a) Basic Concepts

Introduction of Indexing

The main goal of designing the database is faster access to any data in the database and quicker insert/delete/update to any data. This is because no one likes waiting. When a database is very huge, even a smallest transaction will take time to perform the action. In order to reduce the time spent in transactions, Indexes are used.

Indexes are similar to book catalogues in library or even like an index in a book. What it does? It makes our search simpler and quicker. Same concept is applied here in DBMS to access the files from the memory.

When records are stored in the primary memory like RAM, accessing them is very easy and quick. But records are not limited in numbers to store in RAM. They are very huge and we have to store it in the secondary memories like hard disk. They are stored in the form of files in different data blocks. Each block is capable of storing one or more records depending on its size.

When we have to retrieve any required data or perform some transaction on those data, we have to pull them from memory, perform the transaction and save them back to the memory. In order to do all these activities, we need to have a link between the records and the data blocks so that we can know where these records are stored. This link between the records and the data block is called index. It acts like a bridge between the records and the data block.

How do we index in a book? We list main topics first and under that we group different sub-topic right? We do the same thing in the database too. Each table will have unique column or primary key column which uniquely determines each record in the table. Most of the time, we use this primary key to create index. Sometimes, we will have to fetch the records based on other columns in the table which are not primary key. In such cases we create index on those columns. But what is this index? Index in databases is the pointer to the block address in the memory. But these pointers are stored as (column, block_address) format

Introduction of Hashing

Hash File organization method is the one where data is stored at the data blocks whose address is generated by using hash function. The memory location where these records are stored is called as data block or data bucket. This data bucket is capable of storing one or more records.

The hash function can use any of the column value to generate the address. Most of the time, hash function uses primary key to generate the hash index – address of the data block. Hash function can be simple mathematical function to any complex mathematical function. We can even consider primary key itself as address of the data block. That means each row will be stored at the data block whose address will be same as primary key. This implies how simple a hash function can be in database.

In DBMS, hashing is a technique to directly search the location of desired data on the disk without using index structure. Hashing method is used to index and retrieve items in a database as it is faster to search that specific item using the shorter hashed key instead of using its original value. Data is stored in the form of data blocks whose address is generated by applying a hash function in the memory location where these records are stored known as a **data block or data bucket**.

Why Hashing

- For a huge database structure, it's tough to search all the index values through all its level and then you need to reach the destination data block to get the desired data.
- Hashing method is used to index and retrieve items in a database as it is faster to search that specific item using the shorter hashed key instead of using its original value.
- Hashing is an ideal method to calculate the direct location of a data record on the disk without using index structure.
- It is also a helpful technique for implementing dictionaries.

Important Terminologies using in Hashing

Here, are important terminologies which are used in Hashing:

- **Data bucket** – Data buckets are memory locations where the records are stored. It is also known as Unit Of Storage.
- **Key**: A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). This allows you to find the relationship between two tables.
- **Hash function**: A hash function, is a mapping function which maps all the set of search keys to the address where actual records are placed.
- **Linear Probing** – Linear probing is a fixed interval between probes. In this method, the next available data block is used to enter the new record, instead of overwriting on the older record.
- **Quadratic probing**- It helps you to determine the new bucket address. It helps you to add Interval between probes by adding the consecutive output of quadratic polynomial to starting value given by the original computation.
- **Hash index** – It is an address of the data block. A hash function could be a simple mathematical function to even a complex mathematical function.
- **Double Hashing** –Double hashing is a computer programming method used in hash tables to resolve the issues of has a collision.

There are mainly two types of SQL hashing methods:

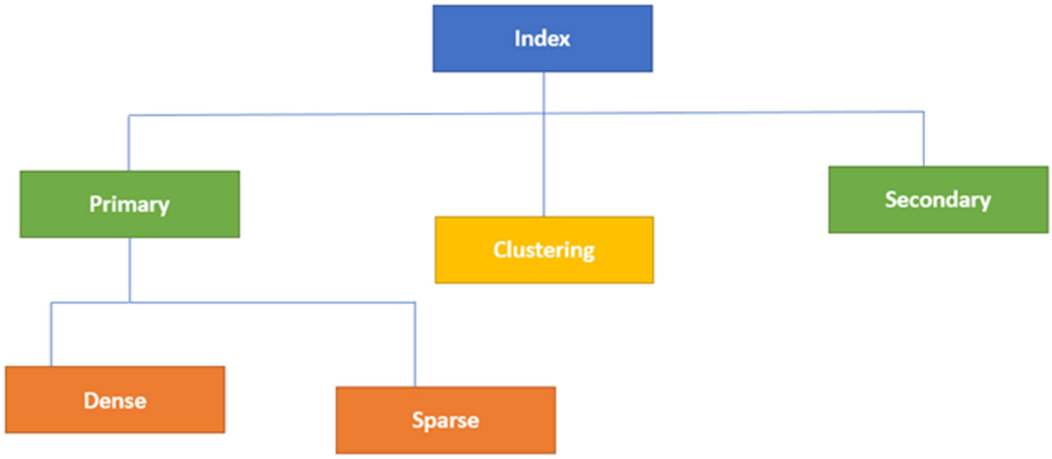
1. Static Hashing
2. Dynamic Hashing

b) Indexing in SQL

INDEXING is a data structure technique which allows you to quickly retrieve records from a database file. An Index is a small table having only two columns. The first column comprises a copy of the primary or candidate key of a table. Its second column contains a set of pointers for holding the address of the disk block where that specific key value stored.

An index -

- Takes a search key as input
- Efficiently returns a collection of matching records.



Type of Indexes

Database Indexing is defined based on its indexing attributes. Two main types of indexing methods are:

- Primary Indexing
- Secondary Indexing

Primary Indexing

Primary Index is an ordered file which is fixed length size with two fields. The first field is the same a primary key and second, filed is pointed to that specific data block. In the primary Index, there is always one to one relationship between the entries in the index table.

The primary Indexing is also further divided into two types.

- Dense Index
- Sparse Index

c) Ordered Indices – denses

In this case, indexing is created for primary key as well as on the columns on which we perform transactions. That means, user can fire query not only based on primary key column. He can query based on any columns in the table according to his requirement. But creating index only on primary key will not help in this case. Hence index on all the search key columns are stored. This method is called dense index.

For example, Student can be searched based on his ID which is a primary key. In addition, we search for student by his first name, last name, particular age group, residing in some place, opted for some course etc. That means most of the columns in the table can be used for searching the student based on different criteria. But if we have index on his ID, other searches will not be efficient. Hence index on other search columns are also stored to make the fetch faster.

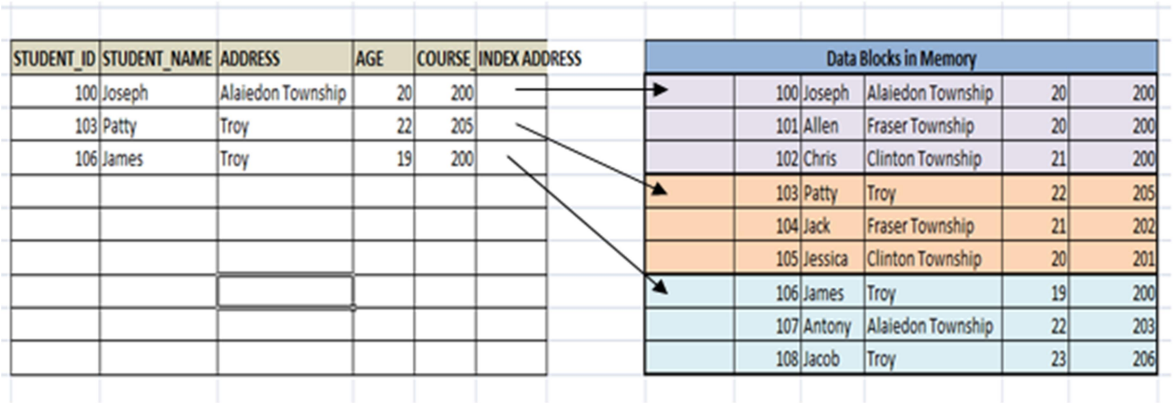
STUDENT_ID	STUDENT_NAME	ADDRESS	AGE	COURSE_	INDEX ADDRESS
100	Joseph	Alaiedon Township	20	200	
101	Allen	Fraser Township	20	200	
102	Chris	Clinton Township	21	200	
103	Patty	Troy	22	205	
104	Jack	Fraser Township	21	202	
105	Jessica	Clinton Township	20	201	
106	James	Troy	19	200	
107	Antony	Alaiedon Township	22	203	
108	Jacob	Troy	23	206	

Data Blocks in Memory				
100	Joseph	Alaiedon Township	20	200
101	Allen	Fraser Township	20	200
102	Chris	Clinton Township	21	200
103	Patty	Troy	22	205
104	Jack	Fraser Township	21	202
105	Jessica	Clinton Township	20	201
106	James	Troy	19	200
107	Antony	Alaiedon Township	22	203
108	Jacob	Troy	23	206

Though it addresses quick search on any search key, the space used for index and address becomes overhead in the memory. Here the (index, address) becomes almost same as (table records, address). Hence more space is consumed to store the indexes as the record size increases.

d) Sparse Index

In order to address the issues of dense indexing, sparse indexing is introduced. In this method of indexing, range of index columns store the same data block address. And when data is to be retrieved, the block address will be fetched linearly till we get the requested data



STUDENT_ID	STUDENT_NAME	ADDRESS	AGE	COURSE	INDEX ADDRESS
100	Joseph	Alaiedon Township	20	200	
103	Patty	Troy	22	205	
106	James	Troy	19	200	

Data Blocks in Memory				
100	Joseph	Alaiedon Township	20	200
101	Allen	Fraser Township	20	200
102	Chris	Clinton Township	21	200
103	Patty	Troy	22	205
104	Jack	Fraser Township	21	202
105	Jessica	Clinton Township	20	201
106	James	Troy	19	200
107	Antony	Alaiedon Township	22	203
108	Jacob	Troy	23	206

In above diagram we can see, we have not stored the indexes for all the records, instead only for 3 records indexes are stored. Now if we have to search a student with ID 102, then the address for the ID less than or equal to 102 is searched – which returns the address of ID 100. This address location is then fetched linearly till we get the records for 102. Hence it makes the searching faster and also reduces the storage space for indexes.

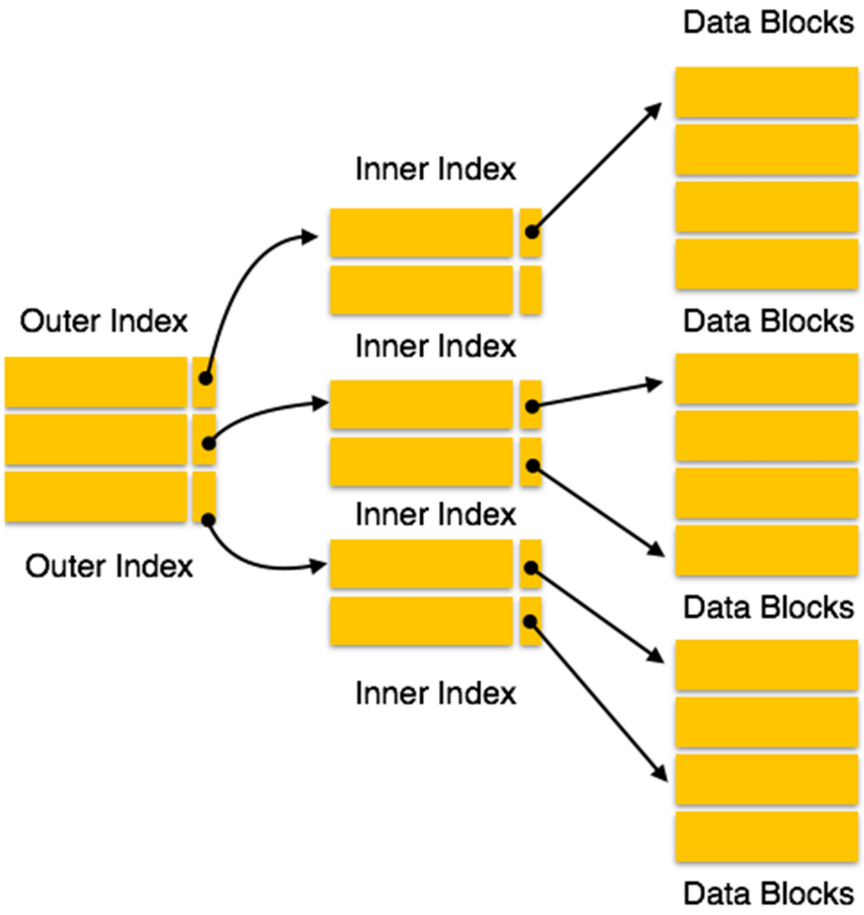
The range of column values to store the index addresses can be increased or decreased depending on the number of record in the table. The main goal of this method should be more efficient search with less memory space.

But if we have very huge table, then if we provide very large range between the columns will not work. We will have to divide the column ranges considerably shorter. In this situation, (index, address) mapping file size grows like we have seen in the dense indexing.

e) Multilevel Indices

Multilevel Indexing is created when a primary index does not fit in memory. In this type of indexing method, you can reduce the number of disk accesses to short any record and kept on a disk as a sequential file and create a sparse base on that file.

Index records comprise search-key values and data pointers. Multilevel index is stored on the disk along with the actual database files. As the size of the database grows, so does the size of the indices. There is an immense need to keep the index records in the main memory so as to speed up the search operations. If single-level index is used, then a large size index cannot be kept in memory which leads to multiple disk accesses.



Multi-level Index helps in breaking down the index into several smaller indices in order to make the outermost level so small that it can be saved in a single disk block, which can easily be accommodated anywhere in the main memory.

f) Static Hashing

In this method of hashing, the resultant data bucket address will be always same. That means, if we want to generate address for EMP_ID = 103 using mod (5) hash function, it always result in the same bucket address 3. There will not be any changes to the bucket address here. Hence number of data buckets in the memory for this static hashing remains constant throughout. In our example, we will have five data buckets in the memory used to store the data.

Searching a record

Using the hash function, data bucket address is generated for the hash key. The record is then retrieved from that location. i.e.; if we want to retrieve whole record for ID 104, and if the hash function is mod (5) on ID, the address generated would be 4. Then we will directly got to address 4 and retrieve the whole record for ID 104. Here ID acts as a hash key.

Inserting a record

When a new record needs to be inserted into the table, we will generate a address for the new record based on its hash key. Once the address is generated, the record is stored in that location.

Delete a record

Using the hash function we will first fetch the record which is supposed to be deleted. Then we will remove the records for that address in memory.

Update a record

Data record marked for update will be searched using static hash function and then record in that address is updated.

Suppose we have to insert some records into the file. But the data bucket address generated by the hash function is full or the data already exists in that address. How do we insert the data? This situation in the static hashing is called **bucket overflow**. This is one of the critical situations/ drawback in this method. Where will we save the data in this case? We cannot lose the data. There are various methods to overcome this situation. Most commonly used methods are listed below:

Closed hashing

In this method we introduce a new data bucket with same address and link it after the full data bucket. These methods of overcoming the bucket overflow are called closed hashing or **overflow chaining.**

Consider we have to insert a new record R2 into the tables. The static hash function generates the data bucket address as 'AACDBF'. But this bucket is full to store the new data. What is done in this case is a new data bucket is added at the end of 'AACDBF' data bucket and is linked to it. Then new record R2 is inserted into the new bucket. Thus it maintains the static hashing address. It can add any number of new data buckets, when it is full.

g) Dynamic Hashing

This hashing method is used to overcome the problems of static hashing – bucket overflow. In this method of hashing, data buckets grows or shrinks as the records increases or decreases. This method of hashing is also known as extendable hashing method. Let us see an example to understand this method.

Consider there are three records R1, R2 and R4 are in the table. These records generate addresses 100100, 010110 and 110110 respectively. This method of storing considers only part of this address – especially only first one bit to store the data. So it tries to load three of them at address 0 and 1.

What will happen to R3 here? There is no bucket space for R3. The bucket has to grow dynamically to accommodate R3. So it changes the address have 2 bits rather than 1 bit, and then it updates the existing data to have 2 bit address. Then it tries to accommodate R3.

Now we can see that address of R1 and R2 are changed to reflect the new address and R3 is also inserted. As the size of the data increases, it tries to insert in the existing buckets. If no buckets are available, the number of bits is increased to consider larger address, and hence increasing the buckets. If we delete any record and if the datas can be stored with lesser buckets, it shrinks the bucket size. It does the opposite of what we have seen above. This is how a dynamic hashing works. Initially only partial index/address generated by the hash function is considered to store the data. As the number of data increases and there is a need for more bucket, larger part of the index is consider to store the data.

Advantages of Dynamic hashing

Performance does not come down as the data grows in the system. It simply increases the memory size to accommodate the data.

Since it grows and shrinks with the data, memory is well utilized. There will not be any unused memory lying.

Good for dynamic databases where data grows and shrinks frequently.

h) Comparison of Indexing and hashing

Ordered Indexing	Hashing
Addresses in the memory are sorted for key value. This key value can be primary key or any other column in the table.	Addresses are generated using hash function on the key value. This key value can be primary key or any other column in the table.
Performance of this method comes down as the data increases in the file. Since it stores the data in a sorted form, when there is insert/delete/update operation, an extra effort to sort the record is needed. This reduces its performance.	Performance of dynamic hashing will be good when there is a frequent addition and deletion of data. But if the database is very huge, maintenance will be costlier. Static hashing will be good for smaller databases where record size id previously known. If there is a growth in data, it results in serious problems like bucket overflow.
There will be unused data blocks due to delete/update operation. These data blocks will not be released for re-use. Hence periodic maintenance of the memory is required. Else, memory is wasted and performance will also degrade. Also it will be cost overhead to maintain memory.	In both static and dynamic hashing, memory is well managed. Bucket overflow is also handled to better extent in static hashing. Data blocks are designed to shrink and grow in dynamic hashing. But there will be an overhead of maintaining the bucket address table in dynamic hashing when there is a huge database growth.
Preferred for range retrieval of data- that means when there is retrieval data for particular range, this method is best suited.	This method is suitable to retrieve a particular record based on the search key. But it will not perform better if the hash function is not on the search key.

i) Advantages and Disadvantages of Indexing

Advantages of Indexing

Important pros/ advantage of Indexing are:

- It helps you to reduce the total number of I/O operations needed to retrieve that data, so you don't need to access a row in the database from an index structure.
- Offers faster search and retrieval of data to users.
- Indexing also helps you to reduce table space as you don't need to link to a row in a table, as there is no need to store the ROWID in the Index. Thus you will able to reduce the table space.
- You can't sort data in the lead nodes as the value of the primary key classifies it.

Disadvantages of Indexing

Important drawbacks/cons of Indexing are:

- To perform the indexing database management system, you need a primary key on the table with a unique value.
- You can't perform any other indexes on the Indexed data.
- You are not allowed to partition an index-organized table.
- SQL Indexing Decrease performance in INSERT, DELETE, and UPDATE query.
- The biggest benefit of Indexing is that it helps you to reduce the total number of I/O operations needed to retrieve that data.
- The biggest drawback to performing the indexing database management system, you need a primary key on the table with a unique value.