Q: What is the difference between the primary index and the secondary index?

In the context of databases, primary and secondary indexes are used to improve the efficiency of data retrieval operations, but they serve different purposes and have distinct characteristics:

**Primary Index (Primary Key):**

**Uniqueness:** A primary index is typically associated with the primary key of a table. The primary key uniquely identifies each row in the table, and the primary index enforces this uniqueness constraint. This means that no two rows in the table can have the same primary key value.

**Clustered Index:** In many database systems, the primary index is implemented as a clustered index. This means that the physical order of rows in the table is determined by the order of the primary key values. In other words, the data in the table is physically stored in the order of the primary key.

**One per Table:** Each table can have only one primary index, which is usually associated with the primary key.

**Optimized for Retrieval:** The primary index is optimized for efficient retrieval of specific rows based on the primary key values. It enables fast lookup operations when querying data by the primary key.

**Secondary Index (Secondary Key):**

**Non-Uniqueness:** A secondary index is used to improve the retrieval performance of data based on columns other than the primary key. Unlike the primary key, secondary index values may not be unique, meaning multiple rows can have the same indexed value.

**Non-Clustered Index:** A secondary index is typically implemented as a non-clustered index. In a non-clustered index, the physical order of rows in the table is independent of the index order. Instead, the index contains a pointer or reference to the actual data rows.

**Multiple per Table:** A table can have multiple secondary indexes, each defined on different columns or combinations of columns. These indexes can be used to optimize queries involving those columns.

**Optimized for Query Performance:** Secondary indexes are optimized for efficient query performance when filtering, sorting, or joining data based on the columns covered by the index. They allow for quicker access to data without scanning the entire table.

In summary, the primary index (associated with the primary key) enforces uniqueness and determines the physical order of rows in the table, while the secondary index (associated with secondary keys or other columns) improves query performance for non-primary key columns without enforcing uniqueness. Both primary and secondary indexes are essential for optimizing database queries and ensuring efficient data retrieval.

Q: What is the difference between the clustered Index and the non-clustered Index?

Clustered indexes and non-clustered indexes are two types of indexes used in relational database systems to improve the efficiency of data retrieval operations. They serve different purposes and have distinct characteristics:

**Clustered Index:**

**Physical Data Storage:** A clustered index determines the physical order in which data rows are stored in the table. In other words, the rows in the table are physically sorted and stored on disk based on the order of the clustered index key.

**Uniqueness:** There can be only one clustered index per table, and it is typically associated with the primary key of the table. This means that the values in the clustered index must be unique across all rows in the table.

**Performance:** Because the data rows are physically stored in the order of the clustered index key, queries that involve range scans or retrieval of rows based on the clustered index key are usually very efficient. However, insert, update, and delete operations that affect the order of the clustered index key can be less efficient, as they may require rearranging data on disk.

**Covering Index:** In a clustered index, the index itself contains all the data columns of the table. This makes it a covering index, meaning that queries that can be satisfied using only the clustered index do not require additional disk I/O to fetch data from the table.

**Non-Clustered Index:**

**Physical Data Storage:** A non-clustered index does not determine the physical order of data rows in the table. Instead, it contains a separate data structure that points to the actual data rows.

**Uniqueness:** Multiple non-clustered indexes can exist on a single table, and the indexed columns do not have to be unique. This means that non-clustered indexes can be created on both unique and non-unique columns.

**Performance:** Non-clustered indexes are optimized for improving the performance of queries that involve filtering, sorting, or joining data based on the indexed columns. They provide quick access to the location of data rows based on the indexed values.

**Covering Index:** Non-clustered indexes do not contain all the data columns of the table. Queries that need additional columns beyond what is available in the non-clustered index will require additional disk I/O to retrieve the necessary data from the table.

In summary, the primary difference between clustered and non-clustered indexes lies in how they physically store and organize data. Clustered indexes determine the physical order of data rows and are typically associated with the primary key, while non-clustered indexes provide quick access to rows based on the indexed columns but do not impact the physical order of data on disk. Both types of indexes play crucial roles in optimizing database query performance.

Q: What is the difference between the Clustered Index and the primary index?

The terms "clustered index" and "primary index" are sometimes used interchangeably, but they refer to slightly different concepts, especially in the context of relational databases:

**Primary Index:**

A primary index is an index that is specifically associated with the primary key of a table. The primary key is a column or set of columns that uniquely identifies each row in the table. The primary index enforces the uniqueness constraint of the primary key, ensuring that no two rows in the table can have the same primary key value.

**Key points about the primary index:**

1. It enforces uniqueness based on the primary key columns.
2. There can be only one primary index per table.
3. The primary index determines the physical order of data in the table if it is a clustered index.
4. If the primary key is not specified explicitly during table creation, some database systems automatically create a primary index based on a system-generated identifier.

**Clustered Index:**

A clustered index, on the other hand, is an index that determines the physical order of data rows in a table. In many database systems, the clustered index is associated with the primary key of the table by default. However, it doesn't have to be. You can choose to have a clustered index on a column other than the primary key if it better suits your query and performance requirements.

**Key points about the clustered index**:

1. It determines the physical order of data rows in the table.
2. There can be only one clustered index per table.
3. It's often associated with the primary key, but it doesn't have to be.
4. In summary, a primary index is associated with the primary key and enforces uniqueness based on the primary key columns, while a clustered index determines the physical order of data rows in a table. The key distinction is that the primary index is tied to the primary key, whereas the clustered index affects the physical storage of data, which can be the same as or different from the primary key.

Q: How The heap is stored on the disk?

In a database management system (DBMS), a "heap" typically refers to a heap file or heap table, which is a type of data storage structure used for organizing and storing records in a database. Unlike structured storage mechanisms such as B-trees or clustered indexes, a heap doesn't impose any specific order or organization on the data within it. Records are inserted into the heap as they arrive, and they are stored without any particular sorting or arrangement. Here's how a heap is typically stored on disk:

**Storage Pages:** Data in a heap is organized into storage pages, which are fixed-size blocks of data on the disk. Each page can hold a certain number of records.

**Record Storage:** Records are inserted into storage pages in the order they are received. When a new record is added to the heap and the current page is full, a new page is allocated for storage.

**Record Format:** Each record in the heap typically has a fixed-length format or a variable-length format with pointers to the actual data. The structure of records depends on the database system and the table schema.

**Page Headers:** Each storage page often contains a header that includes metadata about the page, such as a pointer to the next page in the heap, information about the number of records on the page, and possibly other management information.

**No Particular Order:** Unlike structured storage mechanisms like B-trees or clustered indexes, a heap doesn't enforce any specific order or sorting of data. Records are simply inserted as they come, and there's no guarantee that they will be retrieved in any particular sequence.

**Heap Table Scans:** When querying data from a heap, a full table scan is often required. This means that the DBMS must scan all storage pages of the heap to find the records that meet the query criteria, which can be less efficient compared to indexed structures for certain types of queries.

**Efficiency Trade-offs:** Heaps are efficient for inserting new records, as there's no need to maintain a specific order. However, they can be less efficient for retrieval operations, especially when dealing with large datasets and complex queries.

It's important to note that while heaps are relatively simple in terms of data organization, they may not be the best choice for all types of databases or workloads. For example, when you need to frequently retrieve specific records based on certain criteria or perform range queries, other storage structures like B-trees or clustered indexes may be more efficient. The choice of storage structure depends on the specific requirements of the database and the types of queries it needs to support.

Q: If we create a clustered index on a table, will it create another copy of the table and we will have 2 copies of the table one in the heap and the other one in the clustered index, or the clustered index will be created in the heap by resorting it ?

(clustered index vs heap table)

A: The clustered index will be created in the heap by resorting it

### Q: Data Representation in Memory VS. Disk (from Hussin Naser course Q&A section):

The data in memory is stored in pages, those pages have headers and user data and in fixed size. That page to the disk is just a bunch of bytes that get flushed in specific location in a file system. it is possible that the language the db is written in might have a higher level data structure with methods for convenience but access to the raw blob of bytes is essential for performance reasons.

Data structures are constructed after fetching the raw bytes, the DS itself will take its own memory in the heap. Another good example is the page pointers, we discussed in B+Tree section that the leaf pages are linked to each other, each page is linked to the previous and next page. This link technically is stored in the page but its simply a byte representing the page number, but when pulled in memory an actual pointer is created in memory to the correct next/prev page in memory so a dev can traverse pages as linked list.

One must realize that the cost of hydrating these data structure might not be trivial. Pretty sure postgres fixed many bugs and performance issues in this area (load DS as fast as possible or do lazy loading) I would for instance sure do a lazy loading of the pointer pages.

Q (from Hussin Naser course Q&A section): Most DBs have a primary key index which stores data according to its order in mostly contagious rows in blocks/pages. Say I have employee details table with Id(pk) and name.

Now if I have a secondary index on name, when I query using name, db first finds the primary key from secondary index and then loads that particular block of data fetching multiple rows and using the required ones, discard others

Now say I ran a range query in the secondary index, the name between Modi and Rahul,  
Db will find all primary keys for given name ranges and then try to fetch all random blocks of data. Here primary keys we got for name ranges may not be continuous sequence.

Does it mean secondary indexes are much slower than primary indexes? Due to random hops and more io because it has to load multiple blocks of data containing requested rows?

A: another fantastic question and observation and yes reads on secondary indexes will cause random reads on the PK.  
  
that is why postgres random reads are the worse because all indexes are secondary and going to the heap is always random.  
  
doing index only scans and include columns save you the cost .  
  
for instant in your examples say you are pulling the student id only where name is between rahul and hussein . that in itself is covering and doesn’t require to go to the pk as we have the id in the secondary index. but if you are pulling the student grade we need to jump to the pk and we take the hit of random reads.  
  
you can create the secondary index on name to include the grade such that a lookup on the secondary will get us both id and name! this of courses comes at acost of storage and more io in secondary scans..  
  
good q

Q: Where does the index store (clustered and non-clustered)?

Note: The database doesn’t read a single row, it reads a page or more in a single IO and we get a lot of rows in that IO.

Article by Microsoft:

[https://www.sqlshack.com/clustered-index-vs-heap](https://www.sqlshack.com/clustered-index-vs-heap/)