Er diagram table statements:-

CREATE TABLE `Cart\_items` (

`cart\_item\_id ` Int,

`cart\_id ` FK(Int),

`SKU ` Timestamp,

`quantity` Timestamp

);

CREATE TABLE `Inventory\_listings` (

`SKU` FK(int),

`category` String,

`name` String,

`description` String,

`Q in stock` String

);

CREATE TABLE `Orders` (

`Cart\_id` PK(Int),

`user\_id` Int,

`Creation\_date` Date,

`Payment\_id` Int,

`order\_id` Int,

`Payment\_method` String,

`payment\_amount` Decimal,

`transaction\_status` String,

FOREIGN KEY (`Cart\_id`) REFERENCES `Cart\_items`(`cart\_id `)

);

CREATE TABLE `Payment` (

`sku` PK(Int),

`quantity` Int,

`order\_id` Int,

`user\_id` Int,

`order\_date` Date,

`total\_amount` Decimal,

`shipping\_address\_` String,

`billing\_address\_id` String,

`Last\_Updated` Tmestamp,

FOREIGN KEY (`sku`) REFERENCES `Inventory\_listings`(`SKU`),

FOREIGN KEY (`sku`) REFERENCES `Orders`(`order\_id`)

);

CREATE TABLE `Userbase` (

`user\_id` FK,Int,

`username` Type,

`firstname` Type,

`lastname` Type,

`email` Type,

`country` Type,

`address` Type,

`phone\_no` Type,

`registration\_dt` Type

);

CREATE TABLE `Admin` (

`User\_id` PK,int,

`Role` String,

`Lst\_Pswrd\_updt ` Timestamp,

`reset\_key` string,

`otp\_code` int,

`otp\_exp\_time` timestamp,

`Password` Type,

`email/MFA` string/int,

FOREIGN KEY (`Role`) REFERENCES `Payment`(`quantity`),

FOREIGN KEY (`User\_id`) REFERENCES `Userbase`(`user\_id`)

);

CREATE TABLE `Message` (

`notification\_id` INT,

`user\_id` FK(Int),

`message` String,

`notification\_date` Date,

`is\_read` String,

`Keywords` String

);

CREATE TABLE `Session` (

`Session\_id` Pk(Int),

`User\_id` Int,

`Session\_starttime` Timestamp,

`session\_endtime` Timestamp,

FOREIGN KEY (`User\_id`) REFERENCES `Admin`(`User\_id`)

);

**1.Explain about searching performance. How will you handle replication in SQL for searching &Reporting?**

**Data Replication** is the process of storing data in more than one site or node. It is useful in **improving the availability of data**. It is simply copying data from a database from one server to another server so that all the users can share the same data without any inconsistency. The result is a **distributed database** in which users can access data relevant to their tasks without interfering with the work of others. Data replication encompasses duplication of transactions on an ongoing basis, so that the **replicate is in a consistently updated state** and synchronized with the source. However in data replication data is available at different locations, but a particular relation has to reside at only one location. There can be full replication, in which the whole database is stored at every site. There can also be partial replication, in which some frequently used fragment of the database are replicated and others are not replicated.

Data Replication –

1. Transactional Replication – In Transactional replication users receive full initial copies of the database and then receive updates as data changes. Data is copied in real time from the publisher to the receiving database(subscriber) in the same order as they occur with the publisher therefore in this type of replication, transactional consistency is guaranteed. Transactional replication is typically used in server-to-server environments. It does not simply copy the data changes, but rather consistently and accurately replicates each change.
2. Snapshot Replication – Snapshot replication distributes data exactly as it appears at a specific moment in time does not monitor for updates to the data. The entire snapshot is generated and sent to Users. Snapshot replication is generally used when data changes are infrequent. It is bit slower than transactional because on each attempt it moves multiple records from one end to the other end. Snapshot replication is a good way to perform initial synchronization between the publisher and the subscriber.
3. Merge Replication – Data from two or more databases is combined into a single database. Merge replication is the most complex type of replication because it allows both publisher and subscriber to independently make changes to the database. Merge replication is typically used in server-to-client environments. It allows changes to be sent from one publisher to multiple subscribers.
4. **Explain what major factors are taken into consideration for performance.**

**Use Indexing**

Indexing is one of the core ways to give databases a performance boost. There are different ways of approaching indexing, but they all have the same goal: decreasing query wait time by making it easier to find and access data.

Indexes have a search key attached to a value or data reference. The index file will direct a query to a record, “bucket” of data, or group of data, depending on the indexing method used. Choosing a good indexing method for your specific needs will reduce strain on your system by making it much easier for data to be located, since a uniform, systematic organization is applied to the entire database.

## ****Avoid Using Loops****

Many coders learn early on that loops can be both useful and dangerous. It is all too easy to accidentally create an infinite loop and crash your whole system.

Loops are problematic when it comes to database performance because they often are looping redundantly. That is not to say that loops should never be used; they are useful sometimes. It simply depends on the specific case, and removing or minimizing unnecessary loops will help increase performance.

For example, having SQL queries inside of loops is not generally advised, because the system is running the same query numerous times rather than just once. A good rule of thumb is that the more data you have in a loop, the slower it is going to be.

## ****Get a Stronger CPU****

This fix is a classic in computer science. A CPU with better specs will increase system performance. There are ways, like those above, to increase performance within your system’s organization and coding. However, if you find that your database is consistently struggling to keep up, your hardware might be in need of an upgrade.

Even if the CPU you have seems like it should be sufficient, a CPU that is more powerful than your minimum requirements will be able to handle waves of queries with ease. The more data you are working with and the more queries you need to manage, the stronger your CPU needs to be.

## ****Defragment Data****

Data defragmentation is a common solution for performance issues. When data gets accessed, written, and rewritten many times, it can get fragmented from all that copying. It is good practice to go in and clean things up on a regular basis.

One symptom of fragmented data is clogged memory, where tables are taking up more room than they should. Crammed memory, as discussed below, is another common cause of a low-performing database.

## ****Optimize Queries****

There are many ways to go about optimizing queries, depending on the indexing method and the specific needs of your database. When queries aren’t being handled efficiently, the whole system can get backed up, leading to longer wait times for query results. Causes may include duplicate or overlapping indexes and keys or queries that return data that isn’t relevant.

Optimizing queries can be a complex process, but there are [some easy steps](https://www.dnsstuff.com/sql-server-query-optimization) you can take to work out the best plan for your database and identify its specific inefficiencies.

## ****Optimize Memory****

Another hardware fix that may help under-performing databases is additional memory space. Databases need some memory “wiggle room” to operate quickly. When your memory is nearly or completely full, things get backed up while the system struggles to find room for creating temporary files and moving things around. It’s a bit like trying to reorganize a living room that is packed floor-to-ceiling with boxes. You need plenty of empty floor space to maneuver and shuffle things around.

Databases work the same way. Increasing your database’s memory capacity will allow it more flexibility and operating room, reducing stress on the system so it can run more efficiently.

1. **Mention about Indexing, Normalization and Denormalization**.

# **Denormalization in Databases**

Denormalization is a technique used by database administrators to optimize the efficiency of their database infrastructure. This method allows us to add redundant data into a normalized database to alleviate issues with database queries that merge data from several tables into a single table. The denormalization concept is based on the definition of normalization that is defined as arranging a database into tables correctly for a particular purpose.

### **Pros of Denormalization**

The following are the advantages of denormalization:

**Enhance Query Performance**

Fetching queries in a normalized database generally requires joining a large number of tables, but we already know that the more joins, the slower the query. To overcome this, we can add redundancy to a database by copying values between parent and child tables, minimizing the number of joins needed for a query.

**Make database more convenient to manage**

A normalized database is not required calculated values for applications. Calculating these values on-the-fly will take a longer time, slowing down the execution of the query. Thus, in denormalization, fetching queries can be simpler because we need to look at fewer tables.

**Facilitate and accelerate reporting**

### **Cons of Denormalization**

The following are the disadvantages of denormalization:

* It takes large storage due to data redundancy.
* It makes it expensive to updates and inserts data in a table.
* It makes update and inserts code harder to write.
* Since data can be modified in several ways, it makes data inconsistent. Hence, we'll need to update every piece of duplicate data. It's also used to measure values and produce reports. We can do this by using triggers, transactions, and/or procedures for all operations that must be performed together.

# **Normalization**

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

* Making relations very large.
* It isn't easy to maintain and update data as it would involve searching many records in relation.
* Wastage and poor utilization of disk space and resources.
* The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

## Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

|  |  |
| --- | --- |
| **Normal Form** | **Description** |
| [1NF](https://www.javatpoint.com/dbms-first-normal-form) | A relation is in 1NF if it contains an atomic value. |
| [2NF](https://www.javatpoint.com/dbms-second-normal-form) | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| [3NF](https://www.javatpoint.com/dbms-third-normal-form) | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| BCNF | A stronger definition of 3NF is known as Boyce Codd's normal form. |
| [4NF](https://www.javatpoint.com/dbms-forth-normal-form) | A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency. |
| [5NF](https://www.javatpoint.com/dbms-fifth-normal-form) | A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless. |

**Advantages of Normalization**

* Normalization helps to minimize data redundancy.
* Greater overall database organization.
* Data consistency within the database.
* Much more flexible database design.
* Enforces the concept of relational integrity.

**Disadvantages of Normalization**

* You cannot start building the database before knowing what the user needs.
* The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
* It is very time-consuming and difficult to normalize relations of a higher degree.
* Careless decomposition may lead to a bad database design, leading to serious problems.

# **Indexing in DBMS**

* Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
* The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Ordered indices

The indices are usually sorted to make searching faster. The indices which are sorted are known as ordered indices.

**Example**: Suppose we have an employee table with thousands of record and each of which is 10 bytes long. If their IDs start with 1, 2, 3....and so on and we have to search student with ID-543.

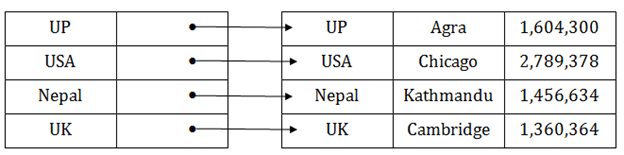
* In the case of a database with no index, we have to search the disk block from starting till it reaches 543. The DBMS will read the record after reading 543\*10=5430 bytes.
* In the case of an index, we will search using indexes and the DBMS will read the record after reading 542\*2= 1084 bytes which are very less compared to the previous case.

**Primary Index**

* If the index is created on the basis of the primary key of the table, then it is known as primary indexing. These primary keys are unique to each record and contain 1:1 relation between the records.
* As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
* The primary index can be classified into two types: Dense index and Sparse index.

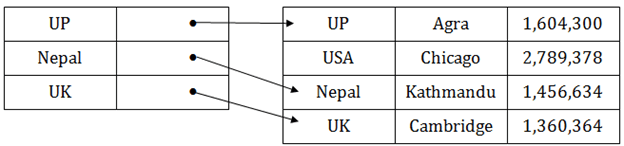
Dense index

* The dense index contains an index record for every search key value in the data file. It makes searching faster.
* In this, the number of records in the index table is same as the number of records in the main table.
* It needs more space to store index record itself. The index records have the search key and a pointer to the actual record on the disk.



**Sparse index**

* In the data file, index record appears only for a few items. Each item points to a block.
* In this, instead of pointing to each record in the main table, the index points to the records in the main table in a gap.



**Clustering Index**

* A clustered index can be defined as an ordered data file. Sometimes the index is created on non-primary key columns which may not be unique for each record.
* In this case, to identify the record faster, we will group two or more columns to get the unique value and create index out of them. This method is called a clustering index.
* The records which have similar characteristics are grouped, and indexes are created for these group.

## Secondary Index

In the sparse indexing, as the size of the table grows, the size of mapping also grows. These mappings are usually kept in the primary memory so that address fetch should be faster. Then the secondary memory searches the actual data based on the address got from mapping. If the mapping size grows then fetching the address itself becomes slower. In this case, the sparse index will not be efficient. To overcome this problem, secondary indexing is introduced.

In secondary indexing, to reduce the size of mapping, another level of indexing is introduced. In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges. The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).

**4.How will you handle scaling, if required at any point of time.**

When choosing how to scale your database, you must consider what's at stake when you scale up and out.Now we'll take a look at some factors to consider so you can choose which scaling system is best for your app:

**Load balancing**

The vertical scaling system is best for balancing loads because you have a single server (vertical scaling), and there is no need to balance your load. Horizontal scaling requires you to balance the workload evenly.

**Point of failure**

The horizontal scaling system has more than one server, so when one server crashes, the next one picks up the slack. This means that there is no single point of failure which makes the system resilient.But in the vertical scaling system, there is only one server, so once the server crashes, everything goes offline.

### **Speed**

In terms of speed, the vertical scaling system is faster because, since it runs on one server, the vertical scaling system has an interprocess communication – that is, the server communicates within itself and it's fast.The horizontal scaling system has network calls between two or more servers. This is also known as Remote Procedure Calls (RPC). RPCs are slow, though.

### **Data consistency**

When dealing with servers, you'll need to make sure that the data stored in them is consistent when end users send a request.The vertical scaling system is data consistent because all information is on a single server. But the horizontal scaling system is scaled out with multiple servers, so data consistency can be a huge issue.

### **Hardware limitations**

The horizontal scaling system scales well because the number of servers you throw at a request is linear to the number of users in the database or server. The vertical scaling system, on the other hand, has a limitation because everything runs on a single server.When choosing a system to scale your database, make sure to make a pros and cons list of the information in this article. It will help you decide which to use.