http://beginner-sql-tutorial.com/

SQL Tutorial

SQL (Structured Query Language) is used to modify and access data or information from a storage area called database. This beginner sql tutorial website teaches you the basics of SQL and how to write SQL queries. I will be sharing my knowledge on SQL and help you learn SQL better. The sql concepts discussed in this tutorial can be applied to most of database systems. The syntax used to explain the concepts is similar to the one used in Oracle database.

SQL Introduction

SQL stands for “Structured Query Language” and can be pronounced as “SQL” or “sequel – (Structured English Query Language)”. It is a query language used for accessing and modifying information in the database. IBM first developed SQL in 1970s. Also it is an ANSI/ISO standard. It has become a Standard Universal Language used by most of the relational database management systems (RDBMS). Some of the RDBMS systems are: Oracle, Microsoft SQL server, Sybase etc. Most of these have provided their own implementation thus enhancing it's feature and making it a powerful tool. Few of the sql commands used in sql programming are SELECT Statement, UPDATE Statement, INSERT INTO Statement, DELETE Statement, WHERE Clause, ORDER BY Clause, GROUP BY Clause, ORDER Clause, Joins, Views, GROUP Functions, Indexes etc.

In a simple manner, SQL is a non-procedural, English-like language that processes data in groups of records rather than one record at a time. Few functions of SQL are:

- store data
- modify data
- retrieve data
- modify data
- delete data
- create tables and other database objects
- delete data

SQL Commands:

SQL commands are instructions used to communicate with the database to perform specific task that work with data. SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users. SQL commands are grouped into four major categories depending on their functionality:

- **Data Definition Language (DDL)** - These SQL commands are used for creating, modifying, and dropping the structure of database objects. The commands are CREATE, ALTER, DROP, RENAME, and TRUNCATE.

- **Data Manipulation Language (DML)** - These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

- **Transaction Control Language (TCL)** - These SQL commands are used for managing changes affecting the data. These commands are COMMIT, ROLLBACK, and SAVEPOINT.

- **Data Control Language (DCL)** - These SQL commands are used for providing security to database objects. These commands are GRANT and REVOKE.

SQL SELECT Statement

The most commonly used SQL command is SELECT statement. The SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT Statement, you must specify the column(s) name and the table name. The whole query is called SQL SELECT Statement.

Syntax of SQL SELECT Statement:

```
SELECT column_list FROM table-name
[WHERE Clause]
[GROUP BY clause]
[HAVING clause]
[ORDER BY clause];
```

- *table-name* is the name of the table from which information is retrieved.
- *column_list* includes one or more columns from which data is retrieved.
- The code within the brackets is optional.

database table student_details;

id	first_name	last_name	age	subject	games
100	Rahul	Sharma	10	Science	Cricket
101	Anjali	Bhagwat	12	Maths	Football
102	Stephen	Fleming	09	Science	Cricket
103	Shekar	Gowda	18	Maths	Badminton
104	Priya	Chandra	15	Economics	Chess

NOTE: ***These database tables are used here for better explanation of SQL commands. In reality, the tables can have different columns and different data.***

For example, consider the table student_details. To select the first name of all the students the query would be like:

```
SELECT first_name FROM student_details;
```

NOTE: ***The commands are not case sensitive. The above SELECT statement can also be written as "select first_name from students_details;"***

You can also retrieve data from more than one column. For example, to select first name and last name of all the students.

```
SELECT first_name, last_name FROM student_details;
```

You can also use clauses like WHERE, GROUP BY, HAVING, ORDER BY with SELECT statement. We will discuss these commands in coming chapters.

NOTE: ***In a SQL SELECT statement only SELECT and FROM statements are mandatory. Other clauses like WHERE, ORDER BY, GROUP BY, HAVING are optional.***

How to use expressions in SQL SELECT Statement?

Expressions combine many arithmetic operators, they can be used in SELECT, WHERE and ORDER BY Clauses of the SQL SELECT Statement.

Here we will explain how to use expressions in the SQL SELECT Statement. About using expressions in WHERE and ORDER BY clause, they will be explained in their respective sections.

The operators are evaluated in a specific order of precedence, when more than one arithmetic operator is used in an expression. The order of evaluation is: parentheses, division, multiplication, addition, and subtraction. The evaluation is performed from the left to the right of the expression.

For example: If we want to display the first and last name of an employee combined together, the SQL Select Statement would be like

```
SELECT first_name || ' ' || last_name FROM employee;
```

Output:

```
first_name || ' ' || last_name
--------------------------------
Rahul Sharma
Anjali Bhagwat
Stephen Fleming
Shekar Gowda
Priya Chandra
```

You can also provide aliases as below.

```
SELECT first_name || ' ' || last_name AS emp_name FROM employee;
```

Output:

```
emp_name
------------
Rahul Sharma
Anjali Bhagwat
Stephen Fleming
Shekar Gowda
Priya Chandra
```

SQL Alias

SQL Aliases are defined for columns and tables. Basically aliases is created to make the column selected more readable.

For Example: To select the first name of all the students, the query would be like:

Aliases for columns:

```
SELECT first_name AS Name FROM student_details;
```

```
SELECT first_name      Name FROM student_details;
```

In the above query, the column first_name is given a alias as 'name'. So when the result is displayed the column name appears as 'Name' instead of 'first_name'.

Output:

```
Name
------------
Rahul Sharma
Anjali Bhagwat
Stephen Fleming
Shekar Gowda
Priya Chandra
```

Aliases for tables:

```
SELECT s.first_name FROM student_details s;
```

In the above query, alias 's' is defined for the table student_details and the column first_name is selected from the table.

Aliases is more useful when

- There are more than one tables involved in a query,
- Functions are used in the query,
- The column names are big or not readable,
- More than one columns are combined together

SQL WHERE Clause

The WHERE Clause is used when you want to retrieve specific information from a table excluding other irrelevant data. For example, when you want to see the information about students in class 10th only then you do need the information about the students in other class. Retrieving information about all the students would increase the processing time for the query.

So SQL offers a feature called WHERE clause, which we can use to restrict the data that is retrieved. The condition you provide in the WHERE clause filters the rows retrieved from the table and gives you only those rows which you expected to see. WHERE clause can be used along with SELECT, DELETE, UPDATE statements.

Syntax of SQL WHERE Clause:

```
WHERE {column or expression} comparison-operator value
```

Syntax for a WHERE clause with Select statement is:

```
SELECT column_list FROM table-name

WHERE condition;
```

- *column or expression* - Is the column of a table or a expression
- *comparison-operator* - operators like = <> etc.
- *value* - Any user value or a column name for comparison

For Example: To find the name of a student with id 100, the query would be like:

```
SELECT first_name, last_name FROM student_details

WHERE id = 100;
```

Comparison Operators and Logical Operators are used in WHERE Clause. These operators are discussed in the next chapter.

NOTE: ***Aliases defined for the columns in the SELECT statement cannot be used in the WHERE clause to set conditions. Only aliases created for tables can be used to reference the columns in the table.***

How to use expressions in the WHERE Clause?

Expressions can also be used in the WHERE clause of the SELECT statement.

For example: Lets consider the employee table. If you want to display employee name, current salary, and a 20% increase in the salary for only those products where the percentage increase in salary is greater than 30000, the SELECT statement can be written as shown below

```
SELECT name, salary, salary*1.2 AS new_salary FROM employee
WHERE salary*1.2 > 30000;
```

Output:

name	salary	new_salary
Hrithik	35000	37000
Harsha	35000	37000
Priya	30000	360000

NOTE: ***Aliases defined in the SELECT Statement can be used in WHERE Clause.***

SQL Operators

There are two type of Operators, namely Comparison Operators and Logical Operators. These operators are used mainly in the WHERE clause, HAVING clause to filter the data to be selected.

Comparison Operators:

Comparison operators are used to compare the column data with specific values in a condition.

Comparison Operators are also used along with the SELECT statement to filter data based on specific conditions.

The below table describes each comparison operator.

Comparison Operators	Description
=	equal to
<>, !=	is not equal to
<	less than
>	greater than

> =	greater than or equal to
< =	less than or equal to

Logical Operators:

There are three Logical Operators namely AND, OR and NOT.
Logical operators are discussed in detail in the next section.

SQL Logical Operators

There are three Logical Operators namely, AND, OR, and NOT. These operators compare two conditions at a time to determine whether a row can be selected for the output. When retrieving data using a SELECT statement, you can use logical operators in the WHERE clause, which allows you to combine more than one condition.

Logical Operators	Description
OR	For the row to be selected at least one of the conditions must be true.
AND	For a row to be selected all the specified conditions must be true.
NOT	For a row to be selected the specified condition must be false.

"OR" Logical Operator:

If you want to select rows that satisfy at least one of the given conditions, you can use the logical operator, OR.

For example: if you want to find the names of students who are studying either Maths or Science, the query would be like,

```
SELECT first_name, last_name, subject
FROM student_details
WHERE subject = 'Maths' OR subject = 'Science'
```

The output would be something like,

first_name	last_name	subject
Anajali	Bhagwat	Maths
Shekar	Gowda	Maths
Rahul	Sharma	Science
Stephen	Fleming	Science

The following table describes how logical "OR" operator selects a row.

Column1 Satisfied?	Column2 Satisfied?	Row Selected
YES	YES	YES
YES	NO	YES
NO	YES	YES
NO	NO	NO

"AND" Logical Operator:

If you want to select rows that must satisfy all the given conditions, you can use the logical operator, AND.

For Example: To find the names of the students between the age 10 to 15 years, the query would be like:

```
SELECT first_name, last_name, age
FROM student_details
WHERE age >= 10 AND age <= 15;
```

The output would be something like,

first_name	last_name	age
Rahul	Sharma	10
Anajali	Bhagwat	12
Shekar	Gowda	15

The following table describes how logical "AND" operator selects a row.

Column1 Satisfied?	Column2 Satisfied?	Row Selected
YES	YES	YES
YES	NO	NO
NO	YES	NO
NO	NO	NO

"NOT" Logical Operator:

If you want to find rows that do not satisfy a condition, you can use the logical operator, NOT. NOT results in the reverse of a condition. That is, if a condition is satisfied, then the row is not returned.

For example: If you want to find out the names of the students who do not play football, the query would be like:

```
SELECT first_name, last_name, games

FROM student_details

WHERE NOT games = 'Football'
```

The output would be something like,

first_name	last_name	games
Rahul	Sharma	Cricket
Stephen	Fleming	Cricket
Shekar	Gowda	Badminton
Priya	Chandra	Chess

The following table describes how logical "NOT" operator selects a row.

Column1 Satisfied?	NOT Column1 Satisfied?	Row Selected
YES	NO	NO
NO	YES	YES

Nested Logical Operators:

You can use multiple logical operators in an SQL statement. When you combine the logical operators in a SELECT statement, the order in which the statement is processed is

1) NOT
2) AND
3) OR

For example: If you want to select the names of the students who age is between 10 and 15 years, or those who do not play football, the SELECT statement would be

```
SELECT first_name, last_name, age, games

FROM student_details

WHERE age >= 10 AND age <= 15

OR NOT games = 'Football'
```

The output would be something like,

first_name	last_name	age	games
Rahul	Sharma	10	Cricket
Priya	Chandra	15	Chess

In this case, the filter works as follows:
Condition 1: All the students you do not play football are selected.
Condition 2: All the students whose are aged between 10 and 15 are selected.
Condition 3: Finally the result is, the rows which satisfy at least one of the above conditions is returned.

NOTE: ***The order in which you phrase the condition is important, if the order changes you are likely to get a different result.***

SQL Comparison Keywords

There are other comparison keywords available in sql which are used to enhance the search capabilities of a sql query. They are "IN", "BETWEEN...AND", "IS NULL", "LIKE".

Comparison Operators	Description
LIKE	column value is similar to specified character(s).
IN	column value is equal to any one of a specified set of values.
BETWEEN...AND	column value is between two values, including the end values specified in the range.
IS NULL	column value does not exist.

SQL LIKE Operator

The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose we use a wildcard character '%'.

For example: To select all the students whose name begins with 'S'

```
SELECT first_name, last_name

FROM student_details

WHERE first_name LIKE 'S%';
```

The output would be similar to:

first_name	last_name
Stephen	Fleming
Shekar	Gowda

The above select statement searches for all the rows where the first letter of the column first_name is 'S' and rest of the letters in the name can be any character. There is another wildcard character you can use with LIKE operator. It is the underscore character, ' _ ' . In a search string, the underscore signifies a single character.

For example: to display all the names with 'a' second character,

```
SELECT first_name, last_name

FROM student_details

WHERE first_name LIKE '_a%';
```

The output would be similar to:

first_name	last_name
Rahul	Sharma

NOTE:*Each underscore act as a placeholder for only one character. So you can use more than one underscore. Eg: ' __i% '-this has two underscores towards the left, 'S__j%' - this has two underscores between character 'S' and 'i'.*

SQL BETWEEN ... AND Operator

The operator BETWEEN and AND, are used to compare data for a range of values.
For Example: to find the names of the students between age 10 to 15 years, the query would be like,

```
SELECT first_name, last_name, age

FROM student_details

WHERE age BETWEEN 10 AND 15;
```

The output would be similar to:

first_name	last_name	age
Rahul	Sharma	10

Anajali	Bhagwat	12
Shekar	Gowda	15

SQL IN Operator:

The IN operator is used when you want to compare a column with more than one value. It is similar to an OR condition.

For example: If you want to find the names of students who are studying either Maths or Science, the query would be like,

```
SELECT first_name, last_name, subject
FROM student_details
WHERE subject IN ('Maths', 'Science');
```

The output would be similar to:

first_name	last_name	subject
Anajali	Bhagwat	Maths
Shekar	Gowda	Maths
Rahul	Sharma	Science
Stephen	Fleming	Science

You can include more subjects in the list like ('maths','science','history')

NOTE: ***The data used to compare is case sensitive.***

SQL IS NULL Operator

A column value is NULL if it does not exist. The IS NULL operator is used to display all the rows for columns that do not have a value.

For Example: If you want to find the names of students who do not participate in any games, the query would be as given below

```
SELECT first_name, last_name
FROM student_details
WHERE games IS NULL
```

There would be no output as we have every student participate in a game in the table student_details, else the names of the students who do not participate in any games would be displayed.

SQL ORDER BY

The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order. Oracle sorts query results in ascending order by default.

Syntax for using SQL ORDER BY clause to sort data is:

```
SELECT column-list

FROM table_name [WHERE condition]

[ORDER BY column1 [, column2, .. columnN] [DESC]];
```

database table "employee";

id	name	dept	age	salary	location
100	Ramesh	Electrical	24	25000	Bangalore
101	Hrithik	Electronics	28	35000	Bangalore
102	Harsha	Aeronautics	28	35000	Mysore
103	Soumya	Electronics	22	20000	Bangalore
104	Priya	InfoTech	25	30000	Mangalore

For Example: If you want to sort the employee table by salary of the employee, the sql query would be.

```
SELECT name, salary FROM employee ORDER BY salary;
```

The output would be like

name	salary
Soumya	20000
Ramesh	25000
Priya	30000
Hrithik	35000
Harsha	35000

The query first sorts the result according to name and then displays it.

You can also use more than one column in the ORDER BY clause.
If you want to sort the employee table by the name and salary, the query would be like,

```
SELECT name, salary FROM employee ORDER BY name, salary;
```

The output would be like:

name	salary
Soumya	20000
Ramesh	25000
Priya	30000
Harsha	35000
Hrithik	35000

NOTE: The *columns specified in ORDER BY clause should be one of the columns selected in the SELECT column list.*
You can represent the columns in the ORDER BY clause by specifying the position of a column in the SELECT list, instead of writing the column name.
The above query can also be written as given below,

```
SELECT name, salary FROM employee ORDER BY 1, 2;
```

By default, the ORDER BY Clause sorts data in ascending order. If you want to sort the data in descending order, you must explicitly specify it as shown below.

```
SELECT name, salary

FROM employee

ORDER BY name, salary DESC;
```

The above query sorts only the column 'salary' in descending order and the column 'name' by ascending order.
If you want to select both name and salary in descending order, the query would be as given below.

```
SELECT name, salary

FROM employee

ORDER BY name DESC, salary DESC;
```

How to use expressions in the ORDER BY Clause?

Expressions in the ORDER BY clause of a SELECT statement.

For example: If you want to display employee name, current salary, and a 20% increase in the salary for only those employees for whom the percentage increase in salary is greater than 30000 and in descending order of the increased price, the SELECT statement can be written as shown below

```
SELECT name, salary, salary*1.2 AS new_salary
FROM employee
WHERE salary*1.2 > 30000
ORDER BY new_salary DESC;
```

The output for the above query is as follows.

name	salary	new_salary
Hrithik	35000	37000
Harsha	35000	37000
Priya	30000	36000

NOTE:*Aliases defined in the SELECT Statement can be used in ORDER BY Clause.*

SQL GROUP Functions

Group functions are built-in SQL functions that operate on groups of rows and return one value for the entire group. These functions are: **COUNT, MAX, MIN, AVG, SUM, DISTINCT**

SQL COUNT (): This function returns the number of rows in the table that satisfies the condition specified in the WHERE condition. If the WHERE condition is not specified, then the query returns the total number of rows in the table.

For Example: If you want the number of employees in a particular department, the query would be:

```
SELECT COUNT (*) FROM employee
WHERE dept = 'Electronics';
```

The output would be '2' rows.

If you want the total number of employees in all the department, the query would take the form:

```
SELECT COUNT (*) FROM employee;
```

The output would be '5' rows.

SQL DISTINCT(): This function is used to select the distinct rows.
For Example: If you want to select all distinct department names from employee table, the query would be:

```
SELECT DISTINCT dept FROM employee;
```

To get the count of employees with unique name, the query would be:

```
SELECT COUNT (DISTINCT name) FROM employee;
```

SQL MAX(): This function is used to get the maximum value from a column.
To get the maximum salary drawn by an employee, the query would be:

```
SELECT MAX (salary) FROM employee;
```

SQL MIN(): This function is used to get the minimum value from a column.
To get the minimum salary drawn by an employee, he query would be:

```
SELECT MIN (salary) FROM employee;
```

SQL AVG(): This function is used to get the average value of a numeric column.
To get the average salary, the query would be

```
SELECT AVG (salary) FROM employee;
```

SQL SUM(): This function is used to get the sum of a numeric column
To get the total salary given out to the employees,

```
SELECT SUM (salary) FROM employee;
```

SQL GROUP BY Clause

The SQL GROUP BY Clause is used along with the group functions to retrieve data grouped according to one or more columns.

For Example: If you want to know the total amount of salary spent on each department, the query would be:

```
SELECT dept, SUM (salary)

FROM employee

GROUP BY dept;
```

The output would be like:

dept	salary
Electrical	25000
Electronics	55000
Aeronautics	35000
InfoTech	30000

NOTE: ***The group by clause should contain all the columns in the select list expect those used along with the group functions.***

```
SELECT location, dept, SUM (salary)

FROM employee

GROUP BY location, dept;
```

The output would be like:

location	dept	salary
Bangalore	Electrical	25000
Bangalore	Electronics	55000
Mysore	Aeronautics	35000
Mangalore	InfoTech	30000

SQL HAVING Clause

Having clause is used to filter data based on the group functions. This is similar to WHERE condition but is used with group functions. Group functions cannot be used in WHERE Clause but can be used in HAVING clause.

For Example: If you want to select the department that has total salary paid for its employees more than 25000, the sql query would be like;

```
SELECT dept, SUM (salary)
FROM employee
GROUP BY dept
HAVING SUM (salary) > 25000
```

The output would be like:

dept	salary
Electronics	55000
Aeronautics	35000
InfoTech	30000

When WHERE, GROUP BY and HAVING clauses are used together in a SELECT statement, the WHERE clause is processed first, then the rows that are returned after the WHERE clause is executed are grouped based on the GROUP BY clause. Finally, any conditions on the group functions in the HAVING clause are applied to the grouped rows before the final output is displayed.

SQL INSERT Statement

The INSERT Statement is used to add new rows of data to a table.
We can insert data to a table in two ways,

1) Inserting the data directly to a table.

Syntax for SQL INSERT is:

```
INSERT INTO TABLE_NAME  [ (col1, col2, col3,...colN)]
VALUES (value1, value2, value3,...valueN);
```

- col1, col2,...colN -- the names of the columns in the table into which you want to insert data.

While inserting a row, if you are adding value for all the columns of the table you need not specify the column(s) name in the sql query. But you need to make sure the order of the values is in the same order as the columns in the table. The sql insert query will be as follows

```
INSERT INTO TABLE_NAME
VALUES (value1, value2, value3,... valueN);
```

For Example: If you want to insert a row to the employee table, the query would be like,

```
INSERT INTO employee (id, name, dept, age, salary location)
VALUES (105, 'Srinath', 'Aeronautics', 27, 33000);
```

NOTE:*When adding a row, only the characters or date values should be enclosed with single quotes.*

If you are inserting data to all the columns, the column names can be omitted. The above insert statement can also be written as,

```
INSERT INTO employee
VALUES (105, 'Srinath', 'Aeronautics', 27, 33000);
```

Inserting data to a table through a select statement.

Syntax for SQL INSERT is:

```
INSERT INTO table_name  [(column1, column2, ... columnN)]
SELECT column1, column2, ...columnN
FROM table_name [WHERE condition];
```

For Example: To insert a row into the employee table from a temporary table, the sql insert query would be like,

```
INSERT INTO employee (id, name, dept, age, salary location)
SELECT emp_id, emp_name, dept, age, salary, location
FROM temp_employee;
```

If you are inserting data to all the columns, the above insert statement can also be written as,

```
INSERT INTO employee
SELECT * FROM temp_employee;
```

NOTE:*We have assumed the temp_employee table has columns emp_id, emp_name, dept, age, salary, location in the above given order and the same datatype.*

IMPORTANT NOTE:

1) When adding a new row, you should ensure the data type of the value and the column matches

2) You follow the integrity constraints, if any, defined for the table.

SQL UPDATE Statement

The UPDATE Statement is used to modify the existing rows in a table.

The Syntax for SQL UPDATE Command is:

```
UPDATE table_name

SET column_name1 = value1,  column_name2 = value2, ...

[WHERE condition]
```

- table_name - the table name which has to be updated.
- column_name1, column_name2.. - the columns that gets changed.
- value1, value2... - are the new values.

NOTE:*In the Update statement, WHERE clause identifies the rows that get affected. If you do not include the WHERE clause, column values for all the rows get affected.*

For Example: To update the location of an employee, the sql update query would be like,

```
UPDATE employee

SET location ='Mysore'

WHERE id = 101;
```

To change the salaries of all the employees, the query would be,

```
UPDATE employee

SET salary = salary + (salary * 0.2);
```

SQL Delete Statement

The DELETE Statement is used to delete rows from a table.
The Syntax of a SQL DELETE statement is:

```
DELETE FROM table_name [WHERE condition];
```

- table_name -- the table name which has to be updated.

NOTE: The *WHERE clause in the sql delete command is optional and it identifies the rows in the column that gets deleted. If you do not include the WHERE clause all the rows in the table is deleted, so be careful while writing a DELETE query without WHERE clause.*

For Example: To delete an employee with id 100 from the employee table, the sql delete query would be like,

```
DELETE FROM employee WHERE id = 100;
```

To delete all the rows from the employee table, the query would be like,

```
DELETE FROM employee;
```

SQL TRUNCATE Statement

The SQL TRUNCATE command is used to delete all the rows from the table and free the space containing the table.

Syntax to TRUNCATE a table:

```
TRUNCATE TABLE table_name;
```

For Example: To delete all the rows from employee table, the query would be like,

```
TRUNCATE TABLE employee;
```

Difference between DELETE and TRUNCATE Statements:

DELETE Statement: This command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified. But it does not free the space containing the table.

TRUNCATE statement: This command is used to delete all the rows from the table and free the space containing the table.

SQL DROP Statement:

The SQL DROP command is used to remove an object from the database. If you drop a table, all the rows in the table is deleted and the table structure is removed from the database. Once a table is dropped we cannot get it back, so be careful while using RENAME command. When a table is dropped all the references to the table will not be valid.

Syntax to drop a sql table structure:

```
DROP TABLE table_name;
```

For Example: To drop the table employee, the query would be like

```
DROP TABLE employee;
```

Difference between DROP and TRUNCATE Statement:

If a table is dropped, all the relationships with other tables will no longer be valid, the integrity constraints will be dropped, grant or access privileges on the table will also be dropped, if want use the table again it has to be recreated with the integrity constraints, access privileges and the relationships with other tables should be established again. But, if a table is truncated, the table structure remains the same, therefore any of the above problems will not exist.

SQL CREATE TABLE Statement

The CREATE TABLE Statement is used to create tables to store data. Integrity Constraints like primary key, unique key, foreign key can be defined for the columns while creating the table. The integrity constraints can be defined at column level or table level. The implementation and the syntax of the CREATE Statements differs for different RDBMS.

The Syntax for the CREATE TABLE Statement is:

```
CREATE TABLE table_name (
   column_name1 datatype,
   column_name2 datatype,
   ... column_nameN datatype );
```

- ***table_name*** - is the name of the table.

- ***column_name1, column_name2....*** - is the name of the columns
- ***datatype*** - is the datatype for the column like char, date, number etc.

For Example: If you want to create the employee table, the statement would be like,

```
CREATE TABLE employee (
  id number(5),
  name char(20),
  dept char(10),
  age number(2),
  salary number(10),
  location char(10) );
```

In Oracle database, the datatype for an integer column is represented as "number". In Sybase it is represented as "int".
Oracle provides another way of creating a table.

```
CREATE TABLE temp_employee
SELECT * FROM employee
```

In the above statement, temp_employee table is created with the same number of columns and datatype as employee table.

SQL ALTER TABLE Statement

The SQL ALTER TABLE command is used to modify the definition (structure) of a table by modifying the definition of its columns. The ALTER command is used to perform the following functions.
1) Add, drop, modify table columns
2) Add and drop constraints
3) Enable and Disable constraints

Syntax to add a column

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

For Example: To add a column "experience" to the employee table, the query would be like

```
ALTER TABLE employee ADD experience number(3);
```

Syntax to drop a column

```
ALTER TABLE table_name DROP column_name;
```

For Example: To drop the column "location" from the employee table, the query would be like

```
ALTER TABLE employee DROP location;
```

Syntax to modify a column

```
ALTER TABLE table_name MODIFY column_name datatype;
```

For Example: To modify the column salary in the employee table, the query would be like

```
ALTER TABLE employee MODIFY salary number(15,2);
```

SQL RENAME Command

The SQL RENAME command is used to change the name of the table or a database object.

If you change the object's name any reference to the old name will be affected. You have to manually change the old name to the new name in every reference.

Syntax to rename a table

```
RENAME old_table_name To new_table_name;
```

For Example: To change the name of the table employee to my_employee, the query would be like

```
RENAME employee TO my_emloyee;
```

SQL Integrity Constraints

Integrity Constraints are used to apply business rules for the database tables. The constraints available in SQL are **Foreign Key, Not Null, Unique, Check.** Constraints can be defined in two ways

1) The constraints can be specified immediately after the column definition. This is called column-level definition.
2) The constraints can be specified after all the columns are defined. This is called table-level definition.

1) SQL Primary key:

This constraint defines a column or combination of columns which uniquely identifies each row in the table.

Syntax to define a Primary key at column level:

```
column name datatype [CONSTRAINT constraint_name] PRIMARY KEY
```

Syntax to define a Primary key at table level:

```
[CONSTRAINT constraint_name] PRIMARY KEY

(column_name1,column_name2,..)
```

- **column_name1, column_name2** are the names of the columns which define the primary Key.
- The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.

For Example: To create an employee table with Primary Key constraint, the query would be like.

Primary Key at column level:

```
CREATE TABLE employee (

id number(5) PRIMARY KEY,

name char(20),

dept char(10),

age number(2),

salary number(10),

location char(10) );
```

or

```
CREATE TABLE employee (
id number(5) CONSTRAINT emp_id_pk PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) );
```

Primary Key at table level:

```
CREATE TABLE employee (
id number(5),
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT emp_id_pk PRIMARY KEY (id) );
```

2) SQL Foreign key or Referential Integrity :

This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between different tables. For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring. One or more columns can be defined as Foreign key.

Syntax to define a Foreign key at column level:

```
[CONSTRAINT constraint_name] REFERENCES
Referenced_Table_name(column_name)
```

Syntax to define a Foreign key at table level:

```
[CONSTRAINT constraint_name] FOREIGN KEY(column_name)
REFERENCES referenced_table_name(column_name);
```

For Example:

1) Lets use the "product" table and "order_items".

Foreign Key at column level:

```
CREATE TABLE product
( product_id number(5) CONSTRAINT pd_id_pk PRIMARY KEY,
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
CREATE TABLE order_items
( order_id number(5) CONSTRAINT od_id_pk PRIMARY KEY,
product_id number(5) CONSTRAINT pd_id_fk REFERENCES,
product(product_id),
product_name char(20),
supplier_name char(20),
```

```
unit_price number(10)
);
```

Foreign Key at table level:

```
CREATE TABLE order_items
( order_id number(5) ,
product_id number(5),
product_name char(20),
supplier_name char(20),
unit_price number(10)
CONSTRAINT od_id_pk PRIMARY KEY(order_id),
CONSTRAINT pd_id_fk FOREIGN KEY(product_id) REFERENCES
product(product_id) );
```

2) If the employee table has a 'mgr_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
mgr_id number(5) REFERENCES employee(id),
salary number(10),
```

```
location char(10) );
```

3) SQL Not Null Constraint :

This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.

Syntax to define a Not Null constraint:

```
[CONSTRAINT constraint name] NOT NULL
```

For Example: To create a employee table with Null value, the query would be like

```
CREATE TABLE employee (
id number(5),
name char(20) CONSTRAINT nm_nn NOT NULL,
dept char(10),
age number(2),
salary number(10),
location char(10)
);
```

4) SQL Unique Key:

This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.

Syntax to define a Unique key at column level:

```
[CONSTRAINT constraint_name] UNIQUE
```

Syntax to define a Unique key at table level:

```
[CONSTRAINT constraint_name] UNIQUE(column_name)
```

For Example: To create an employee table with Unique key, the query would be like,

Unique Key at column level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) UNIQUE );
```

or

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) CONSTRAINT loc_un UNIQUE );
```

Unique Key at table level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
```

```
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT loc_un UNIQUE(location) );
```

5) SQL Check Constraint :

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.

Syntax to define a Check constraint:

```
[CONSTRAINT constraint_name] CHECK (condition)
```

For Example: In the employee table to select the gender of a person, the query would be like

Check Constraint at column level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1) CHECK (gender in ('M','F')),
salary number(10),
location char(10) );
```

Check Constraint at table level:

```
CREATE TABLE employee
( id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
gender char(1),
salary number(10),
location char(10),
CONSTRAINT gender_ck CHECK (gender in ('M','F')) );
```

SQL Joins

SQL Joins are used to relate information in different tables. A Join condition is a part of the sql query that retrieves rows from two or more tables. A SQL Join condition is used in the SQL WHERE Clause of select, update, delete statements.

The Syntax for joining two tables is:

```
SELECT col1, col2, col3...
FROM table_name1, table_name2
WHERE table_name1.col2 = table_name2.col1;
```

If a sql join condition is omitted or if it is invalid the join operation will result in a Cartesian product. The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined. For example, if the first table has 20 rows and the second table has 10 rows, the result will be 20 * 10, or 200 rows. This query takes a long time to execute.

Lets use the below two tables to explain the sql join conditions.

database table "product";

product_id	product_name	supplier_name	unit_price
100	Camera	Nikon	300
101	Television	Onida	100
102	Refrigerator	Vediocon	150
103	Ipod	Apple	75
104	Mobile	Nokia	50

database table "order_items";

order_id	product_id	total_units	customer
5100	104	30	Infosys
5101	102	5	Satyam
5102	103	25	Wipro
5103	101	10	TCS

SQL Joins can be classified into Equi join and Non Equi join.

1) SQL Equi joins

It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.

For example: You can get the information about a customer who purchased a product and the quantity of product.

2) SQL Non equi joins

It is a sql join condition which makes use of some comparison operator other than the equal sign like >, <, >=, <=

1) SQL Equi Joins:

An equi-join is further classified into two categories:

a) SQL Inner Join

b) SQL Outer Join

a) SQL Inner Join:

All the rows returned by the sql query satisfy the sql join condition specified.

For example: If you want to display the product information for each order the query will be as given below. Since you are retrieving the data from two tables, you need to identify the common column between these two tables, which is the product_id.

The query for this type of sql joins would be like,

```
SELECT order_id, product_name, unit_price, supplier_name, total_units
FROM product, order_items
WHERE order_items.product_id = product.product_id;
```

The columns must be referenced by the table name in the join condition, because product_id is a column in both the tables and needs a way to be identified. This avoids ambiguity in using the columns in the SQL SELECT statement.
The number of join conditions is (n-1), if there are more than two tables joined in a query where 'n' is the number of tables involved. The rule must be true to avoid Cartesian product.

We can also use aliases to reference the column name, then the above query would be like,

```
SELECT o.order_id, p.product_name, p.unit_price, p.supplier_name,
o.total_units
FROM product p, order_items o
WHERE o.product_id = p.product_id;
```

b) SQL Outer Join:

This sql join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables. The sql outer join operator in Oracle is (+) and is used on one side of the join condition only.

The syntax differs for different RDBMS implementation. Few of them represent the join conditions as "sql left outer join", "sql right outer join".
If you want to display all the product data along with order items data, with null values displayed for order items if a product has no order item, the sql query for outer join would be as shown below:

```
SELECT p.product_id, p.product_name, o.order_id, o.total_units
FROM order_items o, product p
```

```
WHERE o.product_id (+) = p.product_id;
```

The output would be like,

product_id	product_name	order_id	total_units
100	Camera		
101	Television	5103	10
102	Refrigerator	5101	5
103	Ipod	5102	25
104	Mobile	5100	30

NOTE:*If the (+) operator is used in the left side of the join condition it is equivalent to left outer join. If used on the right side of the join condition it is equivalent to right outer join.*

SQL Self Join:

A Self Join is a type of sql join which is used to join a table to itself, particularly when the table has a FOREIGN KEY that references its own PRIMARY KEY. It is necessary to ensure that the join statement defines an alias for both copies of the table to avoid column ambiguity.

The below query is an example of a self join,

```
SELECT a.sales_person_id, a.name, a.manager_id, b.sales_person_id,
b.name
FROM sales_person a, sales_person b
WHERE a.manager_id = b.sales_person_id;
```

2) SQL Non Equi Join:

A Non Equi Join is a SQL Join whose condition is established using all comparison operators except the equal (=) operator. Like >=, <=, <, >

For example: If you want to find the names of students who are not studying either Economics, the sql query would be like, (lets use student_details table defined earlier.)

```
SELECT first_name, last_name, subject
```

```
FROM student_details
```

```
WHERE subject != 'Economics'
```

The output would be something like,

first_name	last_name	subject
Anajali	Bhagwat	Maths
Shekar	Gowda	Maths
Rahul	Sharma	Science
Stephen	Fleming	Science

SQL Views

A VIEW is a virtual table, through which a selective portion of the data from one or more tables can be seen. Views do not contain data of their own. They are used to restrict access to the database or to hide data complexity. A view is stored as a SELECT statement in the database. DML operations on a view like INSERT, UPDATE, DELETE affects the data in the original table upon which the view is based.

The Syntax to create a sql view is

```
CREATE VIEW view_name AS
```

```
SELECT column_list
```

```
FROM table_name [WHERE condition];
```

- ***view_name*** is the name of the VIEW.
- The SELECT statement is used to define the columns and rows that you want to display in the view.

For Example: to create a view on the product table the sql query would be like

```
CREATE VIEW view_product AS
```

```
SELECT product_id, product_name
```

```
FROM product;
```

SQL Subquery

Subquery or Inner query or Nested query is a query in a query. A subquery is usually added in the WHERE Clause of the sql statement. Most of the time, a subquery is used when you know how to search for a value using a SELECT statement, but do not know the exact value.

Subqueries are an alternate way of returning data from multiple tables.
Subqueries can be used with the following sql statements along with the comparison operators like =, <, >, >=, <= etc.

- SELECT
- INSERT
- UPDATE
- DELETE

For Example:

1) Usually, a subquery should return only one record, but sometimes it can also return multiple records when used with operators like IN, NOT IN in the where clause. The query would be like,

```
SELECT first_name, last_name, subject

FROM student_details

WHERE games NOT IN ('Cricket', 'Football');
```

The output would be similar to:

first_name	last_name	subject
Shekar	Gowda	Badminton
Priya	Chandra	Chess

2) Lets consider the student_details table which we have used earlier. If you know the name of the students who are studying science subject, you can get their id's by using this query below,

```
SELECT id, first_name

FROM student_details

WHERE first_name IN ('Rahul', 'Stephen');
```

but, if you do not know their names, then to get their id's you need to write the query in this manner,

```
SELECT id, first_name

FROM student_details

WHERE id IN (SELECT id FROM student_courses

                WHERE subject= 'Science');
```

Output:

id	first_name
100	Rahul
102	Stephen

In the above sql statement, first the inner query is processed first and then the outer query is processed.

3) Subquery can be used with INSERT statement to add rows of data from one or more tables to another table. Lets try to group all the students who study Maths in a table 'maths_group'.

```
INSERT INTO maths_group(id, name)

SELECT id, first_name || ' ' || last_name

FROM student_details WHERE subject= 'Maths'
```

4) A subquery can be used in the SELECT statement as follows. Lets use the product and order_items table defined in the sql_joins section.

```
select p.product_name, p.supplier_name, (select order_id from

order_items where product_id = 101) as order_id

from product p where p.product_id = 101
```

product_name	supplier_name	order_id
Television	Onida	5103

Correlated Subquery

A query is called correlated subquery when both the inner query and the outer query are interdependent. For every row processed by the inner query, the outer query is processed as well. The inner query depends on the outer query before it can be processed.

```
SELECT p.product_name FROM product p

WHERE p.product_id = (SELECT o.product_id FROM order_items o

                      WHERE o.product_id = p.product_id);
```

NOTE:

1) You can nest as many queries you want but it is recommended not to nest more than 16 subqueries in oracle.

2) If a subquery is not dependent on the outer query it is called a non-correlated subquery.

SQL Index

Index in sql is created on existing tables to retrieve the rows quickly.

When there are thousands of records in a table, retrieving information will take a long time. Therefore indexes are created on columns which are accessed frequently, so that the information can be retrieved quickly. Indexes can be created on a single column or a group of columns. When a index is created, it first sorts the data and then it assigns a ROWID for each row.

Syntax to create Index:

```
CREATE INDEX index_name

ON table_name (column_name1, column_name2...);
```

Syntax to create SQL unique Index:

```
CREATE UNIQUE INDEX index_name

ON table_name (column_name1, column_name2...);
```

- ***index_name*** is the name of the INDEX.
- ***table_name*** is the name of the table to which indexed column belongs.
- ***column_name1, column_name2..*** is the list of columns which make up the INDEX.

In Oracle there are two types of SQL index namely, implicit and explicit.

Implicit Indexes:

They are created when a column is explicitly defined with PRIMARY KEY, UNIQUE KEY Constraint.

Explicit Indexes:

They are created using the "create index.. " syntax.

NOTE:

1) Even though sql indexes are created to access the rows in the table quickly, they slow down DML operations like INSERT, UPDATE, DELETE on the table, because the indexes and tables both are updated along when a DML operation is performed. So use indexes only on columns which are used to search the table frequently.

2) It is not required to create indexes on table which have less data.

3) In oracle database you can define up to sixteen (16) columns in an INDEX.

DCL commands are used to enforce database security in a multiple user database environment. Two types of DCL commands are GRANT and REVOTE. Only Database Administrator's or owner's of the database object can provide/remove privileges on a database object.

SQL GRANT Command

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

The Syntax for the GRANT command is:

```
GRANT privilege_name
ON object_name
TO {user_name |PUBLIC |role_name}
[WITH GRANT OPTION];
```

- ***privilege_name*** is the access right or privilege granted to the user. Some of the access rights are ALL, EXECUTE, and SELECT.
- ***object_name*** is the name of an database object like TABLE, VIEW, STORED PROC and SEQUENCE.

- ***user_name*** is the name of the user to whom an access right is being granted.
- ***user_name*** is the name of the user to whom an access right is being granted.
- ***PUBLIC*** is used to grant access rights to all users.
- ***ROLES*** are a set of privileges grouped together.
- ***WITH GRANT OPTION*** - allows a user to grant access rights to other users.

For Example: GRANT SELECT ON employee TO user1;This command grants a SELECT permission on employee table to user1.You should use the WITH GRANT option carefully because for example if you GRANT SELECT privilege on employee table to user1 using the WITH GRANT option, then user1 can GRANT SELECT privilege on employee table to another user, such as user2 etc. Later, if you REVOKE the SELECT privilege on employee from user1, still user2 will have SELECT privilege on employee table.

SQL REVOKE Command:

The REVOKE command removes user access rights or privileges to the database objects.

The Syntax for the REVOKE command is:

```
REVOKE privilege_name

ON object_name

FROM {user_name |PUBLIC |role_name}
```

For Example: REVOKE SELECT ON employee FROM user1;This command will REVOKE a SELECT privilege on employee table from user1.When you REVOKE SELECT privilege on a table from a user, the user will not be able to SELECT data from that table anymore. However, if the user has received SELECT privileges on that table from more than one users, he/she can SELECT from that table until everyone who granted the permission revokes it. You cannot REVOKE privileges if they were not initially granted by you.

Privileges and Roles:

Privileges: Privileges defines the access rights provided to a user on a database object. There are two types of privileges.

1) System privileges - This allows the user to CREATE, ALTER, or DROP database objects.

2) Object privileges - This allows the user to EXECUTE, SELECT, INSERT, UPDATE, or DELETE data from database objects to which the privileges apply.
Few CREATE system privileges are listed below:

System Privileges	Description
CREATE object	allows users to create the specified object in their own schema.
CREATE ANY object	allows users to create the specified object in any schema.

The above rules also apply for ALTER and DROP system privileges.
Few of the object privileges are listed below:

Object Privileges	Description
INSERT	allows users to insert rows into a table.
SELECT	allows users to select data from a database object.
UPDATE	allows user to update data in a table.
EXECUTE	allows user to execute a stored procedure or a function.

Roles: Roles are a collection of privileges or access rights. 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 by oracle.
Some of the privileges granted to the system roles are as given below:

System Role	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

Creating Roles:
The Syntax to create a role is:

```
CREATE ROLE role_name  [IDENTIFIED BY password];
```

For example: To create a role called "developer" with password as "pwd", the code will be as follows

```
CREATE ROLE testing  [IDENTIFIED BY pwd];
```

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, when you GRANT or REVOKE privileges to the role, you definitely have to identify it with the password.
We can GRANT or REVOKE privilege to a role as below.

For example: To grant CREATE TABLE privilege to a user by creating a testing role:
First, create a testing Role

```
CREATE ROLE testing
```

Second, grant a CREATE TABLE privilege to the ROLE testing. You can add more privileges to the ROLE.

```
GRANT CREATE TABLE TO testing;
```

Third, grant the role to a user.

```
GRANT testing TO user1;
```

To revoke a CREATE TABLE privilege from testing ROLE, you can write:

```
REVOKE CREATE TABLE FROM testing;
```

The Syntax to drop a role from the database is as below:

```
DROP ROLE role_name;
```

For example: To drop a role called developer, you can write:

```
DROP ROLE testing;
```

Oracle Built in Functions
There are two types of functions in Oracle.

1) **Single Row Functions:** Single row or Scalar functions return a value for every row that is processed in a query.
2) **Group Functions:** These functions group the rows of data based on the values returned by the query. This is discussed in SQL GROUP Functions. The group functions are used to calculate aggregate values like total or average, which return just one total or one average value after processing a group of rows.

There are four types of single row functions. They are:

1) **Numeric Functions:** These are functions that accept numeric input and return numeric values.
2) **Character or Text Functions:** These are functions that accept character input and can return both character and number values.
3) **Date Functions:** These are functions that take values that are of datatype DATE as input and return values of datatype DATE, except for the MONTHS_BETWEEN function, which returns a number.
4) **Conversion Functions:** These are functions that help us to convert a value in one form to another form. For Example: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE etc.

You can combine more than one function together in an expression. This is known as nesting of functions.

What is a DUAL Table in Oracle?

This is a single row and single column dummy table provided by oracle. This is used to perform mathematical calculations without using a table.

```
Select * from DUAL
```

Output:

```
DUMMY
-------
X
```

```
Select 777 * 888 from Dual
```

Output:

```
777 * 888
---------
689976
```

1) Numeric Functions:

Numeric functions are used to perform operations on numbers. They accept numeric values as input and return numeric values as output. Few of the Numeric functions are:

Function Name	Return Value
ABS (x)	Absolute value of the number '*x*'
CEIL (x)	Integer value that is Greater than or equal to the number '*x*'
FLOOR (x)	Integer value that is Less than or equal to the number '*x*'
TRUNC (x, y)	Truncates value of number '*x*' up to '*y*' decimal places
ROUND (x, y)	Rounded off value of the '*x*' up to the '*y*' decimal places

The following examples explains the usage of the above numeric functions

Function Name	Examples	Return Value
ABS (x)	ABS (1) ABS (-1)	1 -1
CEIL (x)	CEIL (2.83) CEIL (2.49) CEIL (-1.6)	3 3 -1
FLOOR (x)	FLOOR (2.83) FLOOR (2.49) FLOOR (-1.6)	2 2 -2
TRUNC (x, y)	ROUND (125.456, 1) ROUND (125.456, 0) ROUND (124.456, -1)	125.4 125 120
ROUND (x, y)	TRUNC (140.234, 2) TRUNC (-54, 1) TRUNC (5.7) TRUNC (142, -1)	140.23 54 5 140

These functions can be used on database columns.

For Example: Let's consider the product table used in sql joins. We can use ROUND to round off the unit_price to the nearest integer, if any product has prices in fraction.

```
SELECT ROUND (unit_price) FROM product;
```

2) Character or Text Functions:

Character or text functions are used to manipulate text strings. They accept strings or characters as input and can return both character and number values as output.

Few of the character or text functions are as given below:

Function Name	Return Value
LOWER (string)	All the letters in *'string'* is converted to lowercase.
UPPER (string)	All the letters in *'string'* is converted to uppercase.
INITCAP (string)	All the letters in *'string'* is converted to mixed case.
LTRIM (string, trim_text)	All occurrences of *'trim_text'* is removed from the left of *'string'*.
RTRIM (string, trim_text)	All occurrences of *'trim_text'* is removed from the right of *'string'* .

TRIM (trim_text FROM string)	All occurrences of *'trim_text'* from the left and right of *'string'* , *'trim_text'* can also be only one character long .
SUBSTR (string, m, n)	Returns *'n'* number of characters from *'string'* starting from the *'m'* position.
LENGTH (string)	Number of characters in *'string'* in returned.
LPAD (string, n, pad_value)	Returns *'string'* left-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.
RPAD (string, n, pad_value)	Returns *'string'* right-padded with *'pad_value'* . The length of the whole string will be of *'n'* characters.

For Example, we can use the above UPPER() text function with the column value as follows.

```
SELECT UPPER (product_name) FROM product;
```

The following examples explains the usage of the above character or text functions

Function Name	Examples	Return Value
LOWER(string_value)	LOWER('Good Morning')	good morning
UPPER(string_value)	UPPER('Good Morning')	GOOD MORNING
INITCAP(string_value)	INITCAP('GOOD MORNING')	Good Morning
LTRIM(string_value, trim_text)	LTRIM ('Good Morning', 'Good)	Morning
RTRIM (string_value, trim_text)	RTRIM ('Good Morning', ' Morning')	Good
TRIM (trim_text FROM string_value)	TRIM ('o' FROM 'Good Morning')	Gd Mrning
SUBSTR (string_value, m, n)	SUBSTR ('Good Morning', 6, 7)	Morning
LENGTH (string_value)	LENGTH ('Good Morning')	12
LPAD (string_value, n, pad_value)	LPAD ('Good', 6, '*')	**Good
RPAD (string_value, n, pad_value)	RPAD ('Good', 6, '*')	Good**

3) Date Functions:

These are functions that take values that are of datatype DATE as input and return values of datatypes DATE, except for the MONTHS_BETWEEN function, which returns a number as output.

Few date functions are as given below.

Function Name	Return Value
ADD_MONTHS(date,n)	Returns a date value after adding *'n'* months to the date *'x'*.

MONTHS_BETWEEN (x1, x2)	Returns the number of months between dates x1 and x2.
ROUND (x, date_format)	Returns the date *'x'* rounded off to the nearest century, year, month, date, hour, minute, or second as specified by the *'date_format'*.
TRUNC (x, date_format)	Returns the date *'x'* lesser than or equal to the nearest century, year, month, date, hour, minute, or second as specified by the 'date_format'.
NEXT_DAY (x, week_day)	Returns the next date of the *'week_day'* on or after the date *'x'* occurs.
LAST_DAY (x)	It is used to determine the number of days remaining in a month from the date *'x'* specified.
SYSDATE	Returns the systems current date and time.
NEW_TIME (x, zone1, zone2)	Returns the date and time in zone2 if date 'x' represents the time in zone1.

The below table provides the examples for the above functions

Function Name	Examples	Return Value
ADD_MONTHS ( )	ADD_MONTHS ('16-Sep-81', 3)	16-Dec-81
MONTHS_BETWEEN( )	MONTHS_BETWEEN ('16-Sep-81', '16-Dec-81')	3
NEXT_DAY( )	NEXT_DAY ('01-Jun-08', 'Wednesday')	04-JUN-08
LAST_DAY( )	LAST_DAY ('01-Jun-08')	30-Jun-08
NEW_TIME( )	NEW_TIME ('01-Jun-08', 'IST', 'EST')	31-May-08

4) Conversion Functions:

These are functions that help us to convert a value in one form to another form. For Ex: a null value into an actual value, or a value from one datatype to another datatype like NVL, TO_CHAR, TO_NUMBER, TO_DATE.

Few of the conversion functions available in oracle are:

Function Name	Return Value
TO_CHAR (x [,y])	Converts Numeric and Date values to a character string value. It cannot be used for calculations since it is a string value.
TO_DATE (x [, date_format])	Converts a valid Numeric and Character values to a Date value. Date is formatted to the format specified by *'date_format'*.
NVL (x, y)	If *'x'* is NULL, replace it with *'y'*. *'x'* and *'y'* must be of the same datatype.
DECODE (a, b, c, d, e, default_value)	Checks the value of *'a'*, if *a = b*, then returns *'c'*. If *a = d*, then returns *'e'*. Else, returns *default_value*.

The below table provides the examples for the above functions

Function Name	Examples	Return Value
TO_CHAR ()	TO_CHAR (3000, '$9999') TO_CHAR (SYSDATE, 'Day, Month YYYY')	$3000 Monday, June 2008
TO_DATE ()	TO_DATE ('01-Jun-08')	01-Jun-08
NVL ()	NVL (null, 1)	1

SQL Tuning or SQL Optimization

Sql Statements are used to retrieve data from the database. We can get same results by writing different sql queries. But use of the best query is important when performance is considered. So you need to sql query tuning based on the requirement. Here is the list of queries which we use regularly and how these sql queries can be optimized for better performance.

SQL Tuning / SQL Optimization Techniques:

1) The sql query becomes faster if you use the actual columns names in SELECT statement instead of than '*'.

For Example: Write the query as

```
SELECT id, first_name, last_name, age, subject FROM student_details;
```

Instead of:

```
SELECT * FROM student_details;
```

2) HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter. Do not use HAVING clause for any other purposes.

For Example: Write the query as

```
SELECT subject, count(subject)
FROM student_details
WHERE subject != 'Science'
AND subject != 'Maths'
GROUP BY subject;
```

Instead of:

```
SELECT subject, count(subject)
FROM student_details
GROUP BY subject
HAVING subject!= 'Vancouver' AND subject!= 'Toronto';
```

3) Sometimes you may have more than one subqueries in your main query. Try to minimize the number of subquery block in your query.
For Example: Write the query as

```
SELECT name
FROM employee
WHERE (salary, age ) = (SELECT MAX (salary), MAX (age)
                        FROM employee_details)
AND dept = 'Electronics';
```

Instead of:

```
SELECT name
FROM employee
WHERE salary = (SELECT MAX(salary) FROM employee_details)
AND age = (SELECT MAX(age) FROM employee_details)
AND emp_dept = 'Electronics';
```

4) Use operator EXISTS, IN and table joins appropriately in your query.
a) Usually IN has the slowest performance.
b) IN is efficient when most of the filter criteria is in the sub-query.
c) EXISTS is efficient when most of the filter criteria is in the main query.

For Example: Write the query as

```
Select * from product p
where EXISTS (select * from order_items o
        where o.product_id = p.product_id)
```

Instead of:

```
Select * from product p
where product_id IN
    (select product_id from order_items)
```

5) Use EXISTS instead of DISTINCT when using joins which involves tables having one-to-many relationship.

For Example: Write the query as

```
SELECT d.dept_id, d.dept
FROM dept d
WHERE EXISTS ( SELECT 'X' FROM employee e WHERE e.dept = d.dept);
```

Instead of:

```
SELECT DISTINCT d.dept_id, d.dept
FROM dept d, employee e
WHERE e.dept = d.dept;
```

6) Try to use UNION ALL in place of UNION.

For Example: Write the query as

```
SELECT id, first_name FROM student_details_class10
UNION ALL
SELECT id, first_name FROM sports_team;
```

Instead of:

```
SELECT id, first_name FROM student_details_class10
UNION
SELECT id, first_name FROM sports_team;
```

7) Be careful while using conditions in WHERE clause.
For Example: Write the query as

```
SELECT id, first_name, age FROM student_details WHERE age > 10;
```

Instead of:

```
SELECT id, first_name, age FROM student_details WHERE age != 10;
```

Write the query as

```
SELECT id, first_name, age
FROM student_details
WHERE first_name LIKE 'Chan%';
```

Instead of:

```
SELECT id, first_name, age
FROM student_details
WHERE SUBSTR(first_name,1,3) = 'Cha';
```

Write the query as

```
SELECT id, first_name, age
FROM student_details
WHERE first_name LIKE NVL ( :name, '%');
```

Instead of:

```
SELECT id, first_name, age
```

```
FROM student_details
WHERE first_name = NVL ( :name, first_name);
```

Write the query as

```
SELECT product_id, product_name
FROM product
WHERE unit_price BETWEEN MAX(unit_price) and MIN(unit_price)
```

Instead of:

```
SELECT product_id, product_name
FROM product
WHERE unit_price >= MAX(unit_price)
and unit_price <= MIN(unit_price)
```

Write the query as

```
SELECT id, name, salary
FROM employee
WHERE dept = 'Electronics'
AND location = 'Bangalore';
```

Instead of:

```
SELECT id, name, salary
FROM employee
WHERE dept || location= 'ElectronicsBangalore';
```

Use non-column expression on one side of the query because it will be processed earlier.

Write the query as

```
SELECT id, name, salary

FROM employee

WHERE salary < 25000;
```

Instead of:

```
SELECT id, name, salary

FROM employee

WHERE salary + 10000 < 35000;
```

Write the query as

```
SELECT id, first_name, age

FROM student_details

WHERE age > 10;
```

Instead of:

```
SELECT id, first_name, age

FROM student_details

WHERE age NOT = 10;
```

8) Use DECODE to avoid the scanning of same rows or joining the same table repetitively. DECODE can also be made used in place of GROUP BY or ORDER BY clause.

For Example: Write the query as

```
SELECT id FROM employee

WHERE name LIKE 'Ramesh%' and location = 'Bangalore';
```

Instead of:

```
SELECT DECODE(location,'Bangalore',id,NULL) id FROM employee
WHERE name LIKE 'Ramesh%';
```

9) To store large binary objects, first place them in the file system and add the file path in the database.

10) To write queries which provide efficient performance follow the general SQL standard rules.

a) Use single case for all SQL verbs

b) Begin all SQL verbs on a new line

c) Separate all words with a single space

d) Right or left aligning verbs within the initial SQL verb