SQL

LECTURE 7

What is SQL?

Structured Query Language (SQL)

It is a language designed to perform complex queries on databases.

- SQL is case insensitive. But it is a recommended practice to use keywords (like SELECT, UPDATE, CREATE, etc) in capital letters and use user defined things (liked table name, column name, etc) in small letters.
- SQL is the programming language for relational databases (explained below) like MySQL, Oracle, Sybase, SQL Server, Postgre, etc. Other non-relational databases (also called NoSQL) databases like MongoDB, DynamoDB, etc do not use SQL
- Although there is an ISO standard for SQL, most of the implementations slightly vary in syntax. So we may encounter queries that work in SQL Server but do not work in MySQL.

What is SQL?

With SQL, operations can only be performed on the database.

with SQL;

records can be added to the database,

records can be changed.

can be deleted and

Lists can be created from these records.

Databases Using SQL Language

MySQL

Mssql

PostgreSQL

Microsoft SQL Server

Oracle

Firebird

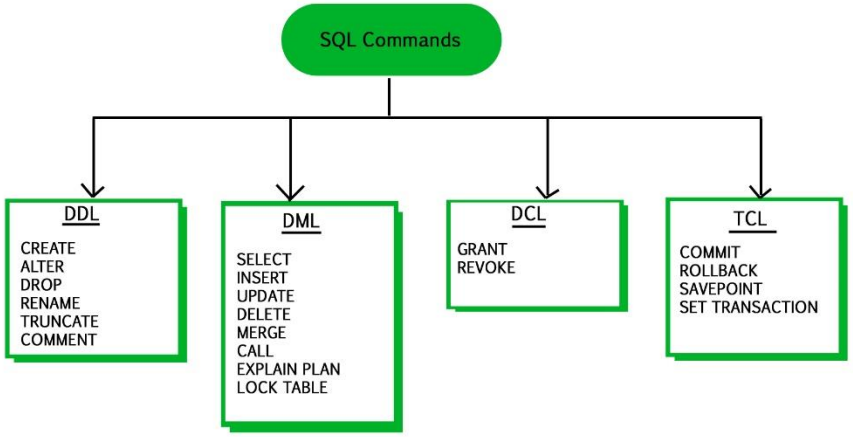
SQL Commands

Structured Query Language (SQL-Structured Query Language)

Data Manipulation Language (DML)

Data Definition Language (DDL)

Data Control Language (DCL-Data Control Language)



SQL Commands

Data Manipulation Language (DML)

1. It performs query, insertion, update and deletion operations on the data in a table.
   - select
   - insert
   - Update
   - delete

SQL Komutları

2. Data Definition Language (DDL)

It performs table creation, modification, table creation, index creation and deletion from scratch.

- Create Table
- Drop Table
- Alter Table
- Create Index
- Drop Index
- Alter View

SQL Komutları

3. Data Control Language (DCL-Data Control Language)

It contains SQL commands that allow users to perform operations such as granting or withdrawing some rights on the database.

- Create User
- Drop User
- Alter User
- Grant
- revoke

Select command

Data Query Language: It is used to extract the data from the relations. e.g.; SELECT

So first we will consider the Data Query Language. A generic query to retrieve from a relational database is:

SELECT [**DISTINCT**] Attribute_List **FROM** R1,R2….RM

[**WHERE** condition]

[**GROUP BY** (Attributes)[**HAVING** condition]]

[**ORDER BY**(Attributes)[**DESC**]];

Select command

SELECT [**DISTINCT**] Attribute_List **FROM** R1,R2….RM

[**WHERE** condition]

[**GROUP BY** (Attributes)[**HAVING** condition]]

[**ORDER BY**(Attributes)[**DESC**]];

Part of the query represented by statement 1 is compulsory if you want to retrieve from a relational database. The statements written inside [] are optional. We will look at the possible query combination on relation shown in Table 1.

Select command

Case 1: If we want to retrieve attributes **ROLL_NO** and **NAME** of all students, the query will be:

```
SELECT ROLL_NO, NAME FROM STUDENT;
```

ROLL_NO	NAME
1	RAM
2	RAMESH
3	SUJIT
4	SURESH

Select command

Case 2: If we want to retrieve **ROLL_NO** and **NAME** of the students whose **ROLL_NO** is greater than 2, the query will be:

```
SELECT ROLL_NO, NAME FROM STUDENT
WHERE ROLL_NO > 2;
```

ROLL_NO	NAME
3	SUJIT
4	SURESH

Select command

CASE 3: If we want to retrieve all attributes of students, we can write * in place of writing all attributes as:

```
SELECT * FROM STUDENT
WHERE ROLL_NO>2;
```

ROLL_NO	NAME	ADDRESS	PHONE	AGE
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Select command

CASE 4: If we want to represent the relation in ascending order by **AGE**, we can use ORDER BY clause as:

```
SELECT * FROM STUDENT ORDER BY AGE;
```

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
4	SURESH	DELHI	9156768971	18
3	SUJIT	ROHTAK	9156253131	20

Select command

Note: ORDER BY **AGE** is equivalent to ORDER BY **AGE** ASC. If we want to retrieve the results in descending order of **AGE**, we can use ORDER BY **AGE** DESC.

ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
4	SURESH	DELHI	9156768971	18
3	SUJIT	ROHTAK	9156253131	20

Select command

CASE 5: If we want to retrieve distinct values of an attribute or group of attribute, DISTINCT is used as in:

```
SELECT DISTINCT ADDRESS FROM STUDENT;
```

ADDRESS
DELHI
GURGAON
ROHTAK

If DISTINCT is not used, DELHI will be repeated twice in result set. Before understanding GROUP BY and HAVING, we need to understand aggregations functions in SQL.

AGGRATION FUNCTIONS

Aggregation functions are used to perform mathematical operations on data values of a relation. Some of the common aggregation functions used in SQL are:

COUNT: Count function is used to count the number of rows in a relation. e.g;

```
SELECT COUNT (PHONE) FROM STUDENT;
```

COUNT(PHONE)
4

AGGRATION FUNCTIONS

SUM: SUM function is used to add the values of an attribute in a relation. e.g;

SELECT SUM (AGE) **FROM** STUDENT;

SUM(AGE)
74

AGGRATION FUNCTIONS

In the same way, MIN, MAX and AVG can be used. As we have seen above, all aggregation functions return only 1 row.

AVERAGE: It gives the average values of the tupples. It is also defined as sum divided by count values.
Syntax:AVG(attributename)
OR
Syntax:SUM(attributename)/COUNT(attributename)
The above mentioned syntax also retrieves the average value of tupples.

MAXIMUM: It extracts the maximum value among the set of tupples.
Syntax:MAX(attributename)

MINIMUM: It extracts the minimum value amongst the set of all the tupples.
Syntax:MIN(attributename)

AGGRATION FUNCTIONS

GROUP BY: Group by is used to group the tuples of a relation based on an attribute or group of attribute. It is always combined with aggregation function which is computed on group. e.g.;

```
SELECT ADDRESS, SUM(AGE) FROM STUDENT
GROUP BY (ADDRESS);
```

AGGRATION FUNCTIONS

```
SELECT ADDRESS, SUM(AGE) FROM STUDENT
GROUP BY (ADDRESS);
```

In this query, SUM(**AGE**) will be computed but not for entire table but for each address. i.e.; sum of AGE for address DELHI(18+18=36) and similarly for other address as well. The output is:

ADDRESS	**SUM(AGE)**
DELHI	36
GURGAON	18
ROHTAK	20

AGGRATION FUNCTIONS

If we try to execute the query given below, it will result in error because although we have computed SUM(AGE) for each address, there are more than 1 ROLL_NO for each address we have grouped. So it can't be displayed in result set. We need to use aggregate functions on columns after SELECT statement to make sense of the resulting set whenever we are using GROUP BY.

```
SELECT ROLL_NO, ADDRESS, SUM(AGE) FROM STUDENT GROUP BY (ADDRESS);
```

NOTE: An attribute which is not a part of GROUP BY clause can't be used for selection. Any attribute which is part of GROUP BY CLAUSE can be used for selection but it is not mandatory. But we could use attributes which are not a part of the GROUP BY clause in an aggregrate function.

SQL Data Types

1. **Binary Datatypes :** There are four subtypes of this datatype which are given below :

Data Type	Description
binary	Maximum length of 8000 bytes (Fixed-Length binary data)
varbinary	Maximum length of 8000 bytes(Variable Length binary data)
varbinary(max)	Maximum length of 231 bytes (SQL Server 2005 only). (Variable Length binary data)
image	Maximum length of 2,147,483,647 bytes(Variable Length binary data)

SQL Data Types

2. **Exact Numeric Datatype :** There are nine subtypes which are given below in the table. The table contains the range of data in a particular type.

Data Type	From	To
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	$-10^{38} + 1$	$10^{38} - 1$
numeric	$-10^{38} + 1$	$10^{38} - 1$
money	-922,337,203,685,477.5808	922,337,203,685,477.5808
smallmoney	-214,748.3648	214,748.3648

SQL Data Types

3. Approximate Numeric Datatype :
The subtypes of this datatype are given in the table with the range.

Data Type	From	To
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

SQL Data Types

4. Character String Datatype :
The subtypes are given in below table –

Data Type	Description
char	Maximum length of 8000 characters. (Fixed-Length non-Unicode Characters)
varchar	Maximum length of 8000 characters. (Variable-Length non-Unicode Characters)
varchar(max)	Maximum length of 231 characters (SQL Server 2005 only). (Variable Length non-Unicode data)
text	Maximum length of 2,147,483,647 characters (Variable Length non-Unicode data)

SQL Data Types

5. Unicode Character String Datatype :
The details are given in below table –

Data Type	Description
nchar	Maximum length of 4000 characters. (Fixed-Length Unicode Characters)
Nvarchar	Maximum length of 4000 characters. (Variable-Length Unicode Characters)
nvarchar(max)	Maximum length of 231 characters (SQL Server 2005 only). (Variable Length Unicode data)

SQL Data Types

6. Date and Time Datatype :
The details are given in below table.

Data Type	From	To
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

SQL Data Types

4. Character String Datatype :
The subtypes are given in below table –

Data Type	Description
char	Maximum length of 8000 characters. (Fixed-Length non-Unicode Characters)
varchar	Maximum length of 8000 characters. (Variable-Length non-Unicode Characters)
varchar(max)	Maximum length of 231 characters (SQL Server 2005 only). (Variable Length non-Unicode data)
text	Maximum length of 2,147,483,647 characters (Variable Length non-Unicode data)

SQL | Constraints

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

SQL | Constraints

The available constraints in SQL are:

NOT NULL: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.

UNIQUE: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.

PRIMARY KEY: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.

FOREIGN KEY: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.

CHECK: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to ensure that the value stored in a column meets a specific condition.

DEFAULT: This constraint specifies a default value for the column when no value is specified by the user.

SQL | Constraints

How to specify constraints?
We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.

SQL | Constraints

Syntax:
Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.

```
CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name, ....
);
```

sample_table: Name of the table to be created.
data_type: Type of data that can be stored in the field.
constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

SQL | Constraints

1. NOT NULL –
If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to insert a new row in the table without specifying any value to this field.
For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

SQL | Constraints

2. UNIQUE –
This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.
For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

SQL | Constraints

3. PRIMARY KEY –
Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints.
A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
```

SQL | Constraints

4. FOREIGN KEY –
Foreign Key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.
Consider the two tables as shown below:

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8532	1

SQL | Constraints

As we can see clearly that the field C_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.

Orders

O_ID	ORDER_NO	C_ID
1	2253	3
2	3325	3
3	4521	2
4	8532	1

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

SQL | Constraints

Syntax:

```
CREATE TABLE Orders
(
O_ID int NOT NULL,
ORDER_NO int NOT NULL,
C_ID int,
PRIMARY KEY (O_ID),
FOREIGN KEY (C_ID)
REFERENCES Customers(C_ID)
)
```

Customers

C_ID	NAME	ADDRESS
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

SQL Commands

(i) CHECK –
Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.
For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18.

```
CREATE TABLE Student
(

ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)

);
```

SQL Commands

(ii) DEFAULT –
This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.
For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student
(

ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18

);
```

SQL | Creating Roles

A role is created to ease setup and maintenance of the security model. It is a named group of related privileges that can be granted to the user. When there are many users in a database it becomes difficult to grant or revoke privileges to users. Therefore, if you define roles:

You can grant or revoke privileges to users, thereby automatically granting or revoking privileges.

You can either create Roles or use the system roles pre-defined.

SQL | Creating Roles

A role is created to ease setup and maintenance of the security model. It is a named group of related privileges that can be granted to the user. When there are many users in a database it becomes difficult to grant or revoke privileges to users. Therefore, if you define roles:

You can grant or revoke privileges to users, thereby automatically granting or revoking privileges.

You can either create Roles or use the system roles pre-defined.

SQL | Creating Roles

Some of the privileges granted to the system roles are as given below:

System Roles	Privileges granted to the Role
Connect	Create table, Create view, Create synonym, Create sequence, Create session etc.
Resource	Create Procedure, Create Sequence, Create Table, Create Trigger etc. The primary usage of the Resource role is to restrict access to database objects.
DBA	All system privileges

SQL | Creating Roles

Creating and Assigning a Role –

First, the (Database Administrator)DBA must create the role. Then the DBA can assign privileges to the role and users to the role.

Syntax –

```
CREATE ROLE manager;


Role created.
```

In the syntax:
'manager' is the name of the role to be created.

- Now that the role is created, the DBA can use the GRANT statement to assign users to the role as well as assign privileges to the role.
- It's easier to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user.
- If a role is identified by a password, then GRANT or REVOKE privileges have to be identified by the password.

SQL | Creating Roles

Grant privileges to a role –

```
GRANT create table, create view TO manager; Grant succeeded.
```

Grant a role to users

```
GRANT manager TO SAM, STARK; Grant succeeded.
```

Revoke privilege from a Role :

```
REVOKE create table FROM manager;
```

Drop a Role :

```
DROP ROLE manager;
```

Explanation –

Firstly it creates a manager role and then allows managers to create tables and views. It then grants Sam and Stark the role of managers. Now Sam and Stark can create tables and views. If users have multiple roles granted to them, they receive all of the privileges associated with all of the roles. Then create table privilege is removed from role ‘manager’ using Revoke.The role is dropped from the database using drop.

SQL indexes

An index is a schema object. It is used by the server to speed up the retrieval of rows by using a pointer. It can reduce disk I/O(input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements. Indexes can be created or dropped with no effect on the data. In this article, we will see how to create, delete, and uses the INDEX in the database.

SQL indexes

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and is then referred to one or more specific page numbers.

SQL indexes

Creating an Index:

Syntax:

```
CREATE INDEX index
ON TABLE column;
```

where the **index** is the name given to that index and **TABLE** is the name of the table on which that index is created and **column** is the name of that column for which it is applied.

SQL indexes

For multiple columns:

Syntax:

```
CREATE INDEX index
ON TABLE (column1, column2,.....);
```

SQL indexes

Unique Indexes:

Unique indexes are used for the maintenance of the integrity of the data present in the table as well as for the fast performance, it does not allow multiple values to enter into the table.

Syntax:

```
CREATE UNIQUE INDEX index
ON TABLE column;
```

SQL indexes

When should indexes be created:

- A column contains a wide range of values.
- A column does not contain a large number of null values.
- One or more columns are frequently used together in a where clause or a join condition.

SQL indexes

When should indexes be avoided:

- The table is small
- The columns are not often used as a condition in the query
- The column is updated frequently

SQL indexes

Removing an Index:

To remove an index from the data dictionary by using the **DROP INDEX** command.

Syntax:

```
DROP INDEX index;
```

To drop an index, you must be the owner of the index or have the **DROP ANY INDEX** privilege.

SQL indexes

Altering an Index:

To modify an existing table's index by rebuilding, or reorganizing the index.

```
ALTER INDEX IndexName
ON TableName REBUILD;
```

SQL indexes

Confirming Indexes :

You can check the different indexes present in a particular table given by the user or the server itself and their uniqueness.

Syntax:

```
select * from USER_INDEXES;
```

It will show you all the indexes present in the server, in which you can locate your own tables too.

SQL indexes

Renaming an index :

You can use the system stored procedure sp_rename to rename any index in the database.

Syntax:

```
EXEC sp_rename index_name, new_index_name, N'INDEX';
```

SQL | SEQUENCES

Sequence is a set of integers 1, 2, 3, … that are generated and supported by some database systems to produce unique values on demand.

A sequence is a user defined schema bound object that generates a sequence of numeric values.

Sequences are frequently used in many databases because many applications require each row in a table to contain a unique value and sequences provides an easy way to generate them.

The sequence of numeric values is generated in an a**scending or descending order** at defined intervals and can be configured to restart when exceeds max_value.

SQL | SEQUENCES

Syntax:

```
CREATE SEQUENCE sequence_name
START WITH initial_value
INCREMENT BY increment_value
MINVALUE minimum value
MAXVALUE maximum value
CYCLE|NOCYCLE ;
```

sequence_name: Name of the sequence.
initial_value: starting value from where the sequence starts. Initial_value should be greater than or equal to minimum value and less than equal to maximum value.
increment_value: Value by which sequence will increment itself. Increment_value can be positive or negative.
minimum_value: Minimum value of the sequence.
maximum_value: Maximum value of the sequence.
cycle: When sequence reaches its set_limit it starts from beginning.
nocycle: An exception will be thrown if sequence exceeds its max_value.

SQL | SEQUENCES

Example

Following is the sequence query creating sequence in ascending order.

•**Example 1:**

```
CREATE SEQUENCE sequence_1

start with 1
increment by 1
minvalue 0
maxvalue 100
cycle;
```

Above query will create a sequence named *sequence_1*.Sequence will start from 1 and will be incremented by 1 having maximum value 100. Sequence will repeat itself from start value after exceeding 100.

SQL | SEQUENCES

•**Example 2:**

Following is the sequence query creating sequence in descending order.

```
CREATE SEQUENCE sequence_2
start with 100
increment by -1
minvalue 1
maxvalue 100
cycle;
```

Above query will create a sequence named *sequence_2*.Sequence will start from 100 and should be less than or equal to maximum value and will be incremented by -1 having minimum value 1.

SQL | SEQUENCES

Example to use sequence : create a table named students with columns as id and name.

```
CREATE TABLE students
(
ID number(10),
NAME char(20)
);
```

Now insert values into table

```
INSERT into students VALUES(sequence_1.nextval,'Ramesh');
INSERT into students VALUES(sequence_1.nextval,'Suresh');
```

where *sequence_1.nextval* will insert id's in id column in a sequence as defined in sequence_1.

SQL | SEQUENCES

Example to use sequence : create a table named students with columns as id and name.

```
CREATE TABLE students
(
ID number(10),
NAME char(20)
);
```

Output:

```
_______________
| ID | NAME |
--------------
| 1 | Ramesh |
| 2 | Suresh |
--------------
```

Now insert values into table

```
INSERT into students VALUES(sequence_1.nextval,'Ramesh');
INSERT into students VALUES(sequence_1.nextval,'Suresh');
```

where *sequence_1.nextval* will insert id's in id column in a sequence as defined in sequence_1.

SQL Trigger | Student Database

Trigger: A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs. For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.

SQL Trigger | Student Database

Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Explanation of syntax:

1.create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2.[before | after]: This specifies when the trigger will be executed.
3.{insert | update | delete}: This specifies the DML operation.
4.on [table_name]: This specifies the name of the table associated with the trigger.
5.[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6.[trigger_body]: This provides the operation to be performed as trigger is fired

SQL Trigger | Student Database

BEFORE and AFTER of Trigger:

BEFORE triggers run the trigger action before the triggering statement is run.

AFTER triggers run the trigger action after the triggering statement is run.

SQL Trigger | Student Database

Example:
Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

SQL Trigger | Student Database

Suppose the database Schema –

```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SQL Trigger | Student Database

SQL Trigger to problem statement.

```
create trigger stud_marks

before INSERT
On
Student
for each row

set Student.total = Student.subj1 + Student.subj2 + Student.subj3,
Student.per = Student.total * 60 / 100;
```

SQL Trigger | Student Database

SQL Trigger to problem statement.

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database,

trigger will compute those two values and insert with the entered values. i.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0, 0);
Query OK, 1 row affected (0.09 sec)

mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```

In this way trigger can be created and executed in the databases.

SQL Trigger | Book Management Database

For example, given Library Book Management database schema with Student database schema. In these databases, if any student borrows a book from library then the count of that specified book should be decremented. To do so,

Suppose the schema with some data,

```
mysql> select * from book_det;
+-----+-------------+--------+
| bid | btitle      | copies |
+-----+-------------+--------+
|   1 | Java        |     10 |
|   2 | C++         |      5 |
|   3 | MySql       |     10 |
|   4 | Oracle DBMS |      5 |
+-----+-------------+--------+
4 rows in set (0.00 sec)

mysql> select * from book_issue;
+------+------+--------+
| bid  | sid  | btitle |
+------+------+--------+
1 row in set (0.00 sec)
```

To implement such procedure, in which if the system inserts the data into the book_issue database a trigger should automatically invoke and decrements the copies attribute by 1 so that a proper track of book can be maintained.

SQL Trigger | Book Management Database

Trigger for the system –

```
create trigger book_copies_deducts
after INSERT
on book_issue
for each row
update book_det set copies = copies - 1
where bid = new.bid;
```

Above trigger, will be activated whenever an insertion operation performed in a book_issue database, it will update the book_det schema setting copies decrements by 1 of current book id(bid).

```
mysql> select * from book_det;
+-----+-------------+--------+
| bid | btitle      | copies |
+-----+-------------+--------+
|   1 | Java        |     10 |
|   2 | C++         |      5 |
|   3 | MySql       |     10 |
|   4 | Oracle DBMS |      5 |
+-----+-------------+--------+
4 rows in set (0.00 sec)

mysql> select * from book_issue;
+------+------+--------+
| bid  | sid  | btitle |
+------+------+--------+
1 row in set (0.00 sec)
```

SQL Trigger | Book Management Database

Trigger for the system –

```
create trigger book_copies_deducts
after INSERT
on book_issue
for each row
update book_det set copies = copies - 1
where bid = new.bid;
```

Above trigger, will be activated whenever an insertion operation performed in a book_issue database, it will update the book_det schema setting copies decrements by 1 of current book id(bid).

Results –

```
mysql> insert into book_issue values(1, 100, "Java");
Query OK, 1 row affected (0.09 sec)

mysql> select * from book_det;
+-----+-------------+--------+
| bid | btitle      | copies |
+-----+-------------+--------+
|   1 | Java        |      9 |
|   2 | C++         |      5 |
|   3 | MySql       |     10 |
|   4 | Oracle DBMS |      5 |
+-----+-------------+--------+
4 rows in set (0.00 sec)

mysql> select * from book_issue;
+------+------+--------+
| bid  | sid  | btitle |
+------+------+--------+
|    1 |  100 | Java   |
+------+------+--------+
1 row in set (0.00 sec)
```

SQL Trigger | Book Management Database

Before –

```
mysql> select * from book_det;
+-----+-------------+--------+
| bid | btitle      | copies |
+-----+-------------+--------+
|   1 | Java        |     10 |
|   2 | C++         |      5 |
|   3 | MySql       |     10 |
|   4 | Oracle DBMS |      5 |
+-----+-------------+--------+
4 rows in set (0.00 sec)

mysql> select * from book_issue;
+------+------+--------+
| bid  | sid  | btitle |
+------+------+--------+
1 row in set (0.00 sec)
```

```
mysql> insert into book_issue values(1, 100, "Java");
Query OK, 1 row affected (0.09 sec)

mysql> select * from book_det;
+-----+-------------+--------+
| bid | btitle      | copies |
+-----+-------------+--------+
|   1 | Java        |      9 |
|   2 | C++         |      5 |
|   3 | MySql       |     10 |
|   4 | Oracle DBMS |      5 |
+-----+-------------+--------+
4 rows in set (0.00 sec)

mysql> select * from book_issue;
+------+------+--------+
| bid  | sid  | btitle |
+------+------+--------+
|    1 |  100 | Java   |
+------+------+--------+
1 row in set (0.00 sec)
```

After –

As above results show that as soon as data is inserted, copies of the book deducts from the book schema in the system.