Introduction to SQL

Introduction to SQL

Structure Query Language(SQL) is a database query language used for storing and managing data in Relational DBMS. Today almost all RDBMS **(MySql, Oracle, Infomix, Sybase, MS Access)** use SQL as the standard database query language. SQL is used to perform all types of data operations in RDBMS.

SQL Command

SQL defines following ways to manipulate data stored in an RDBMS.

- **Data Definition Language (DDL)**
- **Data Manipulation Languate (DML)**
- **Transaction Control Language (TCL)**
- **Data Control Language (DCL)**
- **Data Query Language (DQL)**

Introduction to SQL

Data Definition Language (DDL)

This includes changes to the structure of the table like creation of table, altering table, deleting a table etc. All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
Create	To create a new Table or Database
Alter	For altering the database or table
Truncate	To truncate (delete) all the data from the table
Drop	To drop a table
Rename	To rename a table

Introduction to SQL

Data Manupulation Language (DML)

DML commands are used for manipulating the data stored in the table and not the table itself. DML commands are **not** auto-committed. It means changes are not permanent to database, they can be rolled back.

Command	Description
insert	To insert a new row in the table
Update	To update existing data in the table
Delete	To truncate (delete) all the data from the table
Merge	merging two rows or two tables

Introduction to SQL

Transaction Control Language (TCL)

These commands are to keep a check on other commands and their affect on the database. These commands can annul changes made by other commands by rolling the data back to its original state. It can also make any temporary change permanent.

Command	Description
Commit	To permanent save the changes in the database
rollback	To undo the changes
savepoint	To save the changes temporarily

Introduction to SQL

Data Control Language (DCL)

Data control language are the commands to grant and take back authority from any database user.

Command	Description
grant	To grant permission on database objects
revoke	To revoke the permission on the database objects

Introduction to SQL

Data Query Language (DQL)

Data query language is used to fetch data from tables based on conditions that we can easily apply.

Command	Description
Select	To retrieve data from one or more tables

Create Command

CREATE command is used to create Database and its Tables.

Syntax to create a database :

CREATE DATABASE database_name;

Example to create a database :

CREATE DATABASE student;

Example to create a table :

```
CREATE TABLE users(
id          INT             NOT NULL    AUTO_INCREMENT PRIMARY KEY ,
name        VARCHAR( 25 )   NOT NULL ,
last_name   VARCHAR( 25 )   NOT NULL ,
age         INT NULL  )
```

Data Types

Here we have listed some of the commonly used datatypes used for columns in tables.

Data Type	Use
INT	used for columns which will store integer values.
FLOAT	used for columns which will store float values.
DOUBLE	used for columns which will store float values.
VARCHAR	used for columns which will be used to store characters and integers, basically a string.
CHAR	used for columns which will store char values(single character).
DATE	To store Date and Time
TEXT	used for columns which will store text which is generally long in length. For example, if you create a table for storing profile information of a social networking website, then for about me section you can have a column of type TEXT.

ALTER Command

ALTER command is used for altering the table structure, such as,

- to add a column to existing table
- to rename any existing column
- to change datatype of any column or to modify its size.
- to drop a column from the table.

To ADD a new Column:

```
ALTER TABLE table_name ADD (
  column_name datatype);
```

Example:

```
ALTER TABLE student ADD (
  address VARCHAR(200)
);
```

ALTER Command

To ADD multiple new Columns:

```
ALTER TABLE table_name ADD (
   column_1 datatype1,
    column_2 datatype2);
```

Example:

```
ALTER TABLE student ADD (
   father_name VARCHAR(100)
   mother_name varchar(100)
);
```

To Add Column with default value

```
ALTER TABLE table_name ADD(
   column-name1 datatype1 DEFAULT some_value
);
```

Example:

```
ALTER TABLE student ADD(
   dob DATE DEFAULT '01-Jan-99'
);
```

ALTER Command

To MODIFY an existing Columns:

```
ALTER TABLE table_name modify(
  column_name datatype
);
```

Example:

```
ALTER TABLE student MODIFY(
  address varchar(300));
```

To Rename a Column

```
ALTER TABLE `sampledb`.`student`
CHANGE COLUMN `name` `stu_name` VARCHAR(30) NOT NULL;
```

Example:

```
ALTER TABLE student CHANGE COLUMN
  'address' 'location' VARCHAR(100) NULL;
```

ALTER Command

To Drop a Column:

ALTER TABLE table_name DROP(
column_name);

Example:

ALTER TABLE student DROP(
address);

TRUNCATE & DROP Commands

TRUNCATE command removes all the records from a table. But this command will not destroy the table's structure. When we use TRUNCATE command on a table its (auto-increment) primary key is also initialized.

TRUNCATE TABLE table_name;

Example: TRUNCATE TABLE test_table;

DROP command is used to drop a table from the database permanently or to drop a database from the server.

DROP TABLE Table_Name;

Example: DROP TABLE Student;

DROP DATABASE database_name;

Example: DROP DATABASE sample_db;

RENAME Command

RENAME command is used to set a new name for any existing table.

RENAME TABLE old_table_name to new_table_name;

Example: RENAME TABLE student to students_info;

SQL Constraints

SQL Constraints are rules used to limit the type of data that can go into a table, to maintain the accuracy and integrity of the data inside table. Constraints can be,

- **Column level constraints:** Limits only column data.
- **Table level constraints:** Limits whole table data.

Constraints are used to make sure that the integrity of data is maintained in the database. Following are the most used constraints that can be applied to a table.

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

SQL Constraints

NOT NULL Constraint : NOT NULL constraint restricts a column from having a NULL value. Once NOT NULL constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value. One important point to note about this constraint is that it cannot be defined at table level.

Example

```
CREATE TABLE Student (
    student_id    INT           NOT NULL PRIMARY KEY,
    name          VARCHAR(30)   NOT NULL,
    DOB           DATE          NOT NULL,
    Email         VARCHAR(50)   NULL,
    phone_no      CHAR(10)      NULL

)
```

SQL Constraints

UNIQUE Constraint : UNIQUE constraint ensures that a field or column will only have unique values. A UNIQUE constraint field will not have duplicate data. This constraint can be applied at column level or table level.

Example

```
CREATE TABLE Student(s_id int NOT NULL UNIQUE, Name varchar(60), Age int);

ALTER TABLE Student ADD UNIQUE(s_id);
```

SQL Constraints

Primary Key Constraint : Primary key constraint uniquely identifies each record in a database. A Primary Key must contain unique value and it must not contain null value. Usually, Primary Key is used to index the data inside the table.

Example

```
CREATE table Student (
        s_id int PRIMARY KEY,
         Name varchar(60) NOT NULL,
        Age int
);


ALTER table Student ADD PRIMARY KEY (s_id);
```

SQL Constraints

Foreign Key Constraint : FOREIGN KEY is used to relate two tables. FOREIGN KEY constraint is also used to restrict actions that would destroy links between tables. The restrictions are RESTRICT, CASCADE, NO ACTION, SET DEFAULT and SET NULL.

Example

```
create table subject (
        subject_id tinyint not null primary key,
        subject_name   varchar(20)       not null    );

create table student_subject  (
        student_id        int not null,
        subject_id tinyint not null,
        marks       tinyint null,       grade       char(1) null,
        primary key (student_id, subject_id),
        foreign key (student_id) references student(student_id)  on delete RESTRICT,
        foreign key (subject_id) references subject(subject_id)  on delete RESTRICT
);
```

SQL Constraints

CHECK Constraint : CHECK constraint is used to restrict the value of a column between a range. It performs check on the values, before storing them into the database. Its like condition checking before saving data into a column.

Example

```
CREATE table Student(
   s_id int NOT NULL CHECK(s_id > 0),
   Name varchar(60) NOT NULL,
   Age int  );

ALTER TABLE Student ADD CHECK (age > 0);
```

Index

- A database index is a data structure that improves the speed of operations in a table. Indexes can be created using one or more columns, providing the basis for both rapid random lookups and efficient ordering of access to records.
- While creating index, it should be taken into consideration which all columns will be used to make SQL queries and create one or more indexes on those columns.
- The INSERT, UPDATE and DELETE statements take more time on tables having indexes, whereas the SELECT statements become fast on those tables. The reason is that while doing insert or update, a database needs to insert or update the index values as well.

Index

Simple and Unique Index : A unique index means that two rows cannot have the same index value. Here is the syntax to create an Index on a table.

```
CREATE UNIQUE INDEX index_name ON table_name ( column1, column2,...);
```

```
## Creating a UNIQUE Index on Table Columns
create unique index student_phone_idx on student(phone_no);
```

Non-Unique Index: A non-unique index allows duplicate rows in the table.

```
## Non-unique Index
create index student_name_idx on student(stu_name);

## Dropping an index
DROP INDEX <Index_Name> on <Table_Name>;

DROP INDEX student_name_idex on student;
```

Index

Column Prefix Key Parts : For string columns, indexes can be created that use only the leading part of column values, using col_name(length).

CREATE INDEX part_of_name_idx on student (stu_name{10));

Functional Key Parts: A "normal" index indexes column values or prefixes of column values. For example, in the following table, the index entry for a given t1 row includes the full col1 value and a prefix of the col2 value consisting of its first 10 characters:

```
CREATE TABLE t1 (col1 INT, col2 INT, INDEX func_index ((ABS(col1))));
CREATE INDEX idx1 ON t1 ((col1 + col2));
CREATE INDEX idx2 ON t1 ((col1 + col2), (col1 - col2), col1);
ALTER TABLE t1 ADD INDEX ((col1 * 40) DESC);
```

SEQUENCE

A sequence is a set of integers 1, 2, 3, ... that are generated in order on a specific demand. Sequences are frequently used in the databases because many applications require each row in a table to contain a unique value and sequences provide an easy way to generate them.

AUTO_INCREMENT Column : The simplest way in MySQL to use Sequences is to define a column as AUTO_INCREMENT.

Example

```
CREATE TABLE hobbies (
        hobby_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
        PRIMARY KEY (hobby_id),
        hobby_name VARCHAR(30) NOT NULL # type of insect
);
```

When the rows are inserted, there is no need to supply the values for auto_increment columns. The values will be inserted automatically starting from 1 incremented by 1.

SELECT Query

SELECT query is used to retrieve data from a table. It is the most used SQL query. We can retrieve complete table data, or partial by specifying conditions using the WHERE clause.

Syntax

```
SELECT
        column_name1,
        column_name2,
        column_name3,
        ...
        column_nameN
FROM
        table_name;
```

Example

```
SELECT
        name, age
FROM
        students;
```

SELECT Query

Student_id	Name	Age	City
1	Arun	17	Chennai
2	Ashok	18	Bangalore
3	Vikas	16	Pune
4	Sekar	20	Mumbai

Let us Create a Student Table with above columns and insert the rows into the table.

To Select ALL Columns

```
SELECT *
FROM
    student;
```

To Select few Columns

```
SELECT
    name, age
FROM
    student;
```

WHERE Clause

WHERE Clause is used to filter or select certain data based on some condition.

Syntax

```
SELECT
    column_name1,
    column_name2
FROM
    table_name
WHERE
    [condition];
```

Example

```
SELECT
    name, age
FROM
    student
WHERE
    age = 16;
```

WHERE Clause

Operators for WHERE clause condition

Following is a list of operators that can be used while specifying the WHERE clause condition.

Operator	Description
=	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Less than or Equal to
>=	Greate than or Equal to
BETWEEN	Between a specified range of values
LIKE	This is used to search for a pattern in value.
IN	In a given set of values

DISTINCT Clause

DISTINCT Clause is used to remove the duplicates from the data.

Syntax

```
SELECT    DISTINCT column-list
FROM      table-name
ORDER BY ASC | DESC;
```

Example

```
SELECT DISTINCT age
FROM      student
ORDER BY age DESC:

SELECT DISTINCT salary
FROM employee;
```

LIKE Clause

LIKE clause is used in the condition in SQL query with the WHERE clause. LIKE clause compares data with an expression using wildcard operators to match pattern given in the condition.

Wildcard operators

There are two wildcard operators that are used in LIKE clause.

- Percent sign %: represents zero, one or more than one character.
- Underscore sign _: represents only a single character.

Example with %

```
SELECT
    name, age
FROM
    student
WHERE
    name LIKE “A%”;
```

LIKE Clause

Example with % and _

```
SELECT
    name, age
FROM
    student
WHERE
    name LIKE "_s%";
```

Will retrieve the data with name has second character 's' irrespective of First Character.

```
SELECT
    name, age
FROM
    student
WHERE
    name LIKE "%n";
```

Will retrieve the data with name has last character 'n'

ORDER BY Clause

ORDER BY is used to display the data in a sorted order based on certain attributes. It has ASC and DESC orders. The ASC and DESC can be mixed in a single query for different columns.

Syntax

```
SELECT    column-list|*
FROM      table-name
ORDER BY ASC | DESC;
```

Example

```
SELECT * from student
ORDER BY name ASC;

SELECT * from student
ORDER BY name ASC, age DESC;
```

GROUP BY Clause

Group by clause is used to group the results of a SELECT query based on one or more columns. It is also used with SQL functions to group the result from one or more tables.

Syntax

```
SELECT    column_name, function(column_name)
FROM      table_name
WHERE     condition
GROUP BY column_name;
```

Example

```
SELECT name , age
 FROM     employee
GROUP BY salary;

SELECT name, age FROM student
WHERE    age > 25
GROUP BY salary;
```

Employee
Emp_id
Emp_name
age
City
Salary

HAVING Clause

HAVING clause is used with SQL Queries to give more precise condition for a statement. It is used to mention condition in Group by based SQL queries, just like WHERE clause is used with SELECT query.

Syntax

```
SELECT column_name, function(column_name
FROM t    able_name
WHERE    column_name condition
GROUP BY column_name
HAVING   function(column_name) condition;
```

Example

```
SELECT    customer_name, sum(previous_bal)
FROM      sales
GROUP BY customer_name
HAVING sum(previous_bal) > 3000;
```

Sales
Order_id
Order_dt
Customer_name
Previous_bal


thank
you