Chapter 8 – SQL Concepts

Prepared By : Prof. Jayesh Chaudhary
Assistant Professor
Computer Engineering Department
SCET

Introduction to SQL

- SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.
- SQL is a keyword based language. Each statement in SQL begins with a unique keyword. These keywords are not case-sensitive.
- Each SQL request is parsed by RDBMS before execution, to check for proper syntax and to optimize request.
- SQL is an ANSI (American National Standards Institute) standard

Introduction to SQL

What Can SQL do?

SQL can

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

Introduction to SQL

The SQL language has several parts:

Data-definition language (DDL).

The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

Command	Description
CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects in the database

Introduction to SQL

Data Manipulation language (DML).

The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.

Command	Description
SELECT	Retrieves certain records from one or more tables.
INSERT	Creates a record
UPDATE	Modifies records
DELETE	Deletes records.

Introduction to SQL

Data Control language (DCL).

Command	Description
Commit	It will end the current transaction making the changes permanent and visible to all users..
Savepoint	It will identify a point(named SAVEPOINT) in a transaction to which you can later roll back
Rollback	It will undo all the changes made by the current transaction.

SQL Data Definition Language

The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:

- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk.

SQL Data types

Data type	Description	Example
CHARACTER(n)	Character string. Fixed-length n	Name character(10)
VARCHAR(n) or CHARACTER VARYING(n)	Character string. Variable length. Maximum length n	Name Varchar(n)
NUMERIC(p,s)	Exact numerical, precision p, scale s.	Price numeric(8,2)
SMALLINT	Integer numerical (no decimal). Precision 5	Roll_No small int(5)
INTEGER	Integer numerical (no decimal). Precision 10	Roll_No integer
BIGINT	Integer numerical (no decimal). Precision 19	Roll_No big int(12)

SQL Data types

Data type	Description	Example
Real , double precision	Approximate numerical, mantissa precision 7 and 16 for double	Price real
Float(n)	Floating point number with atleast n digits precision	Rate float(6,2)
Date	Date containing YYYY-MM-DD	DOB date
Time	hh:mm:ss	Arrival_time time
Time stamp	Combination of date and time	'2002-04-25 08:25:30'

SQL Data Types

What is NULL value ?

- A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.
- NULL value is different than a zero value or a field that contains spaces.

SQL DDL

CREATE TABLE

```
CREATE TABLE Persons
    (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
    );
```

SQL DDL

CREATE TABLE .. AS SELECT

```
CREATE TABLE  Employee
    (PersonID ,LastName,FirstName ,Address,City)
    As  select personId , lastname,firstname, address,city from persons;
    );
```

ALTER Table

The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.

- ALTER TABLE Persons ADD DateOfBirth date;
- ALTER TABLE Persons ADD (DateOfBirth date ,
  Age int);

SQL DDL

ALTER Table …. Modify

The ALTER TABLE … Modify statement is used to change the width or data type of field.

➡ ALTER TABLE BOOK

```
MODIFY( Pub_Year number(4)
);
```

ALTER TABLE … DROP

Delete the column named "DateOfBirth" in the "Persons" table.

```
ALTER TABLE Persons
  DROP DateOfBirth
```

SQL DDL

Example

Employee (PersonID ,LastName,FirstName ,Address,City)

Q.1 Add the Salary Attribute in Employee table

```
ALTER TABLE  EMPLOYEE  ADD SALARY int;
```

Employee (PersonID ,LastName,FirstName ,Address,City , Salary)

Q.2 Modify the Salary Attribute in Employee table with number(8,2)

```
ALTER TABLE  EMPLOYEE  MODIFY (SALARY NUMBER(8,2));
```

Q.3 Delete City Attribute from Employee table

```
ALTER TABLE EMPLOYEE DROP CITY
```

SQL DDL

DROP TABLE

- The DROP TABLE statement is used to delete a table.

```
DROP TABLE table_name
```

- The DROP DATABASE statement is used to delete a database.

```
DROP DATABASE database_name
```

TRUNCATE TABLE

- Delete the data inside the table.

```
TRUNCATE   TABLE   table_name
```

RENAME TABLE

```
RENAME  OldTableName TO  NewTableName;
```

SQL Constraints

- SQL constraints are used to specify rules for the data in a table.
- If there is any violation between the constraint and the data action, the action is aborted by the constraint.
- Constraints can be specified when the table is created
- You can also create constraint After the table is created (inside the ALTER TABLE statement).

Primary Key Constraint

- A primary key is one or more columns in a table used to uniquely identify each row in the table.
- It is combination of not null and unique across the column

SQL Constraints

```
CREATE TABLE supplier
(
        supplier_id numeric(10) not null Primary Key,
        supplier_name varchar2(50) not null,
        contact_name varchar2(50),
 );
```

Primary Key With Constraint Keyword

```
CREATE TABLE supplier
(
 supplier_id numeric(10) not null,
 supplier_name varchar2(50) not null,
 contact_name varchar2(50),
 CONSTRAINT supplier_pk PRIMARY KEY (supplier_id)
 );
```

SQL Constraints

To create a PRIMARY KEY constraint on the "P_Id" column when the table is already created.

```
ALTER TABLE Persons   ADD PRIMARY KEY (P_Id)
```

OR

```
ALTER TABLE Persons
ADD CONSTRAINT pk_PersonID PRIMARY KEY (P_Id)
```

To DROP a PRIMARY KEY Constraint

```
ALTER TABLE Persons DROP CONSTRAINT pk_PersonID
```

SQL Constraints

For Composite primary key

- If Combination of more than one field is used as primary key then it can be referred as composite primary key.

```
CREATE TABLE Persons
(
    P_Id int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    CONSTRAINT pk_PersonID PRIMARY KEY (P_Id,LastName)
)
```

SQL Constraints

Foreign Key Constraint

- Foreign key represent relationship between tables.
- The existence of foreign key implies that the table with foreign key is related to the primary key table from which the foreign key is derived.
- Rejects insert or update of a value if corresponding value does not exist in primary key table.
- Rejects Delete if it would invalidate a Reference constraint.
- Must reference a table not view.

SQL Constraints

```
CREATE TABLE Orders
    (
        O_Id int NOT NULL PRIMARY KEY,
        OrderNo int NOT NULL,
        P_Id int  FOREIGN KEY REFERENCES Persons(P_Id)
    );
```

OR

```
CREATE TABLE Orders
    (
        O_Id int NOT NULL,
        OrderNo int NOT NULL,
        P_Id int,
        PRIMARY KEY (O_Id),
        CONSTRAINT fk_PerOrders FOREIGN KEY (P_Id)
        REFERENCES Persons(P_Id)
    );
```

SQL Constraints

Example for multiple foreign key

Create table *teaches*

(*ID* ***varchar (5),***

course id ***varchar (8),***

sec id ***varchar (8),***

semester ***varchar (6),***

year ***numeric (4,0),***

primary key (*ID, course id, sec id, semester, year),*

foreign key (*course id, sec id, semester, year) references

section,

foreign key (*ID) references instructor

);

SQL Constraints

Unique Key Constraint

- Unique key is used to ensure that the information in the column for each record is unique as with Aadhaar Card Number.

NOT NULL Constraint

- Value must be entered in field.
- Null value is appropriate when the actual value is unknown or value is not meaningful.
- null * 10 = null

SQL Constraints

NOT NULL , Unique Example

```
CREATE TABLE Persons
    (
        P_Id int NOT NULL UNIQUE,
        LastName varchar(255) NOT NULL,
        FirstName varchar(255),
        Address varchar(255),
        City varchar(255)
    )
```

SQL Constraints

Check Integrity Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.
- If you define a CHECK constraint on a single column it allows only certain values for this column.

```
CREATE TABLE Persons
(
    P_Id int NOT NULL CHECK (P_Id>0),
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
)
```

SQL Constraints

Check Integrity Constraint

- At Table level

```
CREATE TABLE Persons
(
    P_Id int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255),
    CONSTRAINT chk_Person CHECK (P_Id>0 AND
    City='Surat')
)
```

DML Statements

Select Clause

- The SELECT statement is used to select data from a database.
- The result is stored in a result table, called the result-set.

```
SELECT * FROM Customers;
```

Distinct

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

```
SELECT DISTINCT City FROM Customers;
```

Where Clause

```
SELECT * FROM Customers
        WHERE Country='Mexico';
```

DML Statements

From Clause

- It Specified a cartesian product of the relation in the clause.

Book (Title , Pub_year , Unit_Price , Author_Name, Pub_Name)

Author(Author_Name , Country)

Q. Find the titles of books with author name and country published in year 2004

Select title, book.authorname,country from book,author

Where book.author_name=author.author_name and pub_year ='2004'

DML Statements

INSERT Into

- The INSERT INTO statement is used to insert new records in a table.

```
INSERT INTO Customers (CustomerName, Address, City, PinCode)
VALUES ('Cardinal', 'Skagen21', 'Stavanger' ,395001);
```

INSERT Into... Select

You can also create SQL INSERT statements using SELECT statement.

```
INSERT INTO suppliers   (supplier_id, supplier_name)
SELECT account_no, name
FROM customers    WHERE city = 'Newark';
```

DML Statements

INSERT Into Statement for Checking Conflict

```
INSERT INTO clients
(client_id, client_name, client_type)
SELECT 10345, 'IBM', 'advertising'
FROM dual
WHERE NOT EXISTS (SELECT *
        FROM clients
        WHERE clients.client_id = 10345);
```

SQL Operators

Logical Operator

It is used to produce a single result from combining the two separate conditions

AND

```
SELECT * FROM Customers WHERE Country='Germany'
AND City='Berlin';
```

OR

```
SELECT * FROM Customers WHERE City='Berlin‘
                        OR City='München';
```

```
SELECT * FROM Customers  WHERE Country='Germany'
AND (City='Berlin' OR City='München');
```

SQL Operators

- **IN** Equal to any member of set

Example

```
SELECT * FROM Customers WHERE City IN ('Paris','London');
```

- **Not In - Not** Equal to member of set

Example

```
SELECT * FROM Customers WHERE City Not IN ('Paris','London');
```

- **Is NULL -** test for nulls

Example

```
SELECT LastName,FirstName,Address FROM Persons
WHERE Address IS NULL
```

SQL Operators

- **Is NOT NULL** - test for anything other than null

Example

```
SELECT LastName,FirstName,Address FROM Persons
WHERE Address IS NOT NULL
```

- **LIKE** - The LIKE operator is used to search for a specified pattern in a column .
  - ➡ wildcard characters are used with the SQL LIKE operator.
  - ➡ SQL wildcards are used to search for data within a table.
  - ➡ % - A substitute for zero or more characters
  - ➡ _ - A substitute for a single character

SQL Operators

- **Example**

  SELECT * FROM Customers WHERE City LIKE 's%';

  SELECT * FROM Customers WHERE City LIKE '%s‘

  SELECT * FROM Customers WHERE Country LIKE '%land%';

  SELECT * FROM Customers WHERE Country NOT LIKE '%land%';

  SELECT * FROM Customers WHERE City LIKE '_erlin';

- **Between**

  SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;

SQL Operators

INTERSECT : Return Common rows selected by both queries

```
SELECT City FROM Customers
INTERSECT
SELECT City FROM Suppliers
ORDER BY City;
```

MINUS : Return all distinct rows from first query but not in second

```
SELECT City FROM Customers
MINUS
SELECT City FROM Suppliers
ORDER BY City;
```

SQL Operators

Set Operator

UNION

- The UNION operator is used to combine the result-set of two or more SELECT statements.
- Each SELECT statement within the UNION must have the same number of columns.
- The columns must also have similar data types.
- The columns in each SELECT statement must be in the same order.
- The UNION operator selects only distinct values by default.

SQL Operators

UNION

```
SELECT City FROM Customers
UNION
SELECT City FROM Suppliers
ORDER BY City;
```

UNION ALL

UNION ALL to select **all** (duplicate values also)

```
SELECT City FROM Customers
UNION ALL
SELECT City FROM Suppliers
ORDER BY City;
```

DML Statements

Update

- The UPDATE statement is used to update existing records in a table.

```
UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerName='Alfreds Futterkiste';
```

Update table with data from another table

```
UPDATE Customers
    SET ContactName=(Select SupplierName from Supplier where
suppliername='abc')
        WHERE CustomerName='Alfreds Futterkiste';
```

DML Statements

Delete

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

```
DELETE FROM Customers
WHERE CustomerName='Alfreds Futterkiste' AND
ContactName='Maria Anders';
```

DML Statements

Order By

The ORDER BY clause is used to sort the records in the result set for a SELECT statement.

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city;
```

Order By with ASC/DESC Attribute

```
SELECT supplier_city
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC;
```

DML Statements

Order By with ASC/DESC Attribute

```
SELECT supplier_city,supplier_state
FROM suppliers
WHERE supplier_name = 'IBM'
ORDER BY supplier_city DESC , supplier_state ASC;
```

DML Statements

Tuple Variables

Tuple Variables are defined in the from clause by the way of as clause.

```
Select title , B.author_name , country
From book B , author A
Where B.author_name=A.author_name
```

DML Statements

Aggregate Functions

Aggregate Functions are functions that take a collection of values as input and return a single value.

- Average : avg()
- Minimum : min()
- Maximum : max()
- Total : sum()
- Count : count()

DML Statements

Aggregate Functions - Average : avg

The AVG() function returns the average value of a numeric column , ignoring null values.

```
SELECT AVG(Price) FROM Products
```

Aggregate Functions - Minimum : min

The min() function returns the minimum value of a expression.

```
SELECT min(Price) FROM Products
```

Aggregate Functions - Maximum : max

The max() function returns the maximum value of a expression.

```
SELECT max(Price) FROM Products
```

DML Statements

Aggregate Functions - Total : sum

The sum() function returns the sum of values of n.

```
SELECT sum(amount) as amount FROM Products
```

Aggregate Functions - count : count

The count() function returns the number of rows where expression is not null .

```
SELECT count(product_name) FROM Products
```

DML Statements

Group By

– The GROUP BY statement is used in conjunction with the aggregate functions to group the result-set by one or more columns.

```
SELECT sum(Quantity) as Qty
FROM Products
Group by productname
        OR
Select ClassName,Count(*)
from student
Group by classname
```

DML Statements

Having

- The having clause tells SQL to include only certain groups produced by the group by clause in the query result set.
- Having is equivalent to where clause and is used to specify the search criteria or search condition when group by clause is specified.

```
SELECT sum(Quantity) as Qty
FROM Products
Group by productname
having productname<>'AC'
```

DML Statements

Having

```
Select ClassName,Count(eNROLLMENTnO)

From student

Group by classname

Having classname<>'2COE'
```

DML Statements

SELECT first record from table

```
Select * from Suppliers

Where rownum<=1

Order by rownum;
```

SELECT lastrecord from table

```
Select * from Suppliers

Where rownum<=1

Order by rownum Desc;
```

DML Statements - Subquery

Subquery is a query within a query. You can create subqueries within your SQL statements.

These subqueries can reside in the WHERE clause, the FROM clause, or the SELECT clause.

These subqueries are also called nested subqueries.

Where Clause

```
Select  EnrollmentNo,Classcode  from student
where  classcode IN (select classcode from class where
                        branchcode='co')
```

DML Statements - Subquery

From Clause

A subquery can also be found in the FROM clause. These are called inline views.

```
SELECT suppliers.name, subquery1.total_amt
FROM suppliers,
 (SELECT supplier_id, SUM(orders.amount) AS total_amt
  FROM orders
  GROUP BY supplier_id) subquery1
WHERE subquery1.supplier_id = suppliers.supplier_id;
```

DML Statements - Subquery

Correlated Queries

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.

Not preferable to use Correlated query.Performance can be degraded.

DML Statements - Subquery

Correlated Queries – Example

Find the list of employees (EmpNum , Name(having more salary than the average salary of all employees in that employee's dept.))

```
SELECT empnum,name
FROM employee as e1
Where salary > ( SELECT avg(salary)
                 FROM employee
                 Where department=e1.department);
```

DML Statements - Subquery

EXIST

The Oracle EXISTS condition is used in combination with a subquery and is considered "to be met" if the subquery returns at least one row.

```
SELECT *    FROM customers
WHERE EXISTS (
    SELECT *
    FROM order_details
    WHERE customers.customer_id = order_details.customer_id);
```

DML Statements - Subquery

EXIST With INSERT

```
INSERT INTO contacts (contact_id, contact_name)
SELECT supplier_id, supplier_name
FROM suppliers WHERE EXISTS
    (SELECT *
    FROM order_details
    WHERE suppliers.supplier_id = order_details.supplier_id);
```

DML Statements - Subquery

EXIST With DELETE

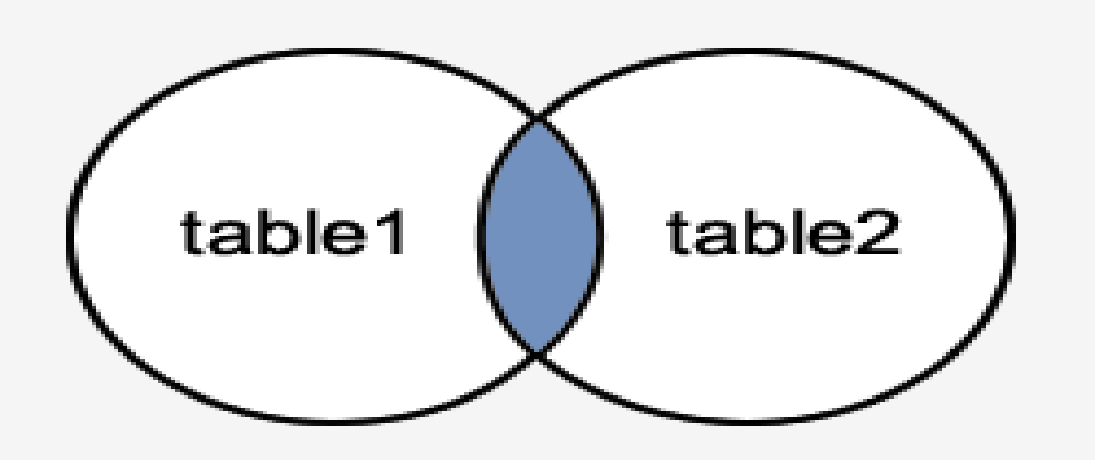
```
DELETE FROM suppliers
WHERE EXISTS     (
    SELECT *
    FROM order_details
    WHERE suppliers.supplier_id = order_details.supplier_id
                    );
```

Joins

- Join is a query in which data is retrieved from two or more table
- A join matches data from two or more tables , based on the values of one or more columns in each table.

Inner Join

- Inner Join returns the matching rows from the tables that are being joined



Joins

Example

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
INNER JOIN orders ON suppliers.supplier_id = orders.supplier_id;
```

DML Statements - Joins

Supplier

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

Order

order_id	supplier_id	order_date
500125	10000	2003/05/12
500126	10001	2003/05/13
500127	10004	2003/05/14

DML Statements - Joins

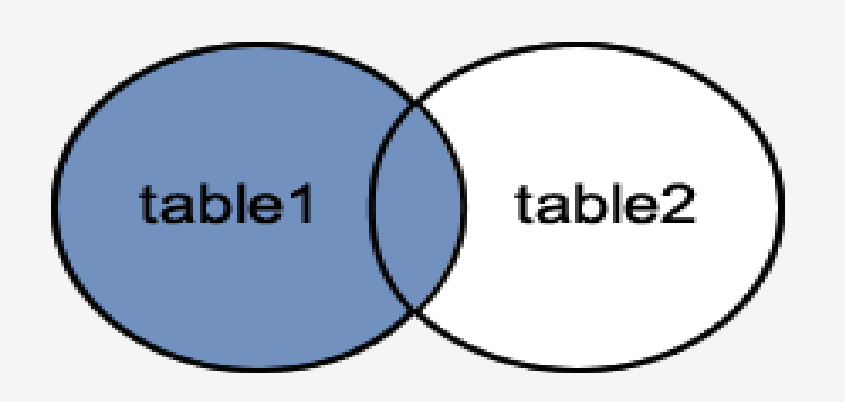
Result of inner Join of Supplier and Order

supplier_id	name	order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13

Joins

LEFT OUTER Join

- **LEFT OUTER** join returns all rows from the LEFT-hand table specified in the ON condition and only those rows from the other table where the joined fields are equal (join condition is met).



- LEFT OUTER JOIN would return the all records from table1 and only those records from table2 that intersect with table1.

DML Statements - Joins

LEFT OUTER Join

Example

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers
LEFT OUTER JOIN
Orders  ON suppliers.supplier_id = orders.supplier_id;
```

DML Statements - Joins

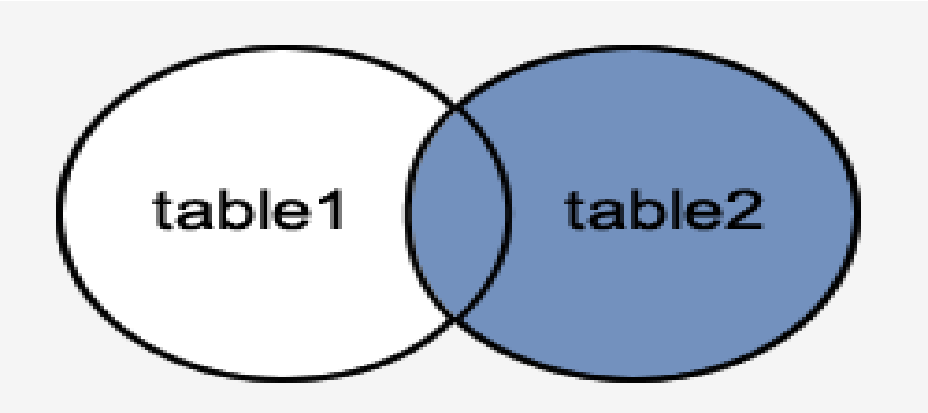
Result of Left Outer Join of Supplier and Order

supplier_id	supplier_name	order_date
10000	IBM	2003/05/12
10001	Hewlett Packard	2003/05/13
10002	Microsoft	<null>
10003	NVIDIA	<null>

DML Statements - Joins

RIGHT OUTER Join

- **RIGHT OUTER** join returns all rows from the RIGHT-hand table specified in the ON condition and **only** those rows from the other table where the joined fields are equal



- The Oracle RIGHT OUTER JOIN would return the all records from *table2* and only those records from *table1* that intersect with *table2*.

DML Statements - Joins

Right OUTER Join

Example

```
SELECT orders.order_id, orders.order_date ,
suppliers.supplier_name
FROM suppliers
RIGHT OUTER JOIN orders ON suppliers.supplier_id =
orders.supplier_id;
```

DML Statements - Joins

Example

Supplier

supplier_id	supplier_name
10000	Apple
10001	Google

Order

order_id	supplier_id	order_date
500125	10000	2013/08/12
500126	10001	2013/08/13
500127	10002	2013/08/14

DML Statements - Joins

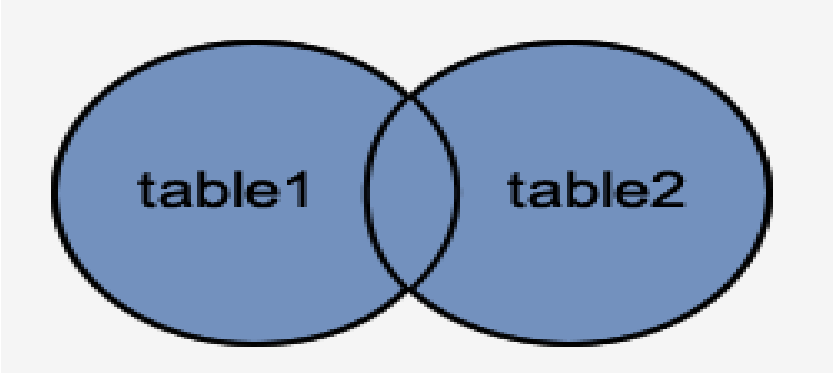
Result of Right Outer Join of Supplier and Order

order_id	order_date	supplier_name
500125	2013/08/12	Apple
500126	2013/08/13	Google
500127	2013/08/14	<null>

DML Statements - Joins

FULL OUTER Join

- **FULL OUTER** join returns all rows from the LEFT-hand table and RIGHT-hand table with nulls in place where the join condition is not met.



- FULL OUTER JOIN would return the all records from both *table1* and *table2*.

DML Statements - Joins

FULL OUTER Join

```
SELECT suppliers.supplier_id, suppliers.supplier_name,
orders.order_date
FROM suppliers FULL OUTER JOIN
orders ON suppliers.supplier_id = orders.supplier_id;
```

DML Statements - Joins

Supplier

supplier_id	supplier_name
10000	IBM
10001	Hewlett Packard
10002	Microsoft
10003	NVIDIA

Order

order_id	supplier_id	order_date
500125	10000	2003/05/12
500126	10001	2003/05/13
500127	10004	2003/05/14

DML Statements - Joins

Result of FULL Outer JOIN

supplier_id	supplier_name	order_date
10000	IBM	2013/08/12
10001	Hewlett Packard	2013/08/13
10002	Microsoft	<null>
10003	NVIDIA	<null>
<null>	<null>	2013/08/14

DML Statements - Joins

Natural JOIN

A natural join is a join statement that compares the common columns of both tables with each other.

- The associated tables have one or more pairs of identically named columns.
- The columns must be the same data type.
- Don’t use ON clause in a natural join

```
SELECT * FROM table1 NATURAL JOIN table2;
```

DML Statements - Joins

Example

Table Name : Foods

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	

DML Statements - Joins

Example

Table Name : Company

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

```
SELECT *
From Foods
Natural Join Company
```

DML Statements - Joins

Example

Output

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

DML Statements - Joins

Pictorial Representation for Natural JOIN

ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_ID
1	Chex Mix	Pcs	16
6	Cheez-It	Pcs	15
2	BN Biscuit	Pcs	15
3	Mighty Munch	Pcs	17
4	Pot Rice	Pcs	15
5	Jaffa Cakes	Pcs	18
7	Salt n Shake	Pcs	-

COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas Foods	Delhi
17	Foodies.	London
19	sip-n-Bite.	New York

** Same column came once

COMPANY_ID	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMPANY_NAME	COMPANY_CITY
16	1	Chex Mix	Pcs	Akas Foods	Delhi
15	6	Cheez-It	Pcs	Jack Hill Ltd	London
15	2	BN Biscuit	Pcs	Jack Hill Ltd	London
17	3	Mighty Munch	Pcs	Foodies.	London
15	4	Pot Rice	Pcs	Jack Hill Ltd	London
18	5	Jaffa Cakes	Pcs	Order All	Boston

DML Statements – Joins -- Inner Join

Difference Between natural join and inner join

- There is one significant difference between INNER JOIN and NATURAL JOIN is the number of columns returned.

```
SELECT *
FROM company
INNER JOIN foods
ON company.company_id = foods.company_id;
```

DML Statements – Joins -- Inner Join

COMPANY_ID	COMPANY_NAME	COMPANY_CITY	ITEM_ID	ITEM_NAME	ITEM_UNIT	COMP
16	Akas Foods	Delhi	1	Chex Mix	Pcs	16
15	Jack Hill Ltd	London	6	Cheez-It	Pcs	15
15	Jack Hill Ltd	London	2	BN Biscuit	Pcs	15
17	Foodies.	London	3	Mighty Munch	Pcs	17
15	Jack Hill Ltd	London	4	Pot Rice	Pcs	15
18	Order All	Boston	5	Jaffa Cakes	Pcs	18

DML Statements - Joins

Difference Between natural join and inner join

- A **Natural Join** is where 2 tables are joined on the basis of all common columns.
- A **Inner Join** is where 2 tables are joined on the basis of common columns mentioned in the ON clause. ...
- **Inner join**, **join** two table where column name is same.
- **Natural join**, **join** two table where column name and data types are same.

DML Statements – Joins - Natural Join

Difference Between natural join and inner join

- There is one significant difference between INNER JOIN and NATURAL JOIN is the number of columns returned.

```
SELECT *
FROM company
NATURAL JOIN foods
```

DML Statements – Joins – natural join

COMPANY_ID	COMPANY_NAME	COMPANY_CITY	ITEM_ID	ITEM_NAME	ITEM_UNIT
16	Akas Foods	Delhi	1	Chex Mix	Pcs
15	Jack Hill Ltd	London	6	Cheez-It	Pcs
15	Jack Hill Ltd	London	2	BN Biscuit	Pcs
17	Foodies.	London	3	Mighty Munch	Pcs
15	Jack Hill Ltd	London	4	Pot Rice	Pcs
18	Order All	Boston	5	Jaffa Cakes	Pcs

DML Statements - Joins

Equi Join

- The Join condition specified determines what type of join it is.
- When you relate two tables on the join condition by equating the columns with equal(=) symbol, then it is called an Equi-Join.
- Equi-joins are also called as simple joins.

```
SELECT employee_id, department_name
FROM employee e, departments d
WHERE e.department_id = d.department_id;
```

DML Statements - Joins

Equi Join - Example

Customers(customer_id, customer_name)

Products(product_id, product_name)

Sales(sale_id, price, customer_id, product_id)

Q. Write a sql query to get the products purchased by a customer.

```
SELECT c.customer_name, p.product_name
FROM Customers c, Sales s, Products p
WEHRE c.customer_id = s.customer_id AND s.product_id =
p.product_id
```

View

- An Oracle VIEW, in essence, is a virtual table that does not physically exist. Rather, it is created by a query joining one or more tables.

Syntax

```
CREATE VIEW view_name AS
SELECT columns
FROM tables
WHERE conditions;
```

View

Example

```
CREATE VIEW sup_orders AS
SELECT suppliers.supplier_id, orders.quantity, orders.price
FROM suppliers
INNER JOIN orders ON suppliers.supplier_id = orders.supplier_id
WHERE suppliers.supplier_name = 'Microsoft';
```

View

Update View

- You can modify the definition of an Oracle VIEW without dropping it by using the Oracle CREATE OR REPLACE VIEW Statement.
- If the Oracle VIEW did not yet exist, the VIEW would merely be created for the first time.

Syntax

```
CREATE OR REPLACE VIEW view_name AS
SELECT columns
FROM table
WHERE conditions;
```

View

Update View

Example

```
CREATE or REPLACE VIEW sup_orders AS
SELECT suppliers.supplier_id, orders.quantity, orders.price
FROM suppliers
INNER JOIN orders ON suppliers.supplier_id = orders.supplier_id
WHERE suppliers.supplier_name = 'Apple';
```

View

Update View

Point to remember

- The Primary key column of the table should be included in the view.
- Aggregate functions cannot be used in select statement.
- Select statement used for creating a view should not include Distinct ,Group by or Having Clause.
- The Select statement used for creating a view should not include subqueries.

Drop View

```
DROP VIEW sup_orders;
```

Transaction Control Commands

- Transaction control commands manage changes made by DML commands.

What is Transaction ?

- Collection of operation that forms a single logical unit of work are called Transaction.
- Transaction can either be one DML statement or a group of statements.
- Transaction must be atomic.
- All transactions have beginning and an end.
- A transaction can be saved or undone.
- If transaction failed in middle no part of the transaction can be saved.

Transaction Control Commands

COMMIT

- **COMMIT statement** commits all changes for the current transaction. Once a commit is issued, other users will be able to see your changes.

  COMMIT [WORK] [COMMENT clause] [WRITE clause] [FORCE clause];

  Work : Optional. It was added by Oracle to be SQL-compliant. Issuing the COMMIT with or without the WORK parameter will result in the same outcome.

  Comment : Optional. It is used to specify a comment to be associated with the current transaction.

- The comment that can be up to 255 bytes of text enclosed in single quotes.

Transaction Control Commands

COMMIT

WRITE clause : Optional.

- It is used to specify the priority that the redo information for the committed transaction is to be written to the redo log.
- With this clause, you have two parameters to specify:

***WAIT* or *NOWAIT* (*WAIT* is the default if omitted)**

WAIT - means that the commit returns to the client only after the redo information is persistent in the redo log.

NOWAIT - means that the commit returns to the client right away regardless of the status of the redo log.

- It is stored in the system view called DBA_2PC_PENDING along with the transaction ID if there is a problem.

Transaction Control Commands

***IMMEDIATE* or *BATCH* (*IMMEDIATE* is the default if omitted)**

***IMMEDIATE* -** forces a disk I/O causing the log writer to write the redo information to the redo log.

***BATCH* -** forces a "group commit" and buffers the redo log to be written with other transactions.

Transaction Control Commands

FORCE clause

- Optional. It is used to force the commit of a transaction that may be corrupt or in doubt.
- ***FORCE 'string'*** - It allows you to commit a corrupt or in doubt transaction in a distributed database system by specifying the transaction ID in single quotes as *string*.
- You can find the transaction ID in the system view called DBA_2PC_PENDING.

FORCE CORRUPT_XID 'string' -

- It allows you to commit a corrupt or in doubt transaction by specifying the transaction ID in single quotes as *string*.

Transaction Control Commands

- You can find the transaction ID in the system view called V$CORRUPT_XID_LIST.
- ***FORCE CORRUPT_XID_ALL*** - It allows you to commit all corrupted transactions.

- You must have DBA privileges to access the system views - DBA_2PC_PENDING and V$CORRUPT_XID_LIST.
- You must have DBA privileges to specify certain features of the COMMIT statement.

```
COMMIT COMMENT 'This is the comment for the transaction';
COMMIT FORCE '22.14.67';
```