Database Management System (CS-2004) Lab

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

School of Computer Engineering

4 Credit

Mr. Rajat Kumar Behera - Associate Professor

Lab Contents

Sr #	Major and Detailed Coverage Area	Lab#
1	DQL – Part 3 ❑ Table Joining ❑ Union Operator ❑ Union All Operator ❑ Select Top Operator	5
2	DDL Recap ❑ Auto Increment ❑ Select Into clause ❑ Creation of table from existing table	

Selecting data from Multiple Tables

Having the capability to select data from multiple tables is one of SQL's most powerful features. Without this capability, the entire relational database concept would not be feasible. Single-table queries are sometimes quite informative, but in the real world, the most practical queries are those whose data is acquired from multiple tables within the database.

Joins

A join combines two or more tables to retrieve data from multiple tables. Although different implementations have many ways of joining tables, we will concentrate on the most common joins in this lesson. The types of joins are:

- Inner joins
- Outer joins
  - Left Outer joins
  - Right Outer joins
  - Full Outer joins
- Self joins
- Non-equi joins
- Cross joins or Cartesian Product
- Natural joins

Reference Table

Customer

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

Order

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

Reference Table

Employee

EmployeeID	LastName	FirstName	BirthDate	Photo	Notes
1	Davolio	Nancy	12/8/1968	EmpID1.pic	Education includes a BA in psychology.....
2	Fuller	Andrew	2/19/1952	EmpID2.pic	Andrew received his BTS commercial and....
3	Leverling	Janet	8/30/1963	EmpID3.pic	Janet has a BS degree in chemistry....

Supplier

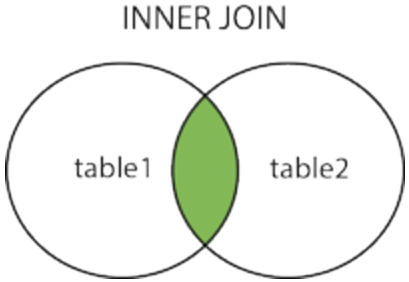
SupplierID	SupplierName	ContactName	Address	City	PostalCode	Country
1	Exotic Liquid	Charlotte Cooper	49 Gilbert St.	London	EC1 4SD	UK
2	New Orleans Cajun Delights	Shelley Burke	P.O. Box 78934	New Orleans	70117	USA
3	Grandma Kelly's Homestead	Regina Murphy	707 Oxford Rd.	Ann Arbor	48104	USA

Components of Join

- Both the SELECT and FROM clauses are required in the SQL statement.
- WHERE clause is a required element of an SQL statement when joining tables as the join is performed in the WHERE clause.
- Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT.
- However, the most common operator is the equal symbol.

Inner Join

Perhaps the most used and important. It joins two tables with a common column in which each or one is usually the primary key. It selects all rows from both tables as long as there is a match between the columns in both tables.



Inner & Outer Join

Syntax 1:	**Syntax 2:**
SELECT column_name(s)	SELECT column_name(s)
FROM table1	FROM table1
INNER JOIN table2	**JOIN** table2
ON table1.column_name=table2.column_name;	*ON table1.column_name=table2.column_name;*

Example:

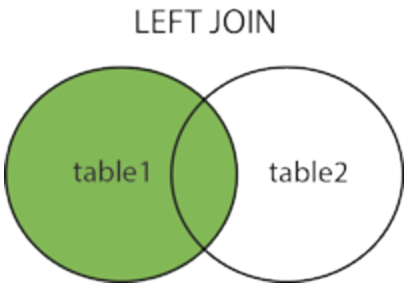
```
SELECT CUSTOMER.CUSTOMERNAME, ORDER.ORDERID
FROM CUSTOMER
INNER JOIN ORDER
ON CUSTOMER.CUSTOMERID=ORDER.CUSTOMERID;
```

Outer Join

Outer join finds and returns matching data and some dissimilar data from tables.

Left Outer Join

The LEFT OUTER JOIN returns all rows from the left table (table1), with the matching rows in the right table (table2). The result is **NULL in the right side** when there is no match.



Syntax:

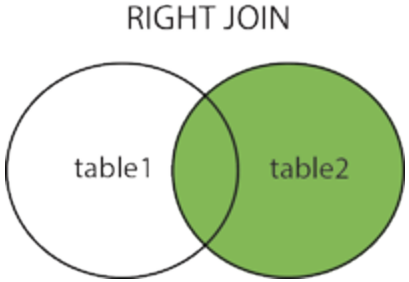
```
SELECT column_name(s)
FROM table1
LEFT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

Example:

```
SELECT CUSTOMER.CUSTOMERNAME, ORDER.ORDERID
FROM CUSTOMER
LEFT OUTER JOIN ORDER
ON CUSTOMER.CUSTOMERID=ORDER.CUSTOMERID;
```

Right Outer Join

The RIGHT JOIN returns all rows from the right table (table2), with the matching rows in the left table (table1). The result is **NULL in the left side** when there is no match.



Syntax:

```
SELECT column_name(s)
FROM table1
RIGHT OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

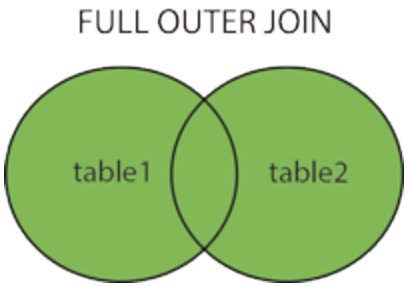
Example:

```
SELECT ORDER.ORDERID, EMPLOYEE.FIRSTNAME
FROM ORDER
RIGHT OUTER JOIN EMPLOYEE
ON ORDER.EMPLOYEEID=EMPLOYEE.EMPLOYEEID;
```

Full Outer Join

The FULL OUTER JOIN returns all rows from the left table (table1) and from the right table (table2). It combines the result of both LEFT and RIGHT joins.

Syntax:




```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name=table2.column_name;
```

Example:

```
SELECT CUSTOMER.CUSTOMERNAME, ORDER.ORDERID
FROM CUSTOMER
FULL OUTER JOIN ORDER
ON CUSTOMER.CUSTOMERID=ORDER.CUSTOMERID
ORDER BY CUSTOMER.CUSTOMERNAME;
```

Outer Join Explained

R	
Employee	**Dept**
Smith	Sales
Black	Production
White	Production

S	
Dept	**Head**
Production	Mori
Purchasing	Brown

Outer Join Results

R LEFT OUTER JOIN S		
Employee	**Dept**	**Head**
Smith	Sales	**NULL**
Black	Production	Mori
White	Production	Mori

R RIGHT OUTER JOIN S		
Employee	**Dept**	**Head**
Black	Production	Mori
White	Production	Mori
NULL	Purchasing	Brown

R FULL OUTER JOIN S		
Employee	**Dept**	**Head**
Smith	Sales	**NULL**
Black	Production	Mori
White	Production	Mori
NULL	Purchasing	Brown

Self Join

A self join is a join in which a table is joined with itself. To join a table itself means that each row of the table is combined with itself and with every other row of the table. The self join can be viewed as a join of two copies of the same table. The table is not actually copied, but SQL performs the command as though it were. The syntax of the command for joining a table to itself is almost same as that for joining two different tables. To distinguish the column names from one another, aliases for the actual the table name are used, since both the tables have the same name. Table name aliases are defined in the FROM clause of the SELECT statement.

Syntax :

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

Self Join cont...

EMP_ID	EMP_NAME	DT_OF_JOIN	EMP_SUPV
20051	Vijes Setthi	15-JUN-09	-
20073	Unnath Nayar	09-AUG-10	20051
20064	Rakesh Patel	23-OCT-09	20073
20069	Anant Kumar	03-DEC-08	20051
20055	Vinod Rathor	27-NOV-09	20051
20075	Mukesh Singh	25-JAN-11	20073

Query

```
SELECT a.emp_id AS "Emp_ID",a.emp_name AS "Employee Name",
b.emp_id AS "Supervisor ID",b.emp_name AS "Supervisor Name"
FROM employee a, employee b  WHERE a.emp_supv = b.emp_id;
```

Output

Emp_ID	Employee Name	Supervisor ID	Supervisor Name
20055	Vinod Rathor	20051	Vijes Setthi
20069	Anant Kumar	20051	Vijes Setthi
20073	Unnath Nayar	20051	Vijes Setthi
20075	Mukesh Singh	20073	Unnath Nayar
20064	Rakesh Patel	20073	Unnath Nayar

Non-equi and Cross Join

- A non-equijoin joins two or more tables based on a specified column value not equaling a specified column value in another table. The syntax for the non-equijoin is

  FROM TABLE1, TABLE2 [, TABLE3]

  WHERE TABLE1.COLUMN_NAME != TABLE2.COLUMN_NAME

  [AND TABLE1.COLUMN_NAME != TABLE2.COLUMN_NAME]

- The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from the two or more joined tables. Syntax is

  SELECT table1.column1, table2.column2... FROM table1, table2 [, table3]

  Example – SELECT CUSTOMERNAME, CITY FROM CUSTOMER, ORDER

Natural Join

EQUI JOIN performs a JOIN against equality or matching column(s) values of the associated tables and an equal sign (=) is used as comparison operator in the where clause to refer equality. The **NATURAL JOIN** is a type of EQUI JOIN and is structured in such a way that, columns with the same name of associated tables **will appear only once**.

Guidelines:

- 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.

Syntax : SELECT * FROM table1 NATURAL JOIN table2;

Example : SELECT * FROM CUSTOMER NATURAL JOIN ORDER;

Union Operator

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

Syntax:

SELECT column_name(s) FROM table1

UNION

SELECT column_name(s) FROM table2;

Example:

SELECT City FROM Customer

UNION

SELECT City FROM Suppliers

ORDER BY City;

Union All Operator

The UNION ALL operator select **all** including duplicate values.

Syntax:

SELECT column_name(s) FROM table1

UNION ALL

SELECT column_name(s) FROM table2;

Example :

SELECT City FROM Customer

UNION ALL

SELECT City FROM Suppliers

ORDER BY City;

Class work

Select all (duplicate values also) German cities from the Customer and Supplier tables.

Select Top

The SELECT TOP clause is used to specify the number of records to return. It can be very useful on large tables with thousands of records. Returning a large number of records can impact on performance.

Syntax:

SELECT column_name(s) FROM table_name WHERE **ROWNUM <condition> e.g. <= number**;

Examples:

SELECT * FROM EMPLOYEE WHERE ROWNUM <=5;

SELECT * FROM Customer WHERE ROWNUM >5 AND ROWNUM <15;

Class work

Select all (duplicate values also) German top 5 cities from the Customer and Supplier tables.

Auto Increment

Very often we would like the value of the primary key field to be created automatically every time a new record is inserted. Auto-increment allows a unique number to be generated when a new record is inserted into a table.

To create it, you will have to create an **auto-increment field** with the **sequence object** (this object generates a number sequence)

Creation of sequence:

```
CREATE SEQUENCE emp_sequence
MINVALUE 1
START WITH 1
INCREMENT BY 1
CACHE 10
```

The code is creating a sequence object called emp_sequence, that starts with 1 and will increment by 1. It will also cache up to 10 values for performance. The cache option specifies how many sequence values will be stored in memory for faster access.

Sequence Usage:

```
INSERT INTO PERSON (ID, FirstName, LastName)
VALUES (emp_sequence.nextval,'Lars','Monsen')
```

SELECT INTO

The **SELECT INTO** statement copies data from one table and inserts it into a new table.

Syntax:

```
SELECT * INTO newtable
FROM table1;
```

```
SELECT column_name(s)
INTO newtable
FROM table1;
```

Examples:

```
SELECT  *
INTO CustomerBackup12072013
FROM Customer;
```

```
SELECT  CustomerName, ContactName
INTO CustomerBackup12072013
FROM Customer;
```

```
SELECT *
INTO CustomerBackup12072013
FROM Customers
WHERE Country='Germany';
```

```
SELECT Customers.CustomerName, Orders.OrderID
INTO CustomerOrderBackup2013
FROM Customer LEFT JOIN Order ON Customers.CustomerID=Orders.CustomerID;
```

Table Creation from existing Table

SQL CREATE TABLE AS statement is used to create a table from an existing table by copying the existing table's columns. It is important to note that when creating a table in this way, the new table will be populated with the records from the existing table.

Copying all columns

Syntax:

CREATE TABLE new_table AS (SELECT * FROM old_table);

Example:

CREATE TABLE SUPPLIER AS (SELECT * FROM COMPANY);

Copying few columns

Syntax:

CREATE TABLE new_table AS (SELECT COLUMN (s) FROM old_table);

Example:

CREATE TABLE supplier AS (SELECT id, address, city, state, zip FROM company);

Copying selected columns from more than one table

CREATE TABLE SUPPLIER AS (SELECT company.id, company.address, category.cat_type FROM company, category WHERE company.id = category.id AND category.id > 1000);

Thank You
End of Lab 5

Assignment

Reference Tables

- **EMPLOYEE** table with the attributes:

  ID, LAST_NAME, FIRST_NAME, MIDDLE_NAME, FATHER_NAME, MOTHER_NAME, SEX, HIRE_DATE, ADDRESS, CITY, STATE, ZIP, PHONE, PAGER, SUPERVISOR_ID, INJECTED_DATE

- **SCHOOL** table with the attributes:

  ID, NAME, INJECTED_DATE

- **EMPLOYEE _ALIGNMENT** table with the attributes:

  EMPLOYEE_ID, SCHOOL_ID, INJECTED_DATE

- **JOB** table with the attributes:

  ID, NAME, TITLE, SALARY, BONUS , INJECTED_DATE

- **EMPLOYEE _PAY** table with the attributes:

  EMPLOYEE_ID, JOB_ID, INJECTED_DATE

Assignment

1. Display all employee's full name and their school names.
2. Display all employee's full name and their job title and salary.
3. Display all employee's full name with their job name, title and total salary for non-null bonus
4. Find the full name of the supervisors
5. Find the name, ID and number of supervisee for each supervisor
6. For each school, find the name, ID and number of supervisee for each supervisor
7. Find the employees who are working as "Associate Professor" in school of "Computer Engineering" or "Electronic Engineering"
8. Find the employees who are working as "Professor" in school of "Computer Engineering" or "Mechanical Engineering" and whose bonus is NULL.
9. For each school & title, find the average salary and number of employees
10. Who works in the same school in which John Smith works?

Assignment

11. For each job title, display the number of employees who are drawing more than 1,00,000 total salary and not working in “Computer Engineering” school
12. Display the count of the employee who are tagged to more than one school.
13. For each school, find the average number of employees who are hired in last year.
14. For each school, find the total number of employees who are hired as “Professor” in last month
15. For each school, find the full name of the employees who served more than 4 years as “Professor” or “Associate Professor”
16. Insert employee ID, last name, first name to a new table who are working as “Professor” in “Computer Engineering” school.
17. Using union, select all the employees working in “Computer Engineering” and “Civil Engineering” school

Assignment

18. Create an new table “EMP_24012017” from EMPLOYEE
19. Insert data to “EMP_24012017” for the employees belongs to “Computer Engineering” school
20. Create a sequence called dummy_seq and associate with a table