Webinar on SQL / PL-SQL

Author:

G Sanjeev M.Tech
Software Developer & Corporate Trainer.
Gold Medalist in Robotics at IIT Kharagpur.

SQL is a Standard - BUT....

SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems. SQL statements are used to retrieve and update data in a database. SQL works with database programs like MS Access, DB2, Informix, MS SQL Server, Oracle, Sybase, etc.

Unfortunately, there are many different versions of the SQL language, but to be in compliance with the ANSI standard, they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others).

Note: Most of the SQL database programs also have their own proprietary extensions in addition to the SQL standard!

SQL Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.
Below is an example of a table called "Persons":

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

SQL Queries

With SQL, we can query a database and have a result set returned.
A query like this:

```
SELECT LastName FROM Persons
```

Gives a result set like this:

LastName
Hansen
Svendson
Pettersen

SQL Data Manipulation Language (DML)

SQL (Structured Query Language) is a syntax for executing queries. But the SQL language also includes a syntax to update, insert, and delete records.
These query and update commands together form the Data Manipulation Language (DML) part of SQL:

- **SELECT** - extracts data from a database table
- **UPDATE** - updates data in a database table
- **DELETE** - deletes data from a database table
- **INSERT INTO** - inserts new data into a database table

SQL Data Definition Language (DDL)

The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables.

The most important DDL statements in SQL are:

- **CREATE TABLE** - creates a new database table
- **ALTER TABLE** - alters (changes) a database table
- **DROP TABLE** - deletes a database table
- **CREATE INDEX** - creates an index (search key)
- **DROP INDEX** - deletes an index

SQL The SELECT Statement

The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

Syntax

```
SELECT column_name(s)
FROM table_name
```

To select the columns named "LastName" and "FirstName", use a SELECT statement like this:

SELECT LastName, FirstName FROM Persons

Persons

LastName	**FirstName**	**Address**	**City**
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

výsledok

LastName	**FirstName**
Hansen	Ola
Svendson	Tove
Pettersen	Kari

Select All Columns

To select all columns from the "Persons" table, use a * symbol instead of column names, like this:

SELECT * FROM Persons

LastName	FirstName	Address	City
Hansen	Ola	Timoteivn 10	Sandnes
Svendson	Tove	Borgvn 23	Sandnes
Pettersen	Kari	Storgt 20	Stavanger

The Result Set

The result from a SQL query is stored in a result-set. Most database software systems allow navigation of the result set with programming functions, like: Move-To-First-Record, Get-Record-Content, Move-To-Next-Record, etc.

Programming functions like these are not a part of this tutorial. To learn about accessing data with function calls.

Semicolon after SQL Statements?

Semicolon is the standard way to separate each SQL statement in database systems that allow more than one SQL statement to be executed in the same call to the server.

Some SQL tutorials end each SQL statement with a semicolon. Is this necessary? We are using MS Access and SQL Server 2000 and we do not have to put a semicolon after each SQL statement, but some database programs force you to use it.

The SELECT DISTINCT Statement

The DISTINCT keyword is used to return only distinct (different) values.

The SELECT statement returns information from table columns. But what if we only want to select distinct elements?

With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement:

Syntax

```
SELECT DISTINCT column_name(s)

FROM table_name
```

Using the DISTINCT keyword

To select ALL values from the column named "Company" we use a SELECT statement like this:

SELECT Company FROM Orders

Orders

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

$\longrightarrow$

Company
Sega
W3Schools
Trio
W3Schools

Note that "W3Schools" is listed twice in the result-set.

To select only DIFFERENT values from the column named "Company" we use a SELECT DISTINCT statement like this:

SELECT **DISTINCT** Company FROM Orders

Orders

Company	OrderNumber
Sega	3412
W3Schools	2312
Trio	4678
W3Schools	6798

$\longrightarrow$

Company
Sega
W3Schools
Trio

Select All Columns

The WHERE clause is used to specify a selection criterion.

The WHERE Clause

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

Syntax

```
SELECT column FROM table
WHERE column operator value
```

With the WHERE clause, the following operators can be used:

Operator	**Description**
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern

Note: In some versions of SQL the <> operator may be written as !=

Using the WHERE Clause

To select only the persons living in the city "Sandnes", we add a WHERE clause to the SELECT statement:

```
SELECT * FROM Persons
WHERE City='Sandnes'
```

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980
Pettersen	Kari	Storgt 20	Stavanger	1960

LastName	FirstName	Address	City	Year
Hansen	Ola	Timoteivn 10	Sandnes	1951
Svendson	Tove	Borgvn 23	Sandnes	1978
Svendson	Stale	Kaivn 18	Sandnes	1980

Using Quotes

Note that we have used single quotes around the conditional values in the examples.

SQL uses single quotes around text values (most database systems will also accept double quotes). Numeric values should not be enclosed in quotes.

For text values:

This is correct:

```
SELECT * FROM Persons WHERE FirstName='Tove'
```

This is wrong:

```
SELECT * FROM Persons WHERE FirstName=Tove
```

The LIKE Condition

The LIKE condition is used to specify a search for a pattern in a column.

Syntax

SELECT column FROM table

WHERE column LIKE pattern

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Using LIKE

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Persons
WHERE FirstName LIKE 'O%'
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%a'
```

Using LIKE 2

The following SQL statement will return persons with first names that contain the pattern 'la':

```
SELECT * FROM Persons
WHERE FirstName LIKE '%la%'
```

SQL The INSERT INTO Statement

The INSERT INTO Statement

The INSERT INTO statement is used to insert new rows into a table.

Syntax

INSERT INTO table_name

VALUES (value1, value2,....)

You can also specify the columns for which you want to insert data:

INSERT INTO table_name (column1, column2,...)

VALUES (value1, value2,....)

Insert a New Row

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger

And this SQL statement:

```
INSERT INTO Persons
VALUES ('Hetland', 'Camilla', 'Hagabakka 24', 'Sandnes')
```

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

Insert Data in Specified Columns

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes

And This SQL statement:

```
INSERT INTO Persons (LastName, Address)
VALUES ('Rasmussen', 'Storgt 67')
```

LastName	FirstName	Address	City
Pettersen	Kari	Storgt 20	Stavanger
Hetland	Camilla	Hagabakka 24	Sandnes
Rasmussen		Storgt 67	

SQL The UPDATE Statement

The Update Statement

The UPDATE statement is used to modify the data in a table.

Syntax

```
UPDATE table_name
SET column_name = new_value
WHERE column_name = some_value
```

Update one Column in a Row

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

We want to add a first name to the person with a last name of "Rasmussen":

```
UPDATE Person SET FirstName = 'Nina'
WHERE LastName = 'Rasmussen'
```

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Storgt 67	

Update several Columns in a Row

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen		Storgt 67	

We want to change the address and add the name of the city:

```
UPDATE Person
SET Address = 'Stien 12', City = 'Stavanger'
WHERE LastName = 'Rasmussen'
```

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

SQL The Delete Statement

The Delete Statement

The DELETE statement is used to delete rows in a table.

Syntax

```
DELETE FROM table_name
WHERE column_name = some_value
```

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

Delete a Row

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger
Rasmussen	Nina	Stien 12	Stavanger

"Nina Rasmussen" is going to be deleted:

DELETE FROM Person WHERE LastName = 'Rasmussen'

LastName	FirstName	Address	City
Nilsen	Fred	Kirkegt 56	Stavanger

Delete All Rows

It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:

DELETE FROM table_name

Or

DELETE * FROM table_name

Miscellaneous Commands

- `show databases;`
  - Show all the databases on the server
- `show tables;`
  - Show all the tables of the present database
- `show columns from table EMPLOYEE;`
- `drop table` *`t_name`*`;`
  - Delete the entire table *t_name*
- `drop database` *`db_name`*`;`
  - Delete the entire database *db_name*
- `load data infile` *`f_name`* `into table` *`t_name`*`;`
  - To be discussed with the next homework.

DCL (Data Control Language) Commands

- The rights or permissions assigned to user(s) to use some or all of Oracle objects are known as privileges.

- **Granting Privileges:** It is used to assigning permissions to users. (Only DBA can assign)
- **Syntax:** grant <permissions> ‘*select, insert, delete, update*
- on <object_name>
- to <username>;

- **Eg.** grant insert on emp to user1; ‘*only user1 can insert*
- grant all on emp to public; ‘*assign all permissions to all users.*

- **Revoking Privileges:** It get back permissions from the users.

- **Syntax:**revoke <permission> on <object_name> from <username>;

- **Examples**

  revoke all on emp from user1; ‘*get back all permissions from user1.*

  revoke select on emp from public; ‘*get back select permission from all users.*

TCL

- **(<u>Transaction Control Language</u>):** It controls over transaction processing by specifying the beginning and ending of transactions.
- **Eg.** Commit, Rollback, Rollback to, Save point etc.

TCL Commands

- Oracle treat a transaction as a single entity & incase of successful termination of transaction the changes are made permanent. The commands used with transactions are:

- **COMMIT:** It ends the current transaction by saving database changes & starts a new transaction.
- **Eg.** commit; *'i.e end or start a transaction*

- **ROLLBACK:** It ends the current transaction by discarding database changes & starts a new transaction.
- **Eg.** rollback; *'i.e undo upto commit*

- **SAVEPOINT:** It defines breakpoints or bookmarks for the transaction to allow partial rollbacks.
- **Eg.** savepoint P1;

- **ROLLBACK TO:** Its undo up to given bookmark or breakpoint. **Eg.** rollback to P1;

TCL Commands

- **Term: <u>SAVEPOINT</u>**

- **Definition:**
  In Oracle PL/SQL, **SAVEPOINT** is a TCL (Transaction Control Language) statement that creates a *break point* as a specified location in the current transaction. A transaction can be partially rolled back to any one of the savepoints. If multiple SAVEPOINT locations are set in the transaction, they are identified by their names, which must be unique. The name specification is optional for a single SAVEPOINT in the transaction.

  Note :

  Note that a <u>COMMIT</u> removes all the savepoints which may have been set earlier in the transaction.

Oracle ORDER BY Clause

In Oracle, ORDER BY Clause is used to sort or re-arrange the records in the result set. The ORDER BY clause is only used with SELECT statement.

- **Syntax:**
- **SELECT** expressions
- **FROM** tables
- **WHERE** conditions
- **ORDER BY** expression [**ASC** | **DESC**];

- **Syntax:**

- **SELECT** * **FROM** database;
- **ORDER BY** name;

Syntax:
- **SELECT** * **FROM** student
- **ORDER BY** name **DESC**;

- **Syntax:**

- **SELECT** * **FROM** supplier
- **ORDER BY** last_name;

Syntax:

- **SELECT** * **FROM** supplier
- **ORDER BY** last_name **DESC**;

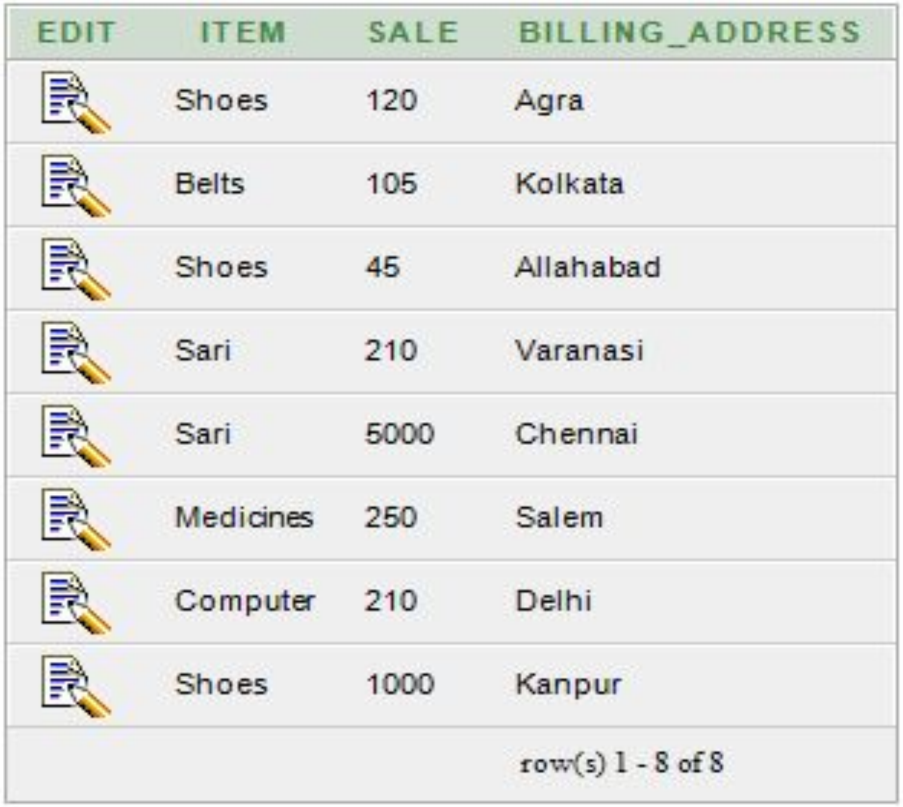
Oracle GROUP BY Clause:

In Oracle GROUP BY clause is used with SELECT

statement to collect data from multiple records

and group the results by one or more columns.

- **CREATE TABLE** "SALESDEPARTMENT"
- ("ITEM" VARCHAR2(4000),
- "SALE" NUMBER,
- "BILLING_ADDRESS" VARCHAR2(4000)
-)

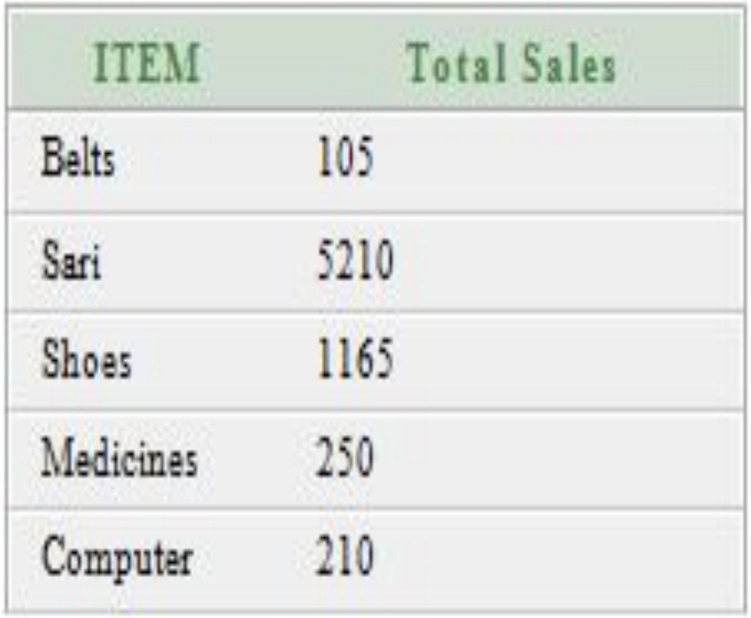


EDIT	ITEM	SALE	BILLING_ADDRESS
	Shoes	120	Agra
	Belts	105	Kolkata
	Shoes	45	Allahabad
	Sari	210	Varanasi
	Sari	5000	Chennai
	Medicines	250	Salem
	Computer	210	Delhi
	Shoes	1000	Kanpur

row(s) 1 - 8 of 8

Oracle GROUP BY Clause:

- **SELECT** item, SUM(sale) **AS** ”Total sales”
- **FROM** salesdepartment
- **GROUP BY** item;



ITEM	Total Sales
Belts	105
Sari	5210
Shoes	1165
Medicines	250
Computer	210

```
CREATE TABLE  CUSTOMERS
 (
 NAME VARCHAR2(4000),  AGE NUMBER,
  SALARY NUMBER, STATE VARCHAR2(4000)
 )
```

Synatax:

```
SELECT state, COUNT(*) AS "Number of customers"

FROM customers

WHERE salary > 10000

GROUP BY state;
```

```
CREATE TABLE  "EMPLOYEES"
   (
    "EMP_ID" NUMBER,
    "NAME" VARCHAR2(4000),
    "AGE" NUMBER,
    "DEPARTMENT" VARCHAR2(4000),
    "SALARY" NUMBER
   )


SELECT department,

MIN(salary) AS "Lowest salary"

FROM employees

GROUP BY department;
```

Oracle HAVING Clause:

In Oracle, HAVING Clause is used with GROUP

BY Clause to restrict the groups of returned rows

where condition is TRUE.

- **Syntax:**

```
SELECT item, SUM(sale) AS "Total sales"

FROM salesdepartment

GROUP BY item

HAVING SUM(sale) < 1000;
```

- **Syntax:**
- **SELECT** department,
- **MAX**(salary) **AS** "Highest salary"
- **FROM** employees
- **GROUP BY** department
- **HAVING MAX**(salary) > 30000;

- **Syntax:**
- **SELECT** state, COUNT(*)
- **AS** "Number of customers"
- **FROM** customers
- **WHERE** salary > 10000
- **GROUP BY** state
- **HAVING** COUNT(*) >= 2;

- **SQL | SUB Queries:**

- In SQL a Subquery can be simply defined as a query within another query. In other words we can say that a Subquery is a query that is embedded in WHERE clause of another SQL query.
- **Important rules for Subqueries:**
- You can place the Subquery in a number of SQL clauses: WHERE You can place the Subquery in a number of SQL clauses: WHERE clause, HAVING clause, FROM clause.

- Subqueries can be used with SELECT, UPDATE, INSERT, DELETE statements along with expression operator. It could be equality operator or comparison operator such as =, >, =, <= and Like operator.
- A subquery is a query within another query. The outer query is called as **main query** and inner query is called as **subquery**.
- The subquery generally executes first, and its output is used to complete the query condition for the main or outer query.
- Subquery must be enclosed in parentheses.
- Subqueries.

- Subqueries are on the right side of the comparison operator.

- ORDER BY command **cannot** be used in a Subquery. GROUP BY command can be used to perform same function as ORDER BY command.

- Use single-row operators with single row Subqueries. Use multiple-row operators with multiple-row

- **Syntax:**
  There is not any general syntax for Subqueries. However, Subqueries are seen to be used most frequently with SELECT statement as shown

- below:

- SELECT column_name FROM table_name WHERE column_name *expression operator* (SELECT COLUMN_NAME from TABLE_NAME WHERE ...);

- **Table name is DATABASE:**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	Chennai	9988775566
Raj	102	Coimbatore	8877665544
Sasi	103	Madurai	7766553344
Ravi	104	Salem	8989898989
Sumathi	105	Kanchipuram	8989856868

- **Table name is Student:**

NAME	ROLL_NO	SECTION
Ravi	104	A
Sumathi	105	B
Raj	102	A

- **Sample Queries**:
- To display NAME, LOCATION, PHONE_NUMBER of the students from DATABASE table whose section is A.
- **Syntax:**
- Select NAME, LOCATION, PHONE_NUMBER from DATABASE WHERE ROLL_NO IN
- (SELECT ROLL_NO from STUDENT where SECTION=’A’);

- **Explanation :**
- First subquery executes “ SELECT ROLL_NO from STUDENT where SECTION=’A’ ” returns ROLL_NO from STUDENT table whose SECTION is ‘A’.Then outer-query executes it and return the NAME, LOCATION, PHONE_NUMBER from the DATABASE table of the student whose ROLL_NO is returned from inner subquery.
- Output:

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ravi	104	Salem	8989898989
Raj	102	Coimbatore	8877665544

- **Insert Query Example:**
- **Table1 : Student1**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989

- **Table2:**
- **Student2**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

Output:

- To insert Student2 into Student1 table:
- INSERT INTO Student1 SELECT * FROM Student2;
- **Output:**

NAME	ROLL_NO	LOCATION	PHONE_NUMBER
Ram	101	chennai	9988773344
Raju	102	coimbatore	9090909090
Ravi	103	salem	8989898989
Raj	111	chennai	8787878787
Sai	112	mumbai	6565656565
Sri	113	coimbatore	7878787878

- To delete students from Student2 table whose
- rollno is same as that in Student1 table and
- having location as chennai

- **Syntax:**

- DELETE FROM
- Student2 WHERE ROLL_NO IN
- (SELECT
- ROLL_NO FROM Student1 WHERE
- LOCATION = ’chennai’);

- To update name of the students to geeks in
- Student2 table whose location is same as
- Raju, Ravi in Student1 table.

- **Syntax:**

UPDATE Student2 SET NAME=’geeks’ WHERE

LOCATION IN (SELECT LOCATION FROM

Student1 WHERE NAME IN (‘Raju’,’Ravi’));

- We have the following two tables
- 'student' and 'marks' with common field 'StudentID'.

- Student:

STUDENT_ID	NAME
S001	Raj
S002	Ravi
s003	Sana

- Marks:

STUDENT_ID	MARKS
S001	95
S002	80
s003	98

- Now we want to write a query to identify all
- students who get better marks than that of the
- student who's StudentID is ‘s002', but we do not
- know the marks of ‘s002'.
- Using the result of this query, to identify the students who get better marks than 80.

- **Oracle UNION Operator:**

In Oracle, UNION operator is used to combine

the result sets of two or more Oracle SELECT

statements. It combines the both SELECT

statement and removes duplicate rows between them.

Each SELECT statement within the UNION operator must

have the same number of fields in the result sets with similar data types.

- **Syntax:**

```
SELECT column name(s)
FROM table name(1)
UNION
SELECT column name(s)
FROM table name(2);

 SELECT supplier_id, supplier_name
FROM suppliers
WHERE supplier_id <= 20
UNION
SELECT s_id, s_name
FROM shopkeepers
WHERE s_name = ’MRF'
ORDER BY 1;
```

Difference between UNION and UNION ALL operators:

UNION operator removes duplicate rows while

UNION ALL operator does not remove duplicate

rows.

- **Syntax:**

SELECT column name(s)
FROM table name(1)
UNION ALL
SELECT column name(s)
FROM table name(2);

Oracle Joins

Join is a query that is used to combine rows from two or more tables, views, or materialized views. It retrieves data from multiple tables and creates a new table.

Join Conditions:

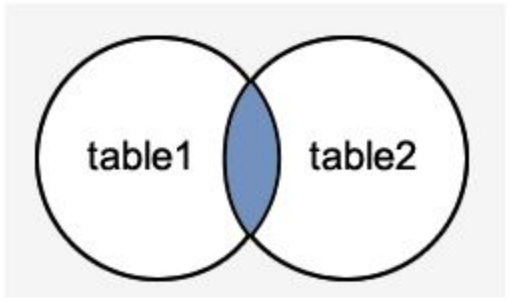
There may be at least one join condition either in the FROM clause or in the WHERE clause for joining two tables. It compares two columns from different tables and combines pair of rows, each containing one row from each table, for which join condition is true.

- **Types of Joins:**

- Inner Joins (Simple Join)
- Outer Joins
  - Left Outer Join (Left Join)
  - Right Outer Join (Right Join)
  - Full Outer Join (Full Join)
- Equijoins
- Self Joins
- Cross Joins (Cartesian Products)
- Antijoins
- Semijoins

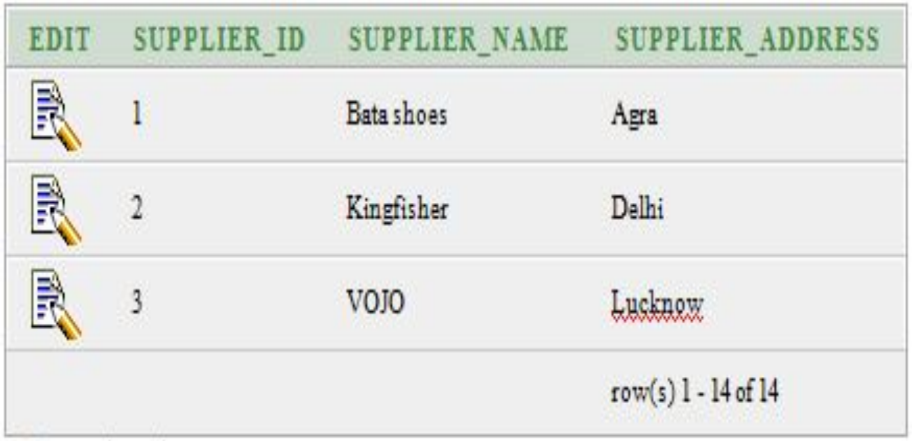
Oracle Joins

- **Oracle INNER JOIN:**
- Inner Join is the simplest and most common type of join. It is also known as simple join. It returns all rows from multiple tables where the join condition is met.



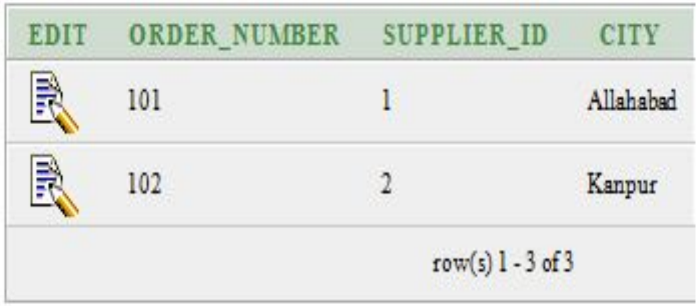
- **Syntax**
- **SELECT** columns
- **FROM** table1
- **INNER** JOIN table2
- **ON** table1.**column** = table2.**column**;

- **Suppliers:**



EDIT	SUPPLIER_ID	SUPPLIER_NAME	SUPPLIER_ADDRESS
	1	Bata shoes	Agra
	2	Kingfisher	Delhi
	3	VOJO	Lucknow
			row(s) 1 - 14 of 14

orders:



EDIT	ORDER_NUMBER	SUPPLIER_ID	CITY
	101	1	Allahabad
	102	2	Kanpur
		row(s) 1 - 3 of 3	

Syntax:

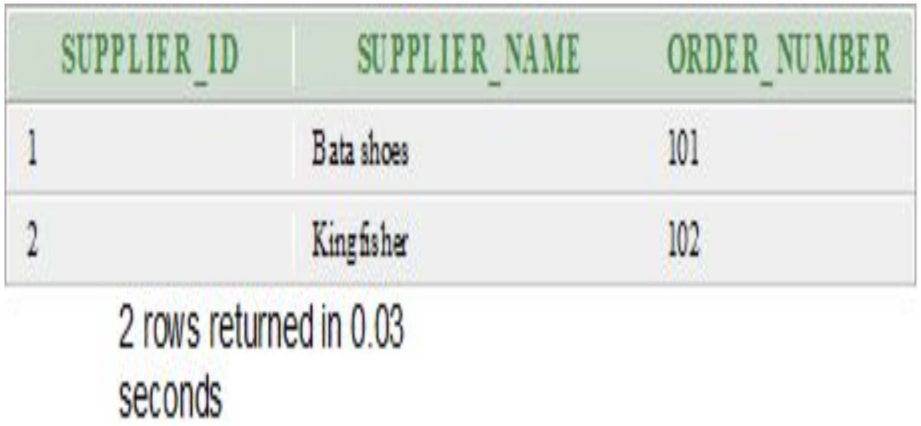
SELECT suppliers.supplier_id, suppliers.supplier_name,

orders.order_number

FROM suppliers

INNER JOIN orders

ON suppliers.supplier_id = orders.supplier_id;

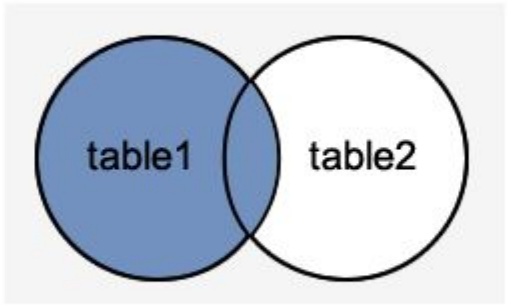


SUPPLIER_ID	SUPPLIER_NAME	ORDER_NUMBER
1	Bata shoes	101
2	Kingfisher	102

2 rows returned in 0.03 seconds

Left Outer Join:

Left Outer Join returns all rows from the left (first) table specified in the ON condition and only those rows from the right (second) table where the join condition is met.

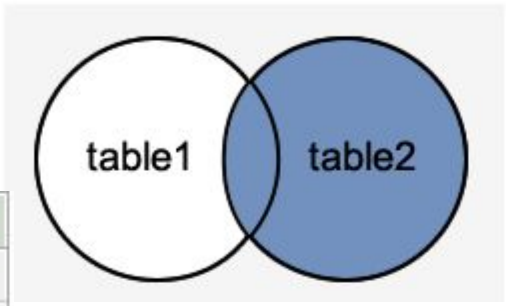


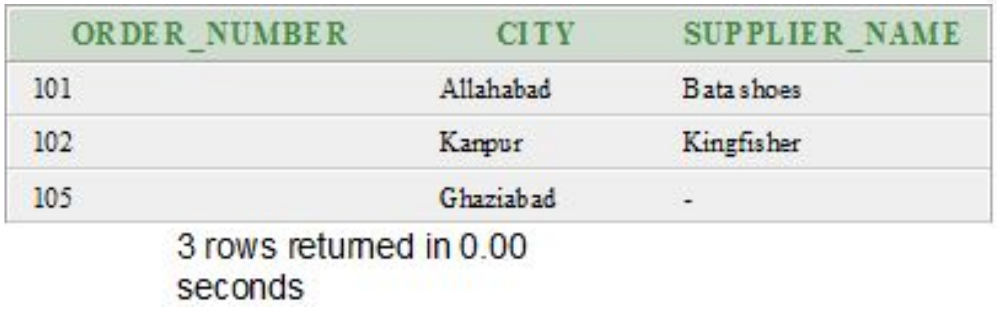
Syntax:

SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_number
FROM suppliers
LEFT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;

Right Outer Join:

The Right Outer Join returns all rows from the right-hand table specified in the ON condition and only those rows from the other table wher condition is met.





ORDER_NUMBER	CITY	SUPPLIER_NAME
101	Allahabad	Bata shoes
102	Kanpur	Kingfisher
105	Ghaziabad	-

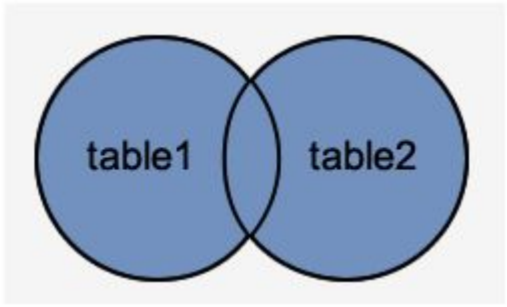
3 rows returned in 0.00 seconds

Syntax:

SELECT orders.order_number, orders.city, suppliers.supplier_name
FROM suppliers
RIGHT OUTER JOIN orders
ON suppliers.supplier_id = orders.supplier_id;

Full Outer Join:

The Full Outer Join returns all rows from the left hand table and right hand table. It places NULL where the join condition is not met.



Syntax:

SELECT suppliers.supplier_id, suppliers.supplier_name, orders.order_number
FROM suppliers
FULL OUTER JOIN order1s
ON suppliers.supplier_id = orders.supplier_id;

Oracle EQUI JOIN:

Oracle Equi join returns the matching column values of the associated tables. It uses a comparison operator in the WHERE clause to refer equality.

Syntax:

SELECT column_list
FROM table1, table2....
WHERE table1.column_name =
table2.column_name;

Oracle SELF JOIN:

Self Join is a specific type of Join. In Self Join, a table is joined with itself (Unary relationship). A self join simply specifies that each rows of a table is combined with itself and every other row of the table.

Syntax:

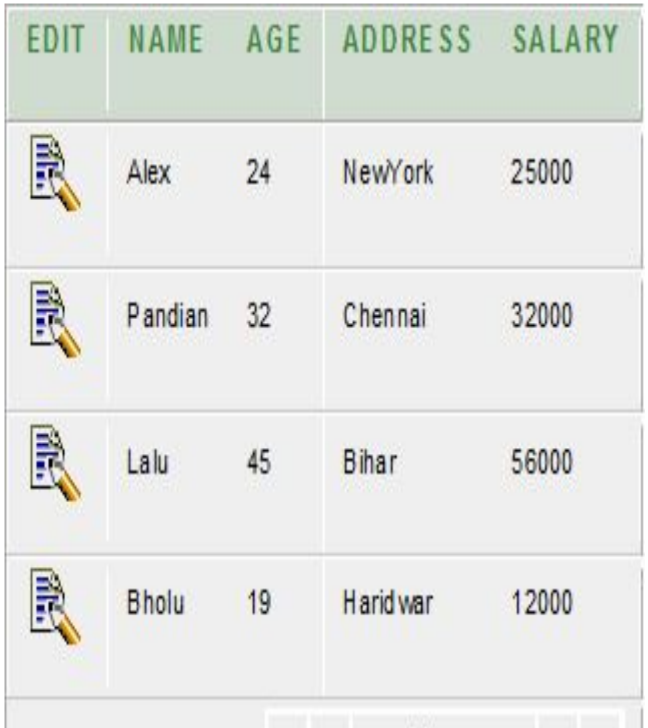
```
SELECT a.column_name, b.column_name...
FROM table1 a, table1 b
WHERE a.common_filed = b.common_field;
```

Oracle SELF JOIN Example:

Let's take a table name "customers".

Syntax:

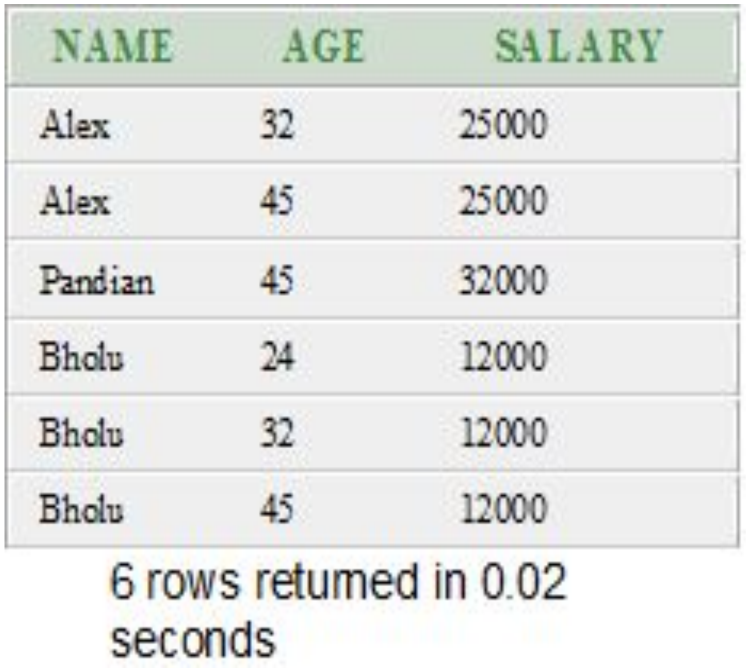
SELECT a.**name**, b.age, a.SALARY
FROM CUSTOMERS a, CUSTOMERS b
WHERE a.SALARY < b.SALARY;



EDIT	NAME	AGE	ADDRESS	SALARY
✎	Alex	24	NewYork	25000
✎	Pandian	32	Chennai	32000
✎	Lalu	45	Bihar	56000
✎	Bholu	19	Haridwar	12000

Oracle SELF JOIN:

Out put:



NAME	AGE	SALARY
Alex	32	25000
Alex	45	25000
Pandian	45	32000
Bholu	24	12000
Bholu	32	12000
Bholu	45	12000

6 rows returned in 0.02 seconds

Oracle Cross Join (Cartesian Products):

The CROSS JOIN specifies that all rows from first table join with all of the rows of second table. If there are "x" rows in table1 and "y" rows in table2 then the cross join result set have **x*y rows**. It normally happens when no matching join columns are specified.

In simple words you can say that if two tables in a join query have no join condition, then the Oracle returns their Cartesian product.

- **Syntax**
- **SELECT** *
- **FROM** table1
- CROSS JOIN table2;
- **Or**
- **SELECT** * **FROM** table1, table2

- **Why do we use SQL constraints? Which constraints we can use while creating database in SQL?**
- Constraints are used to set the rules for all records in the table. If any constraints get violated then it can abort the action that caused it.
- Constraints are defined while creating the database itself with CREATE TABLE statement or even after the table is created once with ALTER TABLE statement.
- ***There are 5 major constraints are used in SQL, such as***
- **NOT NULL:** That indicates that the column must have some value and cannot be left null
- **UNIQUE:** This constraint is used to ensure that each row and column has unique value and no value is being repeated in any other row or column
- **PRIMARY KEY:** This constraint is used in association with NOT NULL and UNIQUE constraints such as on one or the combination of more than one columns to identify the particular record with a unique identity.
- **FOREIGN KEY:** It is used to ensure the referential integrity of data in the table and also matches the value in one table with another using Primary Key
- **CHECK:** It is used to ensure whether the value in columns fulfills the specified condition

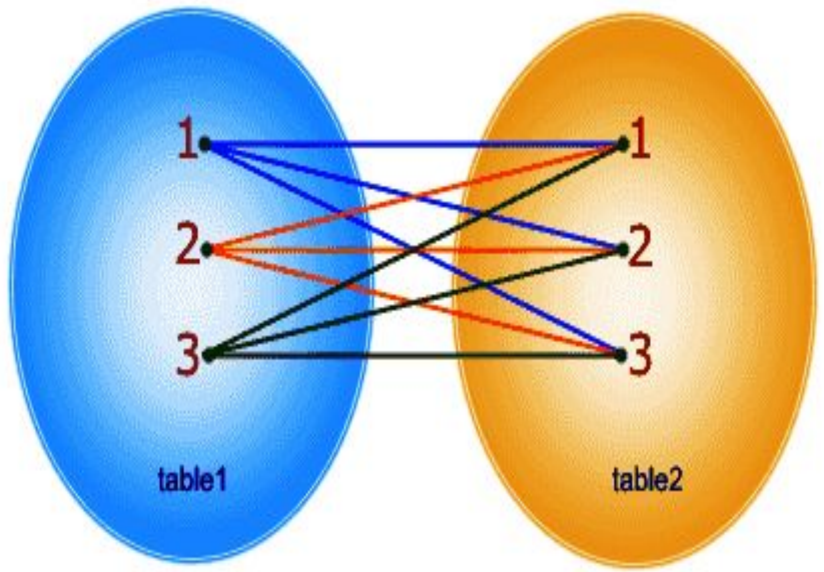
- **NOT NULL Constraint**
- **NOT NULL** constraint restricts a column from having a NULL value. Once **NOT NULL** constraint is applied to a column, you cannot pass a null value to that column. It enforces a column to contain a proper value.
- One important point to note about this constraint is that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

- **Example using NOT NULL constraint**

- CREATE TABLE Student(s_id int NOT NULL, Name varchar(60), Age int);The above query will declare that the **s_id** field of **Student** table will not take NULL value.

Oracle CROSS JOIN:

Syntax: SELECT * FROM customer,supplier



PL/SQL Variables:

A variable is a meaningful name which facilitates a programmer to store data temporarily during the execution of code. It helps you to manipulate data in PL/SQL programs. It is nothing except a name given to a storage area

The variable_name should not exceed 30 characters.

Variable name should not be the same as the table table's column of that block.

```
DECLARE
  a integer := 30;
  b integer := 40;
  c integer;
  f real;
BEGIN
  c := a + b;
  dbms_output.put_line('Value of c: ' || c);
  f := 100.0/2.0;
  dbms_output.put_line('Value of f: ' || f);
END;
```

PL/SQL Variables:

```
DECLARE
 -- Global variables
  num1 number := 95;
  num2 number := 85;
BEGIN
  dbms_output.put_line('Outer Variable num1: ' || num1);
  dbms_output.put_line('Outer Variable num2: ' || num2);
  DECLARE
    -- Local variables
    num1 number := 195;
    num2 number := 185;
  BEGIN
    dbms_output.put_line('Inner Variable num1: ' || num1);
    dbms_output.put_line('Inner Variable num2: ' || num2);
  END;
END;
```

PL/SQL Constants:

A constant is a value used in a PL/SQL block that remains unchanged throughout the program. It is a user-defined literal value.

Syntax to declare a constant:

```
constant_name CONSTANT datatype := VALUE;
```

PL/SQL If:

PL/SQL supports the programming language features like conditional statements and iterative statements.

```
DECLARE
  a number(3) := 500;
BEGIN
  -- check the boolean condition using if statement
  IF( a < 20 ) THEN
    -- if condition is true then print the following
    dbms_output.put_line('a is less than 20 ' );
  ELSE
    dbms_output.put_line('a is not less than 20 ' );
  END IF;
  dbms_output.put_line('value of a is : ' || a);
END;
```

PL/SQL for loop:

PL/SQL for loop is used when when you want to execute a set of statements for a predetermined number of times

```
DECLARE
VAR1 NUMBER;
BEGIN
VAR1:=10;
FOR VAR2 IN 1..10
LOOP
DBMS_OUTPUT.PUT_LINE (VAR1*VAR2);
END LOOP;
END;
```

Oracle Procedures:

A procedure is a group of PL/SQL statements that can be called by name. The call specification (sometimes called call spec) specifies a java method or a third-generation language routine so that it can be called from SQL and PL/SQL.

- **Oracle Create procedure example**
- In this example, we are going to insert record in the "user" table. So you need to create user table first.
- **Table creation:**

1. **create table** user(id number(10) **primary key**,**name** varchar2(100));

- Now write the procedure code to insert record in user table.
- **Procedure Code:**

1. **create** or replace **procedure** "INSERTUSER"
2. (id IN NUMBER,
3. **name** IN VARCHAR2)
4. **is**
5. **begin**
6. **insert into** user **values**(id,**name**);
7. **end**;

Oracle Procedures insert values:

BEGIN

insertuser(101,'Rahul');

END;

DROP PROCEDURE pro1;

Oracle Function:

- A function is a subprogram that is used to return a single value. You must declare and define a function before invoking it. It can be declared and defined at a same time or can be declared first and defined later in the same block.

Oracle Function Example:

- Let's see a simple example to **create a function**.

- **create** or replace **function** adder(n1 in number,
- n2 in number)
- **return** number
- **is**
- n3 number(8);
- **begin**
- n3 :=n1+n2;
- **return** n3;
- **end**;

Oracle Function:

Syntax:

- Now write another program to **call the function**.
- **DECLARE**
-   n3 number(2);
- **BEGIN**
-   n3 := adder(11,22);
-   dbms_output.put_line('Addition is: ' || n3);
- **END**;

Oracle Cursors:

When an SQL statement is processed, Oracle creates a memory area known as context area. A cursor is a pointer to this context area. It contains all information needed for processing the statement.

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
```

Oracle Exceptions:

An error occurs during the program execution is called Exception in PL/SQL.

PL/SQL facilitates programmers to catch such conditions using exception block in the program and an appropriate action is taken against the error condition.

There are two type of exceptions:

System-defined Exceptions
User-defined Exceptions

Oracle Exceptions:

```
DECLARE
   c_id customers.id%type := 5;  // if invalid input exception or ouput
   c_name  customers.name%type;
   c_addr customers.address%type;
BEGIN
   SELECT  name, address INTO  c_name, c_addr
   FROM customers
   WHERE id = c_id;
DBMS_OUTPUT.PUT_LINE ('Name: '||  c_name);
 DBMS_OUTPUT.PUT_LINE ('Address: ' || c_addr);
EXCEPTION
   WHEN no_data_found THEN
      dbms_output.put_line('No such customer!');
   WHEN others THEN
      dbms_output.put_line('Error!');
END;
```

Oracle Triggers:

Trigger is invoked by Oracle engine automatically whenever a specified event occurs.Trigger is stored into database and invoked repeatedly, when specific condition match.

Triggers are written to be executed in response to any of the following events.

A database manipulation (DML) statement (DELETE, INSERT, or UPDATE).

A database definition (DDL) statement (CREATE, ALTER, or DROP).

A database operation (SERVERERROR, LOGON, LOGOFF, STARTUP, or SHUTDOWN).

Oracle Triggers:

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
```

Oracle Triggers:

```
DECLARE
  total_rows number(2);
BEGIN
  UPDATE  customers
  SET salary = salary + 5000;
  IF sql%notfound THEN
    dbms_output.put_line('no customers updated');
  ELSIF sql%found THEN
    total_rows := sql%rowcount;
    dbms_output.put_line( total_rows || ' customers updated ');
  END IF;
END;
```

Table Sample data:

```
create table customers(id int, name varchar2(50), age int, address
varchar2(50), salary int);

insert into customers values(1, 'Ranjan', 34, 'Hyd', 12121);
insert into customers values(2, 'Nitesh', 22, 'Blr', 123123);
insert into customers values(3, 'Shivam', 25, 'Hyd', 82222);
insert into customers values(4, 'Gupta', 23, 'Blr', 7272);
insert into customers values(5, 'Rakesh', 28, 'Chn', 8282);
insert into customers values(6, 'Rahul', 27, 'Pune', 72822);
```