

# Introduction to SQL

## What is SQL?

- SQL stands for Structured Query Language
- SQL lets you access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

# What Can SQL do?

- SQL can execute queries against a database
- SQL can retrieve data from a database
- SQL can insert records in a database
- SQL can update records in a database
- SQL can delete records from a database
- SQL can create new databases
- SQL can create new tables in a database
- SQL can create stored procedures in a database
- SQL can create views in a database
- SQL can set permissions on tables, procedures, and views

# Basic SQL Query and Examples of SQL Queries:

- Data Types

## DDL Commands

- CREATE Table
- Alter Table
- Truncate Table
- Drop Table

## DML Commands

- Select
- Distinct
- Where clause
- Update
- Column aliasing

# Create Table Command :


- The create table command defines each column of the table uniquely. Each column has a minimum of three attributes, a name, data type and size of column.
- Each table column definition is separated by a comma and SQL statement is terminated with a semicolon.

# Rules for creating table :

- A name can have maximum up to 30 character.
- Alphabets from A-Z, a-z and numbers from 0-9 are allowed.
- A name should begin with an alphabet.
- The use of the special character like '\_' is allowed and also recommended. (special character like \$, # are allowed)
- SQL reserved words are not allowed like create, select....

Syntax :

```
CREATE TABLE <table name>  
<column name1> <Data type> (<size>),  
<column name2> <Data type> (<size>);
```

 Oracle SQL\*Plus

File Edit Search Options Help

SQL\*Plus: Release 10.2.0.3.0 - Production on Fri Jan 18 10:31:17 2019

Copyright (c) 1982, 2006, Oracle. All Rights Reserved.

Connected to:

Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - Production  
With the Partitioning, OLAP and Data Mining options

SQL> create table student (Roll\_no Number(2), Name varchar(20), Class varchar(5));

Table created.

SQL>

# Inserting data into table :

- After creating a table, we have to insert values within that table.
- The SQL command INSERT is used for inserting data into the table. While inserting rows you may insert the values for each attribute of the table.
- We can use insert command as following ways-

Syntax :

1) INSERT INTO <Table Name> Values(expression1, expression2,.....);

OR

2) INSERT INTO <Table name> (Column1, Column2, Column3,.....)  
Values (expr1, expr2, expr3, ....);

OR

3) INSERT INTO <Table name> values ('& column1', '&column2', '&column3', .....);

```
SQL> insert into student values (1,'AMAR','BSCCS');
```

```
1 row created.
```

```
SQL> INSERT INTO student (Roll_no, Name, Class) values (2,'VEEDA','BSCCS');
```

```
1 row created.
```

```
SQL> INSERT INTO STUDENT VALUES('&ROLL_NO','&NAME','&CLASS');
```

```
Enter value for roll_no: 3
```

```
Enter value for name: mANISHA
```

```
Enter value for class: BSCCS
```

```
old 1: INSERT INTO STUDENT VALUES('&ROLL_NO','&NAME','&CLASS')
```

```
new 1: INSERT INTO STUDENT VALUES('3','mANISHA','BSCCS')
```

```
1 row created.
```

```
SQL> |
```



# Viewing or selecting data :

- Once data has been inserted into a table we can view from that table according to users requirement. The SELECT command is used to retrieve rows from the table.
- The SELECT command is used for different purpose as follows:


## All Rows & All Columns:

To view all records with all columns the SELECT command is used as follows,

Syntax – **SELECT \* from <table name>**

e.g. SELECT \* from Student;

Above statement display all records from table Student.



```
SQL> select *from student;
```

ROLL_NO	NAME	CLASS
1	AMAR	BSCCS
2	VEEDA	BSCCS
3	MANISHA	BSCCS

```
SQL> |
```



## Filtering Table Data :

While viewing data from a table it is rare that all the data from the table will be required each time. Hence, SQL provides a method of filtering table data that is not required.

The ways of filtering table data are :

- Selected columns and all rows.
- Selected rows and all columns.
- Selected columns and selected rows.

# Selected columns and all rows :

If you want to display the specific columns but all rows from the particular table the select command can be used as follows-

**Syntax – SLECT <column name>, <column name2> FROM<table name>;**

e.g. Consider table Student having column Roll No., Name and Address and you want to retrieve the content of column Roll No. from student table, then the statement can be written as-

**SQL>SELECT Roll No. from Student.**

The above command display all the rows but selected column i.e. Roll No. from student table.



```
SQL> select roll_no from student;
```

```
ROLL_NO
```

```
-----
```

```
1
```

```
2
```

```
3
```

```
SQL>
```



```
SQL> select roll_no, class from student;
```

ROLL_NO	CLASS
1	BSCCS
2	BSCCS
3	BSCCS

```
SQL>
```



# Selected Rows and all column:

- If you want to select specific rows and all columns from particular table then we have to write specific condition. To write specific condition Oracle provides the option of using a WHERE clause.

**Syntax – SELECT \*FROM<Table Name> WHERE<condition>;**

- Here <condition> is always quantified as <column name = value> or with the condition we can use all logical & arithmetic operators.
- e.g. Suppose you want to display the student information whose Roll No = 3 from student table, then SELECT command can be used as-
- `SELECT * FROM Student WHERE Roll No=3;`





```
SQL> select *from student where roll_no=3;
```

ROLL_NO	NAME	CLASS
3	MANISHA	BSCCS

```
SQL> |
```

# Selected Column & Selected Rows :

- If we want to view specific data set from the table & also a selected number of columns then we can use SELECT command as-

```
SELECT <column name1>, <column name2> FROM <Table Name> WHERE <conditions>;
```

- e.g. Consider you want to access or retrieve the students Roll No, Name whose Roll No is greater than 5, then the SELECT command can be written as-

```
SQL>SELECT Roll No, Name from Student WHERE Roll No>5;
```



```
SQL> select Roll_no, name from student where roll_no>2;
```

ROLL_NO	NAME
---------	------

3	MANISHA
---	---------

# Eliminating Duplicate Rows when using a SELECT command

- A table could hold duplicate rows. In such case to view only unique rows the DISTINCT clause can be used.
- The DISTINCT clause allows removing duplicates from the result set. The DISTINCT can only be used with select statement.
- The DISTINCT clause scans through the values of columns specified & displays only unique values from amongst them.

**Syntax: SELECT DISTINCT <column name1>, <column name2> FROM <Table Name>;**

- The SELECT DISTINCT \*SQL Syntax scans through entire rows, & eliminates rows that have exactly the same contents in each column.

**Syntax : SELECT DISTINCT \* FROM <Table Name>;**

- e.g.1 – SELECT DISTINCT Roll No FROM Student;

```
SQL> SELECT DISTINCT *FROM STUDENT  
2 ;
```

ROLL_NO	NAME	CLASS
1	AMAR	BSCCS
3	MANISHA	BSCCS
2	VEEDA	BSCCS

```
SQL> |
```



# Updating the contents of a table :

- The update command is used to change or modify data values in a table.
- The verb update in SQL is used to either update:  
All the rows from a table.
- OR  
A select set of rows from a table.

## Updating All Rows :

- The UPDATE statement updates columns in the existing tables with new values. The SET clause indicated which column data should be modified & the new values that they should hold.
- The WHERE clause, if given, specifies which rows should be updated. Otherwise, all table rows are updated.

**Syntax : UPDATE<Table Name> SET<Column Name 1>=<expression>, <Column Name 2> = <expression2>;**

- e.g. consider student table having Roll No, Name & Class, change the class to BCSTY;

**SQL>UPDATE Student SET Class = BCSTY;**



```
SQL> UPDATE STUDENT SET CLASS=BCSII;  
UPDATE STUDENT SET CLASS=BCSII
```

```
*
```

```
ERROR at line 1:
```

```
ORA-00904: "BCSII": invalid identifier
```

```
SQL> UPDATE STUDENT SET CLASS='BCSII';
```

```
4 rows updated.
```

```
SQL> |
```



```
SQL> SELECT *FROM STUDENT;
```

ROLL_NO	NAME	CLASS
1	AMAR	BSCCS
2	VEEDA	BSCCS
3	MANISHA	BSCCS
1	AMAR	BSCCS

BEFORE UPDATE

AFTER UPDATE

```
SQL> SELECT *FROM STUDENT;
```

ROLL_NO	NAME	CLASS
1	AMAR	BCSII
2	VEEDA	BCSII
3	MANISHA	BCSII
1	AMAR	BCSII

# Update Record Conditionally

Syntax : UPDATE <Table Name> SET <column name1> = <expr1> <column name2> = <expr2>  
WHERE <condition>;

► e.g. update the student name John to Josef.

```
SQL>UPDATE Student SET Name='MOHINI' WHERE  
Name='MANISHA'
```

SQL> SELECT \*FROM STUDENT;

ROLL_NO	NAME	CLASS
1	AMAR	BCSII
2	VEEDA	BCSII
3	MANISHA	BCSII
1	AMAR	BCSII

SQL> UPDATE STUDENT SET NAME='MOHINI' WHERE NAME='MANISHA';

0 rows updated.

SQL> UPDATE STUDENT SET NAME='VIJAY' WHERE NAME='VEEDA';

1 row updated.

SQL> SELECT \* FROM STUDENT;

ROLL_NO	NAME	CLASS
1	AMAR	BCSII
2	VIJAY	BCSII
3	MANISHA	BCSII
1	AMAR	BCSII

SQL> |

# Column Aliasing

- ▶ You can rename a column temporarily by giving another name known as Alias. The use of alias rename the table column temporary and the actual name does not change in the database.
- ▶ Syntex.

```
Select column_name AS alias_name  
From table_name  
Where [condition];
```



```
SQL> SELECT ROLL_NO AS REG_NO FROM STUDENT;
```

```
  REG_NO  
-----  
      1  
      2  
      3  
      1
```

```
SQL> select roll_no, name, class from student;
```

ROLL_NO	NAME	CLASS
1	AMAR	BCSII
2	VIJAY	BCSII
3	MANISHA	BCSII
1	AMAR	BCSII

```
SQL> select roll_no, name as STU_name, class from student;
```

ROLL_NO	STU_NAME	CLASS
1	AMAR	BCSII
2	VIJAY	BCSII
3	MANISHA	BCSII
1	AMAR	BCSII



# Altering Table Structure

- The structure of table can be modified using ALTER TABLE command. ALTER table allows changing the structure of a existing table. We can add or delete columns, change the data types of column, rename table.

## Adding NEW Column-

Syntax:- Alter table <Table Name> ADD (<New Column> <Data Type><Size>);

```
SQL> Alter table student ADD (CITY varchar (10));
```

```
Table altered.
```

```
SQL> desc student
```

Name	Null?	Type
ROLL_NO		NUMBER(2)
NAME		VARCHAR2(20)
CLASS		VARCHAR2(5)
CITY		VARCHAR2(10)

```
SQL> select *from Student;
```

ROLL_NO	NAME	CLASS	CITY
1	AMAR	BCSII	
2	VIJAY	BCSII	
3	MANISHA	BCSII	
1	AMAR	BCSII	



# Dropping Column from a table:

Syntax:- Alter table <table name> drop column <column name>;

```
SQL> alter table student drop column city;
```

```
Table altered.
```

```
SQL> desc student
```

Name	Null?	Type
ROLL_NO		NUMBER(2)
NAME		VARCHAR2(20)
CLASS		VARCHAR2(5)

```
SQL> select *from student;
```

ROLL_NO	NAME	CLASS
1	AMAR	BCSII
2	VIJAY	BCSII
3	MANISHA	BCSII
1	AMAR	BCSII

# Modifying Existing Column:

Syntax: Alter Table <table Name> Modify (<column name>  
<New data type> <new size>);

```
SQL> alter table student add(city number (10));
```

```
Table altered.
```

```
SQL> desc student;
```

Name	Null?	Type
ROLL_NO		NUMBER(2)
NAME		VARCHAR2(20)
CLASS		VARCHAR2(5)
CITY		NUMBER(10)

```
SQL> alter table student modify (city varchar (20));
```

```
Table altered.
```

```
SQL> desc student;
```

Name	Null?	Type
ROLL_NO		NUMBER(2)
NAME		VARCHAR2(20)
CLASS		VARCHAR2(5)
CITY		VARCHAR2(20)

## Renaming Tables:

Following is the syntax to rename the table.

Syntax: Rename <Table name> to <New table Name>

```
SQL> rename student to stu_info;
```

```
Table renamed.
```

```
SQL> desc student;
```

```
ERROR:
```

```
ORA-04043: object student does not exist
```

```
SQL> desc stu_info;
```

Name	Null?	Type
ROLL_NO		NUMBER(2)
NAME		VARCHAR2(20)
CLASS		VARCHAR2(5)
CITY		VARCHAR2(20)

```
SQL> |
```

## Truncating tables:

Truncate table empties a table completely it is equivalent to delete all rows.

**Syntax:- Truncate table <Table Name>;**

It is different from delete because it much faster than delete and not transaction safe means the number of rows deleted not returned.

```
SQL> TRUNCATE TABLE STU_INFO;
```

```
Table truncated.
```

```
SQL> SELECT * FROM STU_INFO;
```

```
no rows selected
```

```
SQL>
```



## DESTROYING Table :

Syntax:- Drop Table <Table Name>;

```
SQL> DROP TABLE STU_INFO;
```

```
Table dropped.
```

# Checking privileges

```
SQL> SELECT *FROM SESSION_PRIVS;  
PRIVILEGE  
-----  
CREATE SESSION  
UNLIMITED TABLESPACE  
CREATE TABLE  
CREATE CLUSTER  
CREATE SEQUENCE  
CREATE PROCEDURE  
CREATE TRIGGER  
CREATE TYPE  
CREATE OPERATOR  
CREATE INDEXTYPE  
  
10 rows selected.
```

## Granting permission to system user

```
SQL> CONN SYS AS SYSDBA;  
Enter password: *****  
Connected.
```

```
SQL> GRANT CREATE VIEW TO SYSTEM;  
Grant succeeded.
```

# Creating View

```
SQL> SELECT *FROM STU;
```

ROLNO	NAME	CITY
1	amar	LATUR
2	ANANT	PUNE
3	MANOJ	LATUR
4	SANDIP	PUNE

```
SQL> CREATE VIEW LATUR AS SELECT ROLNO, NAME, CITY FROM STU WHERE CITY='LATUR';
```

View created.

```
SQL> SELECT * FROM LATUR;
```

ROLNO	NAME	CITY
1	amar	LATUR
3	MANOJ	LATUR



## Creating View

```
SQL> CREATE VIEW PUNE AS SELECT ROLNO, NAME, CITY FROM STU WHERE CITY='PUNE';
```

View created.

```
SQL> SELECT * FROM PUNE;
```

ROLNO	NAME	CITY
2	ANANT	PUNE
4	SANDIP	PUNE

## Inserting the data into view

```
SQL> SELECT * FROM PUNE;
```

ROLNO	NAME	CITY
2	ANANT	PUNE
4	SANDIP	PUNE

```
SQL> INSERT INTO PUNE VALUES (5, 'SHITAL', 'PUNE');
```

```
1 row created.
```

```
SQL> INSERT INTO PUNE VALUES(6, 'DHIRAJ', 'PUNE');
```

```
1 row created.
```

```
SQL> SELECT * FROM PUNE;
```

ROLNO	NAME	CITY
2	ANANT	PUNE
4	SANDIP	PUNE
5	SHITAL	PUNE
6	DHIRAJ	PUNE

## UPDATING the data of view

```
SQL> SELECT * FROM PUNE;
```

ROLNO	NAME	CITY
2	ANANT	PUNE
4	SANDIP	PUNE
5	SHITAL	PUNE
6	DHIRAJ	PUNE

```
SQL> UPDATE PUNE SET NAME='MOHAN' WHERE ROLNO=5;
```

```
1 row updated.
```

```
SQL> SELECT *FROM PUNE;
```

ROLNO	NAME	CITY
2	ANANT	PUNE
4	SANDIP	PUNE
5	MOHAN	PUNE
6	DHIRAJ	PUNE

# Deleting Row from view

```
SQL> SELECT *FROM PUNE;
```

ROLNO	NAME	CITY
2	ANANT	PUNE
4	SANDIP	PUNE
5	MOHAN	PUNE
6	DHIRAJ	PUNE

```
SQL> DELETE FROM PUNE WHERE ROLNO=5;
```

```
1 row deleted.
```

```
SQL> SELECT *FROM PUNE;
```

ROLNO	NAME	CITY
2	ANANT	PUNE
4	SANDIP	PUNE
6	DHIRAJ	PUNE

# Aggregate Operators Group by & order by clause

- Aggregate function often need an added GROUP BY statement.
- **GROUP BY Statement**
- The group by statement is used in conjunction with the aggregate function to group the result set by one or more columns.
- Syntax:

```
SELECT column_name,  
aggregate_function (column_name)  
FROM table_name  
Where column_name operator value  
GROUP BY column_name
```

```
SQL> select *from stu;
```

ROLNO	NAME	CITY	FEES
1	amar	LATUR	100
2	ANANT	PUNE	100
3	MANOJ	LATUR	100
4	SANDIP	PUNE	100
6	DHIRAJ	PUNE	100

```
SQL> select city, sum(fees) from stu group by city;
```

CITY	SUM(FEES)
LATUR	200
PUNE	300

```
SQL> select city, avg(fees) from stu group by city;
```

CITY	AUG(FEES)
LATUR	100
PUNE	100



```
SQL> select city, count(rolno) from stu group by city;
```

CITY	COUNT(ROLNO)
LATUR	2
PUNE	3

```
SQL> select city, count(name) from stu group by city;
```

CITY	COUNT(NAME)
LATUR	2
PUNE	3

```
SQL> alter table stu add (marks number (2));
```

```
Table altered.
```

```
SQL> update stu set marks=23 where rolno=1;
```

```
1 row updated.
```

```
SQL> update stu set marks=20 where rolno=2;
```

```
1 row updated.
```

```
SQL> update stu set marks=15 where rolno=3;
```

```
update stu set marks=15 where rolno=3
```

```
*
```

```
ERROR at line 1:
```

```
ORA-00904: "ROLNO": invalid identifier
```

```
SQL> update stu set marks =15 where rolno=3;
```

```
1 row updated.
```

```
SQL> update stu set marks=12 where rolno=6;
```

```
1 row updated.
```

```
SQL> update stu set marks=19 where rolno=4;
```

```
1 row updated.
```



```
SQL> select *from stu;
```

ROLNO	NAME	CITY	FEES	MARKS
1	amar	LATUR	100	23
2	ANANT	PUNE	100	20
3	MANOJ	LATUR	100	15
4	SANDIP	PUNE	100	19
6	DHIRAJ	PUNE	100	12

```
SQL> select city, avg(marks) from stu group by city;
```

CITY	AUG(MARKS)
LATUR	19
PUNE	17

# Order BY

```
SQL> select rolno, name, city, fees from stu order by city asc;
```

ROLNO	NAME	CITY	FEES
1	amar	LATUR	100
3	MANOJ	LATUR	100
6	DHIRAJ	PUNE	100
4	SANDIP	PUNE	100
2	ANANT	PUNE	100

```
SQL> select rolno, name, city, fees, marks from stu order by name asc;
```

ROLNO	NAME	CITY	FEES	MARKS
2	ANANT	PUNE	100	20
6	DHIRAJ	PUNE	100	12
3	MANOJ	LATUR	100	15
4	SANDIP	PUNE	100	19
1	amar	LATUR	100	23

```
SQL> select rolno, name, marks from stu order by marks desc;
```

ROLNO	NAME	MARKS
1	amar	23
2	ANANT	20
4	SANDIP	19
3	MANOJ	15
6	DHIRAJ	12

```
SQL> select rolno, name, marks from stu order by name desc;
```

ROLNO	NAME	MARKS
1	amar	23
4	SANDIP	19
3	MANOJ	15
6	DHIRAJ	12
2	ANANT	20