

Trigger in SQL

A **Trigger** in Structured Query Language is a set of procedural statements which are executed automatically when there is any response to certain events on the particular table in the database. Triggers are used to protect the data integrity in the database.

In SQL, this concept is the same as the trigger in real life. For example, when we pull the gun trigger, the bullet is fired.

The trigger is always executed with the specific table in the database. If we remove the table, all the triggers associated with that table are also deleted automatically.

In Structured Query Language, triggers are called only either before or after the below events:

1. **INSERT Event:** This event is called when the new row is entered in the table.
2. **UPDATE Event:** This event is called when the existing record is changed or modified in the table.
3. **DELETE Event:** This event is called when the existing record is removed from the table.

Types of Triggers in SQL

Following are the six types of triggers in SQL:

1. **AFTER INSERT Trigger**
This trigger is invoked after the insertion of data in the table.
2. **AFTER UPDATE Trigger**
This trigger is invoked in SQL after the modification of the data in the table.
3. **AFTER DELETE Trigger**
This trigger is invoked after deleting the data from the table.
4. **BEFORE INSERT Trigger**
This trigger is invoked before the inserting the record in the table.
5. **BEFORE UPDATE Trigger**
This trigger is invoked before the updating the record in the table.
6. **BEFORE DELETE Trigger**
This trigger is invoked before deleting the record from the table.

Syntax of Trigger in SQL

1. **CREATE TRIGGER** Trigger_Name
2. [**BEFORE** | **AFTER**] [**Insert** | **Update** | **Delete**]
3. **ON** [Table_Name]
4. [**FOR EACH ROW** | **FOR EACH COLUMN**]
5. **AS**
6. **Set of SQL Statement**

Example of Trigger in SQL

To understand the concept of trigger in SQL, first, we have to create the table on which trigger is to be executed.

The following query creates the **Student_Trigger** table in the SQL database:

1. **CREATE TABLE** Student_Trigger

2. (
3. Student_RollNo **INT** NOT NULL **PRIMARY KEY**,
4. Student_FirstName **Varchar** (100),
5. Student_EnglishMarks **INT**,
6. Student_PhysicsMarks **INT**,
7. Student_ChemistryMarks **INT**,
8. Student_MathsMarks **INT**,
9. Student_TotalMarks **INT**,
10. Student_Percentage);

The following query fires a trigger before the insertion of the student record in the table:

1. **CREATE TRIGGER** Student_Table_Marks
2. **BEFORE INSERT**
3. **ON**
4. Student_Trigger
5. **FOR EACH ROW**
6. **SET** new.Student_TotalMarks = new.Student_EnglishMarks + new.Student_PhysicsMarks + new.Student_ChemistryMarks + new.Student_MathsMarks,
7. new.Student_Percentage = (new.Student_TotalMarks / 400) * 100;

Advantages of Triggers in SQL

Following are the three main advantages of triggers in Structured Query Language:

1. SQL provides an alternate way for maintaining the data and referential integrity in the tables.
2. Triggers helps in executing the scheduled tasks because they are called automatically.
3. They catch the errors in the database layer of various businesses.
4. They allow the database users to validate values before inserting and updating.

Disadvantages of Triggers in SQL

Following are the main disadvantages of triggers in Structured Query Language:

1. They are not compiled.
2. It is not possible to find and debug the errors in triggers.
3. If we use the complex code in the trigger, it makes the application run slower.
4. Trigger increases the high load on the database system.

Indexing

Data is stored in the form of records and every record has a key field, which helps it to be recognize uniquely. Indexing is a data structure technique to efficiently retrieve records from the database on some attributes on which the indexing has been done. indexing in database is similar what we see in books,

Indexing in DBMS

- o Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed.
- o The index is

a type of data structure. It is used to locate and access the data in a database table quickly.

- o It is defined based on the indexing attribute.

Index structure, Indexes can be created using some database columns.

Search key	Data Reference
------------	----------------

Fig: Structure of Index

- o The first column of the database is the search key that contains a copy of the primary key or candidate key of the table. The values of the primary key are stored in sorted order so that the corresponding data can be accessed easily.
- o The second column of the database is the data reference. It contains a set of pointers holding the address of the disk block where the value of the particular key can be found.

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.

- o 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 $543 \times 10 = 5430$ bytes.
- o In the case of an index, we will search using indexes and the DBMS will read the record after reading $542 \times 2 = 1084$ bytes which are very less compared to the previous case.


Indexing Methods:

Primary Index

- o 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.
- o As primary keys are stored in sorted order, the performance of the searching operation is quite efficient.
- o The primary index can be classified into two types: Dense index and Sparse index.

Dense index


- o The dense index contains an index record for every search key value in the data file. It makes searching faster.
- o In this, the number of records in the index table is same as the number of records in the main table.
- o 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.



UP	•	UP	Agra	1,604,300
USA	•	USA	Chicago	2,789,378
Nepal	•	Nepal	Kathmandu	1,456,634
UK	•	UK	Cambridge	1,360,364

Sparse index

- o In the data file, index record appears only for a few items. Each item points to a block.
- o 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.



UP	•	UP	Agra	1,604,300
Nepal	•	USA	Chicago	2,789,378
UK	•	Nepal	Kathmandu	1,456,634
		UK	Cambridge	1,360,364

Clustering Index

- o 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.
- o 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.
- o The records which have similar characteristics are grouped, and indexes are created for these group.

Secondary Index

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).

Hashing

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 do we need Hashing?

Here, are the situations in the DBMS where you need to apply the Hashing method:

- 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 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.
- **Bucket Overflow**: The condition of bucket-overflow is called collision. This is a fatal stage for any static has to function.

Types of Hashing Techniques

There are mainly two types of SQL hashing methods/techniques:

1. Static Hashing
2. Dynamic Hashing

B Tree

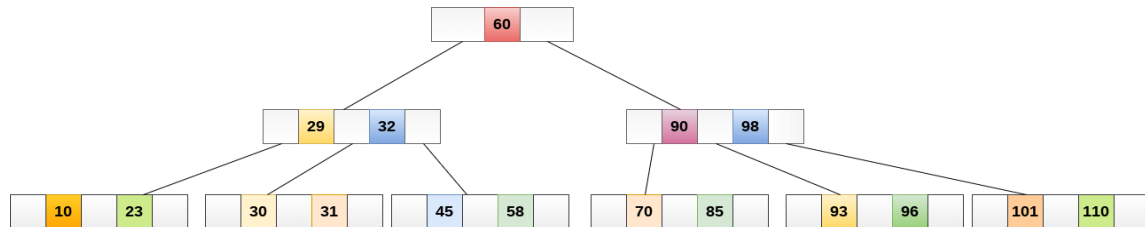
B Tree is a specialized m-way tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children. One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.

A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1. Every node in a B-Tree contains at most m children.
2. Every node in a B-Tree except the root node and the leaf node contain at least $m/2$ children.
3. The root nodes must have at least 2 nodes.
4. All leaf nodes must be at the same level.

It is not necessary that, all the nodes contain the same number of children but, each node must have $m/2$ number of nodes.

A B tree of order 4 is shown in the following image.



Operations

Searching
Inserting
Deletion

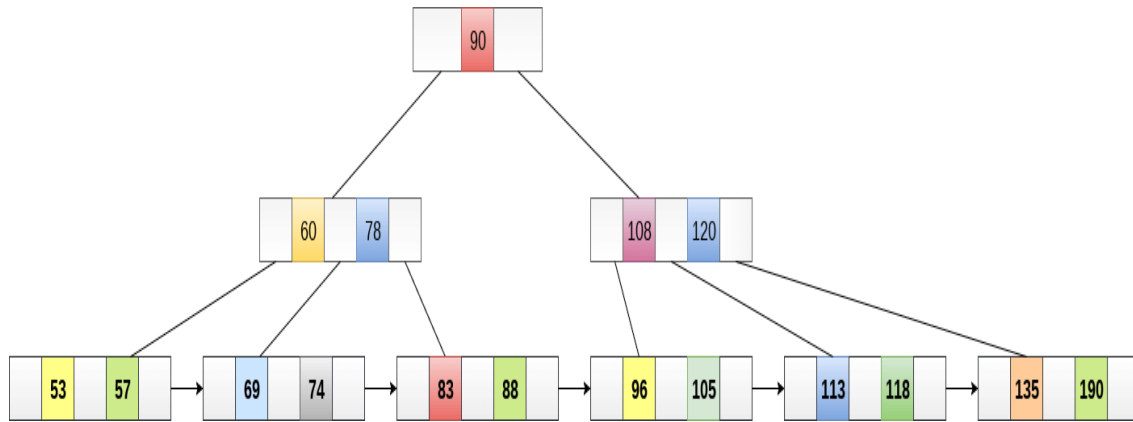
Application of B tree

B tree is used to index the data and provides fast access to the actual data stored on the disks since, the access to value stored in a large database that is stored on a disk is a very time consuming process.

Searching an un-indexed and unsorted database containing n key values needs $O(n)$ running time in worst case. However, if we use B Tree to index this database, it will be searched in $O(\log n)$ time in worst case.

B+ Tree

B+ Tree is an extension of B Tree which allows efficient insertion, deletion and search operations. In B Tree, Keys and records both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values. The leaf nodes of a B+ tree are linked together in the form of a singly linked lists to make the search queries more efficient. B+ Tree are used to store the large amount of data which can not be stored in the main memory. Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory. The internal nodes of B+ tree are often called index nodes. A B+ tree of order 3 is shown in the following figure.



Advantages of B+ Tree

1. Records can be fetched in equal number of disk accesses.
2. Height of the tree remains balanced and less as compare to B tree.
3. We can access the data stored in a B+ tree sequentially as well as directly.
4. Keys are used for indexing.
5. Faster search queries as the data is stored only on the leaf nodes.

B Tree VS B+ Tree

SN	B Tree	B+ Tree
1	Search keys can not be repeatedly stored.	Redundant search keys can be present.
2	Data can be stored in leaf nodes as well as internal nodes	Data can only be stored on the leaf nodes.
3	Searching for some data is a slower process since data can be found on internal nodes as well as on the leaf nodes.	Searching is comparatively faster as data can only be found on the leaf nodes.
4	Deletion of internal nodes are so complicated and time consuming.	Deletion will never be a complexed process since element will always be deleted from the leaf nodes.
5	Leaf nodes can not be linked together.	Leaf nodes are linked together to make the search operations more efficient.

Some Numericals:

Q1.. A data file consisting of 1,50,000 student-records is stored on a hard disk with block size of 4096 bytes. The data file is sorted on the primary key RollNo. The size of

a record pointer for this disk is 7 bytes. Each student-record has a candidate key attribute called ANum of size 12 bytes. Suppose an index file with records consisting of two fields, ANum value and the record pointer the corresponding student record, is built and stored on the same disk. Assume that the records of data file and index file are not split across disk blocks. The number of blocks in the index file is _____ .

- (A) 698
- (B) 898
- (C) 899
- (D) 4096

Answer: (A)

Explanation: Record size in Index

= 12+7

= 19 and Block size

= 4096

Number of Index records in 1 Block

= floor(4096/19)

= 215 records in 1 block.

Number of Blocks in the Index File

= Total Number of records/Records per block

= ceil(1,50,000/215)

= 698

Q2.. Consider a Hard Disk (HD) with Block Size 1000 bytes, each record size is 250 bytes, total records are 10000 and the data entered in HD without any order (unordered). Find average time complexity (I/O cost) to search a record from HD

Solution:

No of records in each block = $1000/250 = 4$

Total no of block = $10000/4 = 2500$

I/O cost = 1 (best case), 2500 (worst case)

Average cost = $2500/2 = 1250$

We can say, it has linear time complexity $O(n)$

If data is sorted, time complexity is $O(\log(n))$

Q3.. Consider a table stored on the hard disk in which block size=2000 Bytes, each record has size =50 Bytes. If total number of records are 8,000,000.

1. What is average time complexity to search an ordered and unordered record from the database table without indexing.

the average search time complexity = $O(n)$, n = number of records

=> Time = 8,000,000 unit time

the average search time complexity = $O(\log n)$, n = number of records

=> Time = $\log_2 8,000,000 = 22.9$ unit time

2. What is average time complexity to search an ordered and unordered record from the database table with indexing, when index table entry size is 20 Bytes.

records in 1 block = $2000/50 = 40$

blocks = Total number of records / Number of records in 1 block = $8,000,000 / 40 = 200,000$

index table entry size is 20 bytes but number of blocks = 200,000 => 1 index points to 10,000 blocks

1 index points to 400,000 records

the average search time complexity = $O(n)$, n = number of records in that index

=> Time = 400,000 unit time

the average search time complexity = $O(\log n)$, n = number of records in that index

=> Time = $\log_2 400,000 = 18.6$ unit time