constraints, if a foreign key in Table 1 refers to the Primary Key of Table 2, then every value of the Foreign Key in Table 1 must be null or be available in Table 2.
4. Key constraints 🡪 Keys are the entity set that is used to identify an entity within its entity set uniquely. An entity set can have multiple keys, but out of which one key will be the primary key. A primary key can contain a unique and null value in the relational table.

**Q20. What is the problem if we have Redundancy in DBMS?  
Ans.**

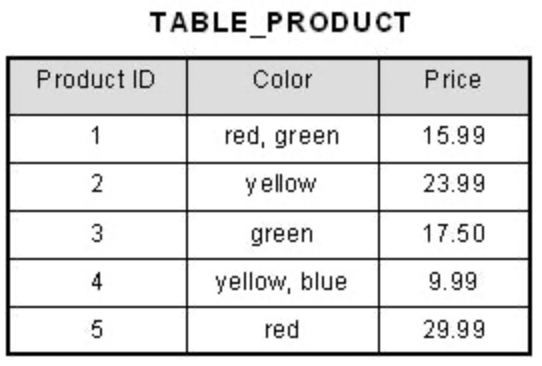
Unnecessary repetition. These anomalies increase size of DB and reduces its performance.

Introduces 3 types of anomalies 🡪

1. Insertion anomaly 🡪 This refers to the situation when it is impossible to insert certain types of data into the database. A new student is admitted who has not opted the branch but the table needs data like branch, HOD, Section etc.
2. Deletion anomaly 🡪 The deletion of data leads to unintended loss of additional data, data that we had wished to preserve. Only one student is there in ISE and we delete his entry the table will not have the branch ISE.
3. Updation anomaly This refers to the situation where updating the value of a column leads to database inconsistencies (i.e., different rows on the table have different values). If some HOD is changed we now need to update all the info everywhere.

**Q21. What are atomic values?**

**Ans.** An atomic value is a value that cannot be divided.

For example, in the table shown below, the values in the [Color] column in the first row can be divided into "red" and "green", hence [TABLE\_PRODUCT] is not in 1NF.

**Q23. What is functional dependency?**

**Ans.**

**Q24. Keys in DBMS?**

**Ans.**

It is a set of attributes(one or more like id, (id and salary) with the help of which identify all the other attributes uniquely.

1. Super keys/keys 🡪 the key that is capable of identifying all other attributes.
2. Candidate key 🡪 A candidate key is a set of one or more columns that can uniquely identify a row within a table .Its optimised form of super key. Super key whose proper subset is not a super key. (In simple manner the id is super key while (id and salary) is not a candidate key.) all candidate keys are super keys. There can be more than one candidate key.
3. Primary key 🡪 Maximum of one PK is possible. It’s a candidate key that is assigned as PK by DBA.

**Q25. What is Lossy Decomposition in DBMS?**

**Ans.**

Lossy decomposition is when a relation gets decomposed into multiple relational schemas, in such a way that retrieving the original relation leads to a loss of information. Thus, a lossy decomposition is bound to lose information.

When we decompose the table T into T1 and T2 there must be a common attribute so that they can be referenced else we cannot connect and there will be loss of information.

**Q26. What is normalization?**

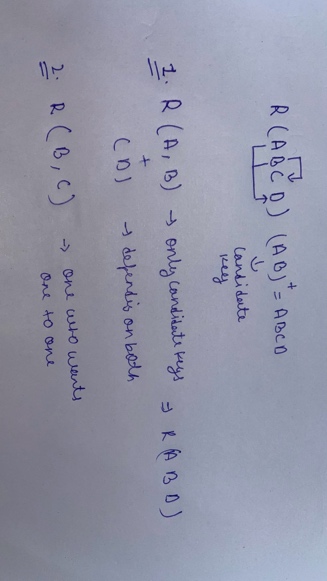
**Ans.**

To avoid redundancy in database we use normalization. Several problems emerge if there is redundant data .

We decompose the given table into multiple tables until SRP(Single Responsibility Principle) is achieved. Means that each table handles single id or each table represents a single idea or information.

1. 1NF

EK cell me ek value. Cannot have multi valued attributes.

1. Every relation must have atomic values.
2. b. relations must not have multivalued attributes.
3. 2NF

a. Relation must be in 1NF

b. there should not be any partial dependency.

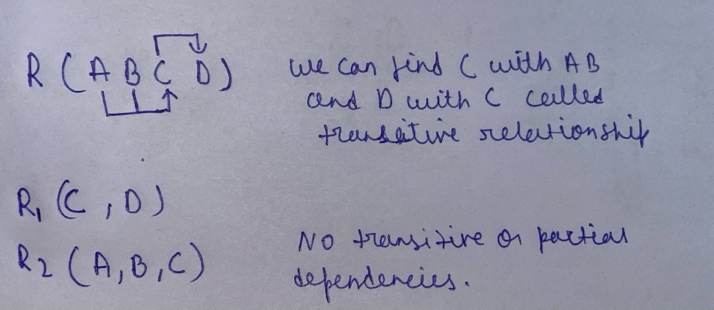
Partial dependencies 🡪 When a non-prime attribute instead of depending on entire candidate key depends on a part of (between B and C because it depends only on B instead of AB).

Prime attribute 🡪 If an attribute is a part of Candidate key( here A,B).

Non-Prime attribute 🡪 Attributes not a part of Candidate key.

What problem does this partial dependency cause?

See we here say that if we have values of Candidate key, we can find any of the remaining attributes by allowing any of them being null at a time but not both at the same time. But here we have no control on the partial dependencies as we rely only on B for the reference of C.

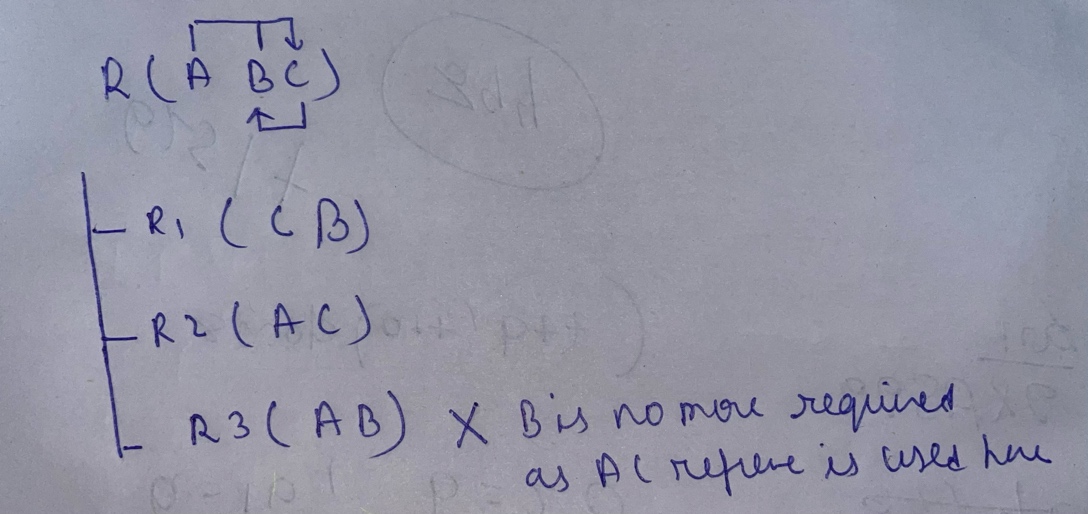
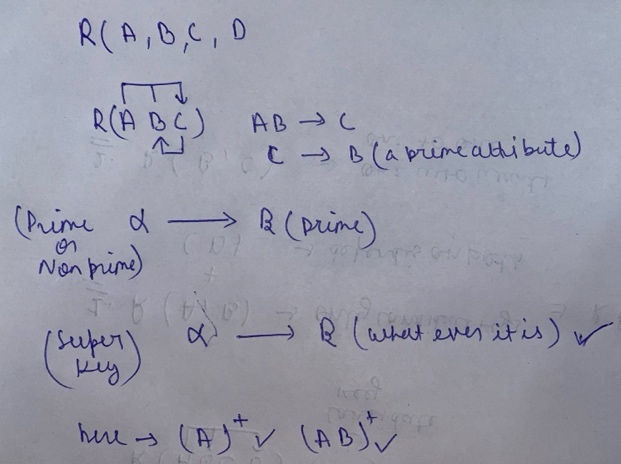
1. 3NF
2. Table is in the 2nd Normal form.
3. No transitive relationship exists

Transitive dependencies 🡪non-prime attribute depending on non-prime attribute is called transitive dependencies.

**Q. What problem does this transitive dependency cause?**

**Ans.**

As we promised earlier, we can find every other attribute if we have the non-null candidate keys fails as we depend on another non-prime attribute(C here) to find something (D here). C can be null for a non-null value of the candidate key and will fail to find the attribute dependent on.

1. BCNF(Boyce-Codd Normal Form) 🡪
2. Must be in 3rd normal form
3. Alpha must be Super key. Must not derive a prime attribute

**Q27. Identification of Normal Forms**

1. BCNF Normal Form

From 🡪 check if must be super key.

1. 3rd Normal Form

(Full candidate key) is candidate key or if is a prime attribute then it’s in 3rd normal

1. 2nd Normal Form

If there is partial dependency then it is not in 2nd else it is.

**Q28. What are transactions?**

**Ans.**

It is a logical unit of work that contains one or more SQL statements. The result of all these statements in a transaction either gets completed successfully or if there is failure it gets rollbacked.

**Q29. What are ACID properties?**

**Ans.**

To ensure integrity of the DB we require to maintain the acid properties of the transaction.

1. Atomicity 🡪
2. Either all operation of the transaction is reflected in the DB or none are .

For example, in case of transaction it happens successfully only if both debit and credit is done without any issues in case there is an issue the whole process needs to be rollbacked.

1. Consistency 🡪
2. Integrity constraints must be maintained before and after the transaction.
3. DB must be consistent after the transaction.

In case of money transfer the total amount in accounts must be same.

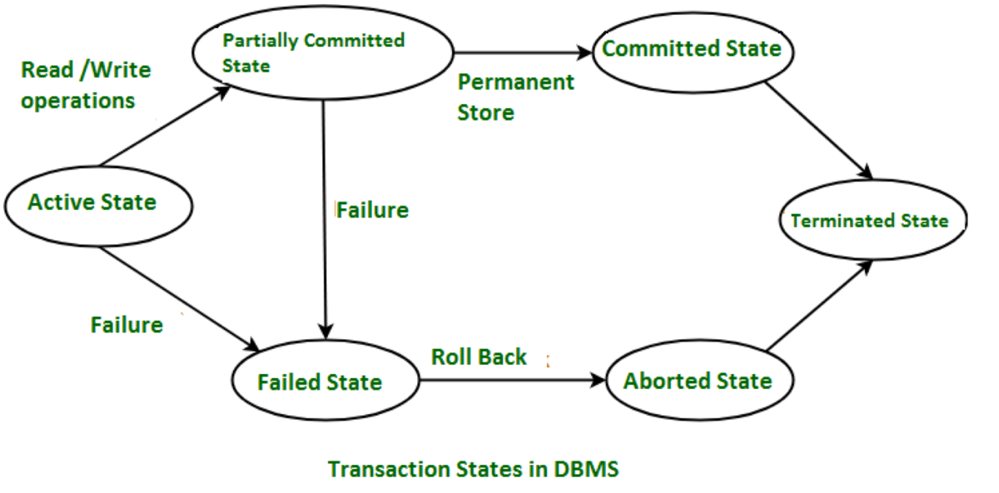
1. Isolation 🡪
2. Even through the multiple transactions occur at a same time but the system guarantees that logically execution is done in sequential way.
3. Each transaction is unaware of the other transaction being executed concurrently.
4. Or, Multiple transaction can happen in the system in isolation without interfering others.

For example, we are debiting money from multiple sources like GPay, netbanking and ATM simultaneously. There will be unstable transfers if the sending or receiving is intervened by some other sending receiving tasks.

1. Durability 🡪
2. After the transaction is completed the changes that are made in the database persist (is permanent) even if there are system failures.

For example, if a transaction is completed , firstly know that all operations are done and stored in main memory and finally committed(write) to the DB, so if the system crashes now we need to update the DB anyhow because if the transaction is successful, it must reflect in the DB. This is taken care by Recovery Management Component.

**Q30. What is Life cycle or States of Transaction?**

**Ans.**

1. Active state

When the instructions of the transaction are running then the transaction is in active state. If all the ‘read and write’ operations are performed without any error then it goes to the “partially committed state”; if any instruction fails, it goes to the “failed state”.

1. Partially Committed

After completion of all the read and write operation the changes are made in main memory or local buffer. If the changes are made permanent on the Database, then the state will change to “committed state” and in case of failure it will go to the “failed state”.

1. Failed State

When any instruction of the transaction fails, it goes to the “failed state” or if failure occurs in making a permanent change of data-on-Data Base.

1. Aborted State

After having any type of failure, the transaction goes from “failed state” to “aborted state” and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

1. Committed State

It is the state when the changes are made permanent on the Data Base and the transaction is complete and therefore terminated in the “terminated state”.

1. Terminated State – If there isn’t any roll-back or the transaction comes from the “committed state”, then the system is consistent and ready for new transaction and the old transaction is terminated.

**Q31. How to implement atomicity and durability?**

**Ans.**

Recovery Management Component of DBMS supports atomicity and durability.

1. **Shadow copy**
2. Based on making copies of DB (aka, shadow copies).
3. Assumption only one Transaction (T) is active at a time.
4. A pointer called db-pointer is maintained on the disk; which at any instant points to current copy of DB.
5. T, that wants to update DB first creates a complete copy of DB.
6. All further updates are done on new DB copy leaving the original copy (shadow copy) untouched.
7. If at any point the T has to be aborted the system deletes the new copy. And the old copy is not affected.
8. If T success, it is committed as,
9. OS makes sure all the pages of the new copy of DB written on the disk.
10. DB system updates the db-pointer to point to the new copy of DB.
11. New copy is now the current copy of DB.
12. The old copy is deleted.
13. The T is said to have been COMMITTED at the point where the updated db-pointer is written to disk.

**Atomicity**

1. If T fails at any time before db-pointer is updated, the old content of DB is not affected.
2. T abort can be done by just deleting the new copy of DB.
3. Hence, either all updates are reflected or none.

**Durability**

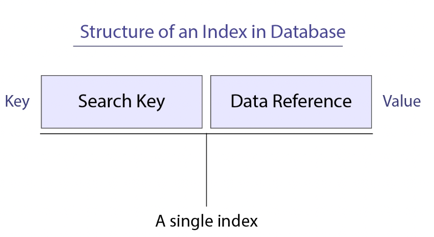
1. Suppose, system fails are any time before the updated db-pointer is written to disk.
2. When the system restarts, it will read db-pointer & will thus, see the original content of DB and none of the effects of T will be visible.
3. T is assumed to be successful only when db-pointer is updated.
4. If system fails after db-pointer has been updated. Before that all the pages of the new copy were written to disk. Hence, when system restarts, it will read new DB copy.

The implementation is dependent on write to the db-pointer being atomic. Luckily, disk system provides atomic updates to entire block or at least a disk sector. So, we make sure db-pointer lies entirely in a single sector. By storing db-pointer at the beginning of a block.

This is Inefficient, as entire DB is copied for every Transaction.

1. **Log-based recovery methods**
2. The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then
3. it can be recovered from there.
4. If any operation is performed on the database, then it will be recorded in the log.
5. But the process of storing the logs should be done before the actual transaction is applied in the database.
6. Stable storage is a classification of computer data storage technology that guarantees atomicity for any given write operation
7. and allows software to be written that is robust against some hardware and power failures.
8. **Deferred DB Modifications**
9. Ensuring atomicity by recording all the DB modifications in the log but deferring the execution of all the write operations until the final action of the T has been executed.
10. Log information is used to execute deferred writes when T is completed.
11. If system crashed before the T completes, or if T is aborted, the information in the logs is ignored.
12. If T completes, the records associated to it in the log file are used in executing the deferred writes.
13. If failure occur while this updating is taking place, we preform redo.
14. **Immediate DB Modifications**
    1. DB modifications to be output to the DB while the T is still in active state.
    2. DB modifications written by active T are called uncommitted modifications.
    3. In the event of crash or T failure, system uses old value field of the log records to restore modified values.
    4. Update takes place only after log records in a stable storage.
    5. Failure handling
15. System failure before T completes, or if T aborted, then old value field is used to undo the T.
16. If T completes and system crashes, then new value field is used to redo T having commit logs in the logs.

**Q32. What is indexing?**

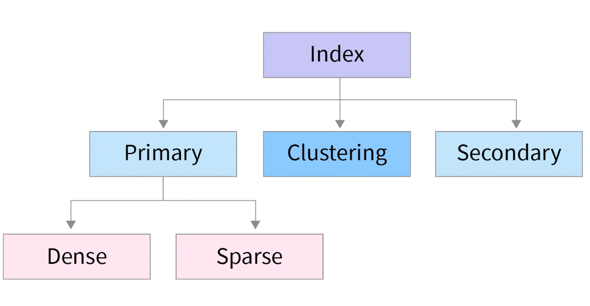
**Ans.**

Indexing is a database technique that enhances retrieval speed by creating a data structure based on sorted key values. This structure contains pointers to data locations, reducing disk visits for queries. Common types include B-tree, bitmap, and hash indexes, each suited for different scenarios. Indexing accelerates searches but requires storage and maintenance consideration. It's vital to select appropriate columns for indexing to optimize performance.

**Q33.Why do we need indexing?**

**Ans.**

* 1. Index file is always sorted.
  2. By ensuring that only unique values are inserted into columns that have been indexed as unique, indexing can also be utilized to ensure the integrity of data. This avoids storing duplicate data in the database, which might lead to issues when performing queries or reports.
  3. Indexing improves database performance by minimizing the number of disc visits required to fulfil a query with read operations like **SELECT** queries, **WHERE** clause etc.
  4. It is a data structure technique used to locate and quickly access data in databases.
  5. **Optimized Data Sorting**
  6. **Efficient Data Access**
  7. **We can apply different searching techniques on the sorted index like binary search.**

**Q34. What are the types of indexing?**

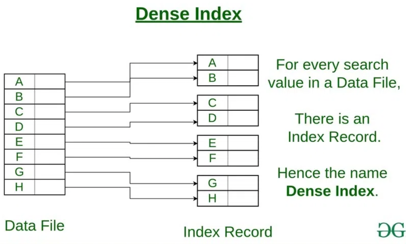
**Ans.**

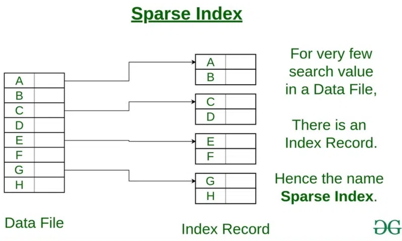
1. **Primary Index**

If the data file containing the records is sequentially ordered, a Primary index is an index whose search key also defines the sequential order of the file.

\*\*The term primary index is sometimes used to mean an index on a primary key. However, such usage is nonstandard and should be avoided.

All files are ordered sequentially on some search key. It could be Primary Key or non-primary key.

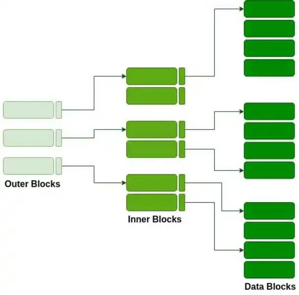
* + 1. **Dense Index🡪**

1. The index contains an index record for every search key values in the data file.
2. The index record contains the search-key value and a pointer to the first data record with that search-key value
3. The index record contains the search-key value and a pointer to the first data record with that search-key value
   * 1. **Sparse Index🡪**
4. Anindex record appears for only some of the search-key values.
5. Sparse Index helps you to resolve the issues of dense Indexing in DBMS. In this method of indexing technique, a range of index columns stores the same data block address, and when data needs to be retrieved, the block address will be fetched.

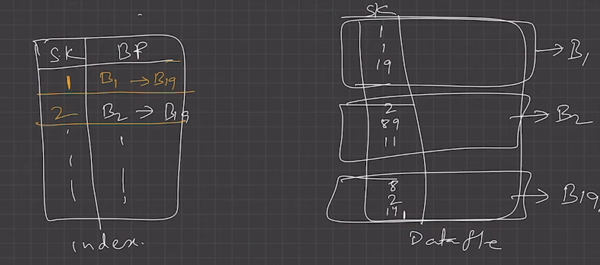
**#1 Based on key attributes**

1. Data file is sorted w.r.t primary key attribute.
2. PK will be used as search-key in Index.
3. Sparse Index will be formed i.e., no. of entries in the index file = no. of blocks in datafile.

**#2 Based on non-key attributes**

1. Data file is sorted w.r.t non-key attribute.
2. No. Of entries in the index = unique non-key attribute value in the data file.
3. This is dense index because all the unique values have an entry in the index file.
4. E.g., Let’s assume that a company recruited many employees in various departments. In this case, clustering indexing in DBMS should be created for all employees who belong to the same dept.

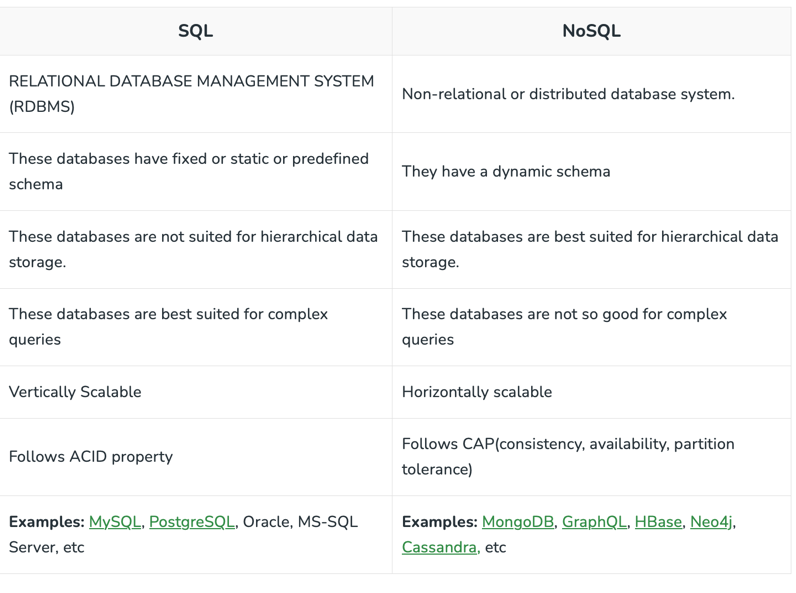
**#3 Multi-level Index**

1. Index with two or more levels.
2. If the single level index become enough large that the binary search itself would take much time, we can break down indexing into multiple levels.
3. **Secondary Indexing**
   * + 1. Data file is not sorted so we cannot use primary indexing.
       2. Can be done using key or non-key attributes.
       3. ****Called secondary indexing because normally one indexing is already applied.
       4. Number of entries in the index file = no. of records in the data file.
       5. It’s an example of dense indexing.
       6. In case there are duplicates in the given data we can use linked lists to store multiple random addresses. Hence in this case number of entries = number of unique elements.

**Q35. Is it possible to delete a primary key?**

**Ans.**

It’s possible to delete a primary key. When you delete a primary key, the related clustered index as well as the uniqueness property of that column gets lost.

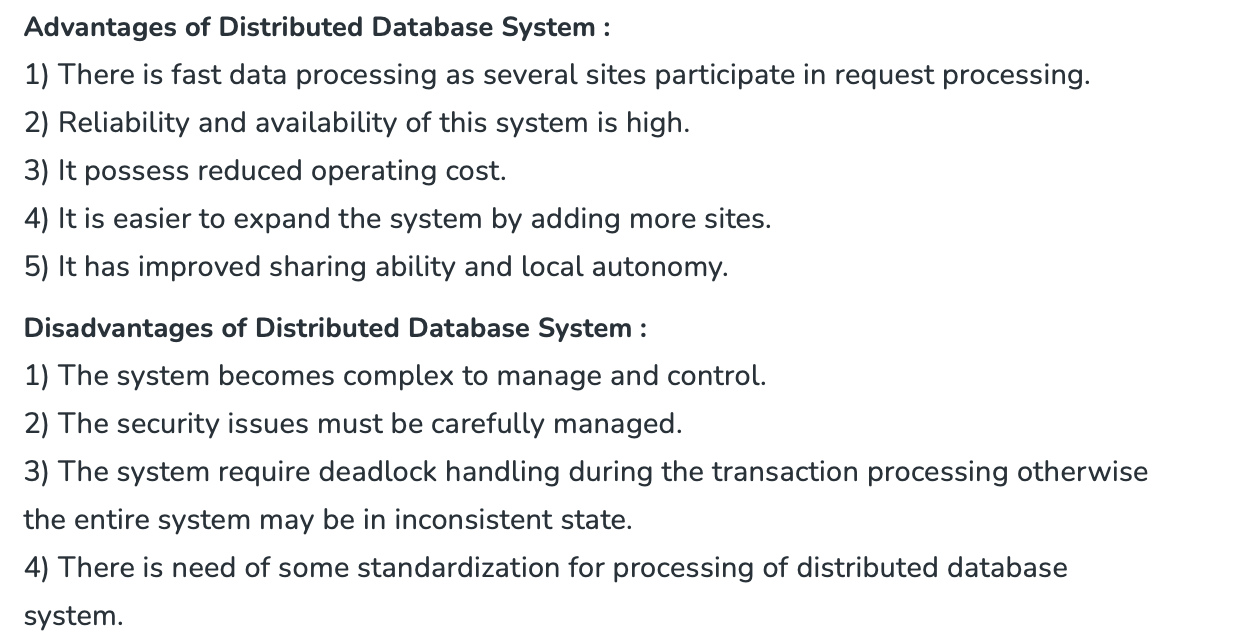
**Q35. What is NoSQL?**

Ans. It is called not only SQL. In different system designs we need to choose between SQL and NoSQL as per need.

* **Excellent for handling "big data" analytics**: NoSQL databases remove the bottleneck of needing to categorize and apply strict structures to massive amounts of information. NoSQL databases like HBase, [Cassandra](https://www.integrate.io/integrations/cassandra/), and CouchDB support the speed and efficiency of server operations while offering the capacity to work with large amounts of data.
* **No limits on types of data you can store**: NoSQL databases give you unlimited freedom to store diverse types of data in the same place. This offers the flexibility to add new and different types of data to your database at any time.
* **Easier to scale**: NoSQL databases are easier to scale. They're designed to be fragmented across multiple data centres without much difficulty.

**Q36. What is distributed database system?**

**Ans.**

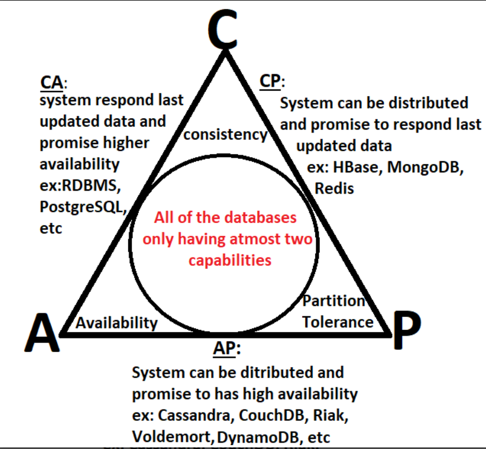
A distributed database is basically a database that is not limited to one system, it is spread over different sites, i.e, on multiple computers or over a network of computers. A distributed database system is located on various sites that don’t share physical components. This may be required when a particular database needs to be accessed by various users globally. It needs to be managed such that for the users it looks like one single database.

**Q37. What is horizontal and vertically scalable?**

**Ans.**

1. Vertical Scaling: When new resources are added to the existing system to meet the expectation, it is known as vertical scaling. Consider a rack of servers and resources that comprises the existing system. Now when the existing system fails to meet the expected needs, and the expected needs can be met by just adding resources, this is considered vertical scaling. Vertical scaling is based on the idea of adding more power(CPU, RAM) to existing systems, basically adding more resources. Vertical scaling is not only easy but also cheaper than Horizontal Scaling. It also requires less time to be fixed.
2. Horizontal Scaling: When new server racks are added to the existing system to meet the higher expectation, it is known as horizontal scaling.

**Q38. Explain CAP theorem.**

**Ans.**

The CAP theorem states that it is not possible to guarantee all three of the desirable properties – consistency, availability, and partition tolerance at the same time in a distributed system with data replication.

The theorem states that systems can only strongly support two of the properties.

1. Consistency

Consistency means that the nodes will have the same copies of a replicated data item visible for various transactions. A guarantee that every node in a distributed cluster returns the same, most recent and a successful write.

1. Availability – Availability means that each read or write request for a data item will either be processed successfully or will receive a message that the operation cannot be completed. Every non-failing node returns a response for all the read and write requests in a reasonable amount of time. The key word here is “every”. In simple terms, every node (on either side of a network partition) must be able to respond in a reasonable amount of time.
2. Partition Tolerance – Partition tolerance means that the system can continue operating even if the network connecting the nodes has a fault that results in two or more partitions, where the nodes in each partition can only communicate among each other. That means, the system continues to function and upholds its consistency guarantees in spite of network partitions. Network partitions are a fact of life. Distributed systems guaranteeing partition tolerance can gracefully recover from partitions once the partition heals.

**Q39. What is Sharding?**

**Ans.**

Sharding is a type of database partitioning in which a large database is divided or partitioned into smaller data and different nodes. These shards are not only smaller, but also faster and hence easily manageable.

Sharding allows the system to scale horizontally by adding more servers or nodes as the data grows. This improves the system’s capacity to handle large volumes of data and requests.

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

**Q40. What is SQL Injection?**

**Ans.**

SQL injection is a code injection technique that might destroy your database. SQL injection is one of the most common web hacking techniques. SQL injection is the placement of malicious code in SQL statements, via web page input.

Example SELECT \* FROM Users WHERE UserId = 105 OR 1=1;