Experiment 7

Objectives: Queries for (Equi-join,Non-Equi join,outer join)

Joins

A join is a query that combines rows from two or more tables, views, or materialized views. Oracle performs a join whenever multiple tables appear in the query's FROM clause. The query's select list can select any columns from any of these tables. If any two of these tables have a column name in common, you must qualify all references to these columns throughout the query with table names to avoid ambiguity.

Join Conditions

Most join queries contain WHERE clause conditions that compare two columns, each from a different table. Such a condition is called a join condition. To execute a join, Oracle combines pairs of rows, each containing one row from each table, for which the join condition evaluates to TRUE. The columns in the join conditions need not also appear in the select list.

To execute a join of three or more tables, Oracle first joins two of the tables based on the join conditions comparing their columns and then joins the result to another table based on join conditions containing columns of the joined tables and the new table. Oracle continues this process until all tables are joined into the result. The optimizer determines the order in which Oracle joins tables based on the join conditions, indexes on the tables, and, in the case of the cost-based optimization approach, statistics for the tables.

Equijoins

An equijoin is a join with a join condition containing an equality operator. An equijoin combines rows that have equivalent values for the specified columns.

Example: Selecting Data from Two Tables (Joining Two Tables)

```
SELECT EMPLOYEES.FIRST_NAME "First",
EMPLOYEES.LAST_NAME "Last",
DEPARTMENTS.DEPARTMENT_NAME "Dept. Name"
FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID;
```

Result:

First	Last	Dept. Name
Jennifer	Whalen	Administration
Michael	Hartstein	Marketing
Pat	Fay	Marketing
Den	Raphaely	Purchasing
Karen	Colmenares	Purchasing
Alexander	Khoo	Purchasing
Shelli	Baida	Purchasing
Sigal	Tobias	Purchasing
Guy	Himuro	Purchasing
Susan	Mavris	Human Resources
Donald	Oconnell	Shipping

First Last Dept. Name

Douglas Grant Shipping

...

Shelley Higgins Accounting

Table-name qualifiers are optional for column names that appear in only one table of a join, but are required for column names that appear in both tables. The following query is equivalent to the query in Example

```
SELECT FIRST_NAME "First",
LAST_NAME "Last",
DEPARTMENT_NAME "Dept. Name"
FROM EMPLOYEES, DEPARTMENTS
WHERE EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID;
```

To make queries that use qualified column names more readable, use table aliases, as in the following example:

```
SELECT FIRST_NAME "First",
LAST_NAME "Last",
DEPARTMENT_NAME "Dept. Name"
FROM EMPLOYEES e, DEPARTMENTS d
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID;
```

Although you create the aliases in the FROM clause, you can use them earlier in the query, as in the following example:

```
SELECT e.FIRST_NAME "First", e.LAST_NAME "Last",
d.DEPARTMENT_NAME "Dept. Name" FROM EMPLOYEES e, DEPARTMENTS d
WHERE e.DEPARTMENT_ID = d.DEPARTMENT_ID;
```

Create the tables with the appropriate integrity constraints and Insert around 10 records in each of the tables

SQL> create table customer1 (cust_id number(5) primary key, cust_name varchar2(15)); Output: Table created.

SQL> desc customer1;

Output:

Name Null? Type

CUST_ID NOT NULL NUMBER(5)

CUST_NAME VARCHAR2(15)

Valid Test Data

b) SQL> insert into customer1 values(&custid,'&custname'); SQL> select * from customer1; Output:

CUST_ID CUST_NAME

---

100 ramu

101	kamal
102	raju
103	raju sundaram
104	lawrence

SQL> create table item(item_id number(4) primary key, item_name varchar2(15),price number(6,2)); SQL> dsec item

Output:

Name	Null?	Type
Cust_id	NOT NULL	NUMBER(4)
Item_name		VARCHAR2(15)
PRICE		NUMBER(6,2)

SQL>insert into item values(&item_id,'&item_name',&price);

SQL> select * from item; Output:

ITEM_ID	ITEM_NAME	PRICE
2334	geera	6.25
4532	corn soup	34.65
2124	lays chips	20
4531	setwet	99.99
2319	duracell	45.5

SQL>create table sale(bill_no number(5) primary key,bill_date date, cust_id number(5) references customer(cust_id), item_id number(4) references item(item_id),qty_sold number(4));

Output: Table Created.

SQL>dsec sale Output:

Name	Null?	Type
BILL_NO	NOT NULL	NUMBER(4)
BILL_DATE		DATE
CUST_ID		NUMBER(5)
ITEM_ID		NUMBER(4)
QTY_SOLD		NUMBER(4)

SQL>insert into Sale values(&bill_no, '&bill_date', &cust_id, &item_id, &qty_sold);

SQL>select * from sale;

Output:

BILL_NO	BILL_DATE	CUST_ID	ITEM_ID	QTY_SOLD
1450	04-JAN-06	100	2124	2
1451	04-JAN-06	101	2319	1
1452	04-JAN-06	103	4531	2
1453	04-JAN-06	102	2334	3
1454	04-JAN-06	104	4532	3

c) List all the bills for the current date with the customer names and item numbers

SQL> select c.custname, i.itemid, s.billno from customer c, item I, sale s where c.custid=s.custid and s.billdate=to_char(sysdate);

CUSTNAME	ITEMID	BILLNO
John	5001	332

d) List the total Bill details with the quantity sold, price of the item and the final amount

SQL> select i.price, s.qty,(i.price*s.qty) total from item I, sale s where i.itemid=s.itemid;

PRICE	QTY	TOTAL
120	2	240
20	3	60
5	2	10
10	1	10
350	4	1400

e) List the details of the customer who have bought a product which has a price>200

SQL> select c.custid, c.custname from customer c, sale s, item i where i.price>200 and c.custid=s.custid and i.itemid=s.itemid;

CUSTID	CUSTNAME
4	duffy

f) Give a count of how many products have been bought by each customer SQL> select custid, count(itemid) from sale group by custid;

CUSTID	COUNT(ITEMID)
1	2
3	1
4	1
5	1

g) Give a list of products bought by a customer having cust_id as 5

SQL> select i.itemname from item i, sale s where s.custid=5 and i.itemid-s.itemid;

ITEMNAME

Pens

h) List the item details which are sold as of today

SQL> select i.itemid, i.itemname from item I, sale s where i.itemid=s.itemid and s.billdate=to_char(sysdate);

ITEMID ITEMNAME

---------- ----------

1234 Pencil

Outer join

An outer join does not require each record in the two joined tables to have a matching record. The joined table retains each record—even if no other matching record exists. Outer joins subdivide further into left outer joins, right outer joins, and full outer joins, depending on which table's rows are retained (left, right, or both).

(In this case left and right refer to the two sides of the JOIN keyword.) No implicit join-notation for outer joins exists in standard SQL.

Left outer join

The result of a left outer join (or simply left join) for tables A and B always contains all records of the "left" table (A), even if the join-condition does not find any matching record in the "right" table (B). This means that if the ON clause matches 0 (zero) records in B (for a given record in A), the join will still return a row in the result (for that record)—but with NULL in each column from B. A left outer join returns all the values from an inner join plus all values in the left table that do not match to the right table.

For example, this allows us to find an employee's department, but still shows the employee(s) even when they have not been assigned to a department (contrary to the inner-join example above, where unassigned employees were excluded from the result).

Example of a left outer join (the OUTER keyword is optional), with the additional result row (compared with the inner join) italicized:

```
SELECT *
FROM employee LEFT OUTER JOIN department
  ON employee.DepartmentID = department.DepartmentID;
```

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
John	NULL	NULL	NULL
Heisenberg	33	Engineering	33

Right outer join

A right outer join (or right join) closely resembles a left outer join, except with the treatment of the tables reversed. Every row from the "right" table (B) will appear in the joined table at least once. If no matching row from the "left" table (A) exists, NULL will appear in columns from A for those records that have no match in B.

A right outer join returns all the values from the right table and matched values from the left table (NULL in the case of no matching join predicate). For example, this allows us to find each employee and his or her department, but still show departments that have no employees.

Below is an example of a right outer join (the OUTER keyword is optional), with the additional result row italicized:

```
SELECT *
```

FROM employee RIGHT OUTER JOIN department

ON employee.DepartmentID = department.DepartmentID;

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Heisenberg	33	Engineering	33
Rafferty	31	Sales	31
NULL	NULL	Marketing	35

Right and left outer joins are functionally equivalent. Neither provides any functionality that the other does not, so right and left outer joins may replace each other as long as the table order is switched.

Self-join

A self-join is joining a table to itself.

Example

A query to find all pairings of two employees in the same country is desired. If there were two separate tables for employees and a query which requested employees in the first table having the same country as employees in the second table, a normal join operation could be used to find the answer table. However, all the employee information is contained within a single large table.

Consider a modified Employee table such as the following:

EmployeeID	LastName	Country	DepartmentID
123	Rafferty	Australia	31
124	Jones	Australia	33
145	Heisenberg	Australia	33
201	Robinson	United States	34
305	Smith	Germany	34
306	John	Germany	NULL

An example solution query could be as follows:

SELECT F.EmployeeID, F.LastName, S.EmployeeID, S.LastName, F.Country

FROM Employee F INNER JOIN Employee S ON F.Country = S.Country

WHERE F.EmployeeID < S.EmployeeID

ORDER BY F.EmployeeID, S.EmployeeID;

Which results in the following table being generated.

EmployeeID	LastName	EmployeeID	LastName	Country
123	Rafferty	124	Jones	Australia
123	Rafferty	145	Heisenberg	Australia
124	Jones	145	Heisenberg	Australia
305	Smith	306	John	Germany

For this example:

- F and S are aliases for the first and second copies of the employee table.
- The condition F.Country = S.Country excludes pairings between employees in different countries. The example question only wanted pairs of employees in the same country.
- The condition F.EmployeeID < S.EmployeeID excludes pairings where the EmployeeID of the first employee is greater than or equal to the EmployeeID of the second employee. In other words, the effect of this condition is to exclude duplicate pairings and self-pairings. Without it, the following less useful table would be generated (the table below displays only the "Germany" portion of the result):

EmployeeID	LastName	EmployeeID	LastName	Country
305	Smith	305	Smith	Germany
305	Smith	306	John	Germany
306	John	305	Smith	Germany
306	John	306	John	Germany