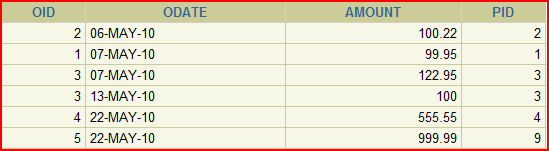
Join tables for practice

CUSTOMERS



ORDERS



Here is the script to run. Just cut and paste it into the iSQLplus box and execute

drop table customers;

drop table orders;

CREATE TABLE CUSTOMERs(

PID NUMBER(1) NOT NULL,

PNAME VARCHAR2(20),

PEMAIL VARCHAR2(20)

);

CREATE TABLE ORDERS

(

OID NUMBER(30) NOT NULL,

ODATE DATE,

AMOUNT NUMBER(6,2),

PID NUMBER (1)

);

INSERT INTO orders VALUES (2, '06-May-2010', 100.22, 2);

INSERT INTO orders VALUES (1, '07-May-2010', 99.95, 1);

INSERT INTO orders VALUES (3, '07-May-2010', 122.95, 3);

INSERT INTO orders VALUES (3, '13-May-2010', 100.00, 3);

INSERT INTO orders VALUES (4, '22-May-2010', 555.55, 4);

INSERT INTO orders VALUES (5, '22-May-2010', 999.99, 9);

INSERT INTO customers VALUES (1, 'John Smith','John.Smith@yahoo.com');

INSERT INTO customers VALUES (2, 'Steven Goldfish','goldfish@fish.net');

INSERT INTO customers VALUES (3, 'Paula Brown', 'pb@domain.org');

INSERT INTO customers VALUES (4, 'James Smith', 'jim@sup.co.uk');

INSERT INTO customers VALUES (5, 'Uncle Joe', 'UNK@sympatico.ca');

select \* from customers;

select \* from orders;

OBJECTIVES – JOINS

**After completing this lesson, you should be able to do the following:**

1 Write SELECT statements to access data from more than one table using

**- Equijoins and**

**- Non-equijoins**

2 Join a table to itself by using a **self-join**

3 View data that generally does not meet a join condition

- By using **outer joins**

4 Join a table to itself by using a **self-join**

5 Generate a Cartesian product of all rows from twoor more tables

Already covered simple JOINS in DBS201. This is a refresher and an expansion

EQUIJOINS – SIMPLE JOINS

Sometimes you need data from more than one table.

Given the tables ORDERS and CUSTOMERS you may need to know data from 2 tables such as

🡪 How much did a customer purchase

The data about customers is in one table and the data about how much was sold is in the ORDERS table.

From your design class you know that there needs to be a common field in order to get the data from 2 or more tables.

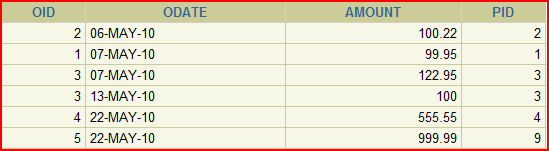
The common field is PID

**PROBLEM PROVIDE A LIST OF CUSTOMERS AND THEIR SALES AMOUNTS**

CUSTOMERS



ORDERS



SIMPLE JOIN and EQUIJOINS

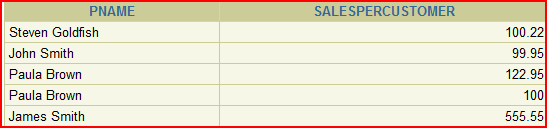
SELECT Customers.pname,

ORDERS.Amount AS SalesPerCustomer

FROM Customers, Orders

WHERE Customers.pid = orders.pid

The condition was to join on the common field



Where there is a common field in both tables the rows are joined making the result a long row. The fields in the select statement are then chosen from the long row and displayed.

NATURAL JOIN

You can join tables based on two tables that

A 🡪 That have matching data types and names

B 🡪 Have the same data type

You can do this by using the keywords 🡪 NATURAL JOIN.

EXAMPLE:

SELECT department\_id, department\_name, location \_id, city

FROM departments

NATURAL JOIN locations

This is slightly different than the SIMPLE or EQUIJOIN

Using SQL1999 JOINS

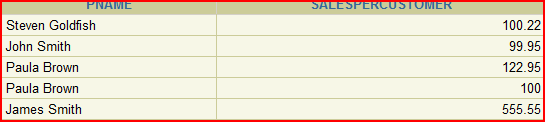
**SELECT pname,**

Slight variation to simple joins designed for readability

**amount AS SalesPerCustomer**

**FROM Customers JOIN Orders**

**ON Customers.pid = orders.pid**

****

**Notice that Paula Brown has 2 orders, so how can it be improved to show a total amount from the customer**

**IMPROVEMENT**

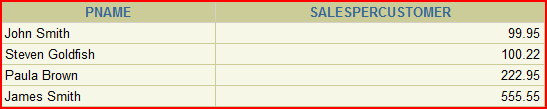
**SELECT pname,**

**SUM(Amount) AS SalesPerCustomer**

**FROM Customers JOIN Orders**

**ON Customers.pid = orders.pid**

**GROUP BY PNAME** 🡨 MUST HAVE THIS IN AS HAVE A GROUP AND SINGLE ROW



JOINS plus WHERE clause

**PROBLEM: 🡪**  The following example limits the rows of output to those with a department ID equal to 20 or 50:

**SELECT department\_id, department\_name, location\_id, city**

**FROM departments**

**NATURAL JOIN locations**

**WHERE department\_id IN (20, 50);**

Qualifying Ambiguous Column Names

SELECT pname,

amount AS SalesPerCustomer

PID exists in both tables. Oracle needs to know which one you are referring to, so needs to be qualified.

FROM Customers JOIN Orders

ON Customers.pid = orders.pid

# CAN USE AN ALIAS

SELECT pname,

amount AS SalesPerCustomer

FROM Customers C, Orders O

WHERE C.pid = O.pid

🡪 Use alias to simplify the queries

🡪 Use table aliases to improve performance

2 types of joins INNER and OUTER

If you don’t state INNER or OUTER the default is INNER

INNER JOIN then is the same as JOIN

# INNER JOINS

The **INNER JOIN** will select all rows from both tables 🡪 as long as there is a match between the columns we are matching on.

🡪🡪 If a customer has not placed an order or has not placed an order in the time we might specify, then this customer will not be listed as there is no common field.

### PROBLEM

**Display ALL customers and their sales**

To solve this requires an OUTER JOIN

2 types of OUTER JOINS

🡪 LEFT

🡪 RIGHT

# LEFT JOIN

SELECT pname,

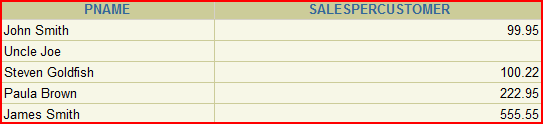
SUM(Amount) AS SalesPerCustomer

FROM Customers **LEFT** JOIN Orders

ON Customers.pid = orders.pid

GROUP BY PNAME

**Now there are 5 rows showing all customers and their orders even if they have no order**



# OLDER METHOD

SELECT pname,

SUM(Amount) AS SalesPerCustomer

FROM Customers, Orders

WHERE Customers.pid(+)= orders.pid

GROUP BY PNAME

# RIGHT JOIN

See what happens

SELECT pname,

SUM(Amount) AS SalesPerCustomer

FROM Customers **RIGHT** JOIN Orders

ON Customers.pid = orders.pid

GROUP BY PNAME

Look at the first row.

# 

# Why is this? 🡨 BAD DESIGN

This is a case of the system allowing an order for a non-existent customer == BAD DESIGN

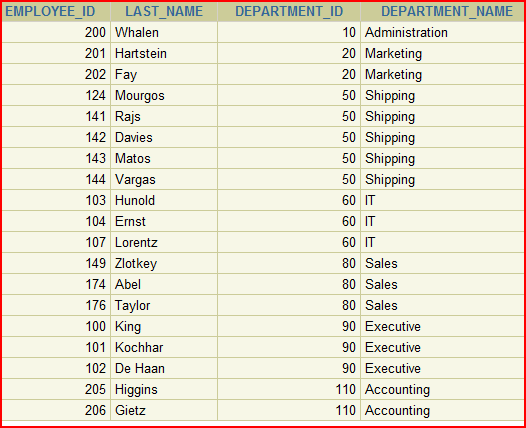
But it does allow us to demonstrate a RIGHT JOIN

USING

SELECT employee\_id, Last\_name, Department\_id, department\_name

FROM employees JOIN departments

USING (department\_id)



Works only if

A) column name the same

AND

B) Data type the same

ON

SELECT employee\_id, Last\_name, departments.Department\_id, department\_name

FROM employees JOIN departments

ON (employees.department\_id = departments.department\_id)

🡨 Need to clarify which department name using

Notice it worked fine for the USING

SELECT Customers.pname,

SUM(orders.Amount) AS SalesPerCustomer

FROM Customers **RIGHT** JOIN Orders

ON Customers.pid = orders.pid

GROUP BY Customers.PNAME

Join more Tables

# NEW PROBLEM using EMPLOYEE table

Sometimes you need to join a table to itself.

# PROBLEM: Find the name of each employees manager

To find the name of Lorentz’s manager, you need to:

- find the employee in the employee table by looking up the LAST\_NAME that matches

- Find the manager number in the row for Lorentz by looking at MANAGER\_ID (103)

- Find name of manager by looking at EMPLOYEE\_ID for 103 and looking at LAST\_NAME

### 🡺 LOOKING AT TABLE TWICE

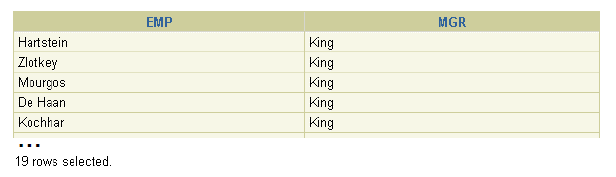
## 🡺 MANAGER\_ID in the worker table is EQUAL to EMPLOYEE\_ID in the MANAGER table

EXAMPLE SELF JOIN

**SELECT e.last\_name AS emp, m.last\_name AS mgr**

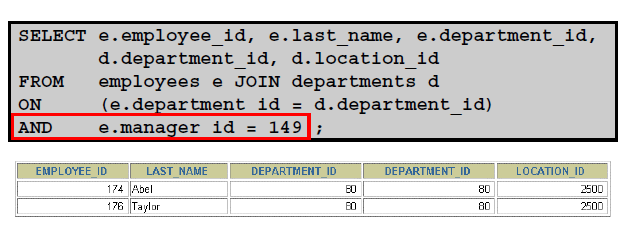
**FROM employees e JOIN employees m**

**ON (e.manager\_id = m.employee\_id);**



**NOTE: 19 rows … WHY? – Why not 20**

Applying Conditions to SELF-JOINS



THREE-WAY JOINS

A three-way join is a join of three tables.

**SELECT employee\_id, city, department\_name**

**FROM employees e**

**JOIN departments d**

**ON d.department\_id = e.department\_id**

**JOIN locations l**

**ON d.location\_id = l.location\_id**

In SQL:1999–compliant syntax, joins are performed from left to right.

So the first join to be performed is EMPLOYEES JOIN DEPARTMENTS.

The first join condition can reference columns in EMPLOYEES and DEPARTMENTS but cannot reference columns in LOCATIONS.

The second join can reference columns from all 3 tables

ALTERNATE METHOD

**SELECT employee\_id, city, department\_name**

**FROM employees e, departments d, locactions l**

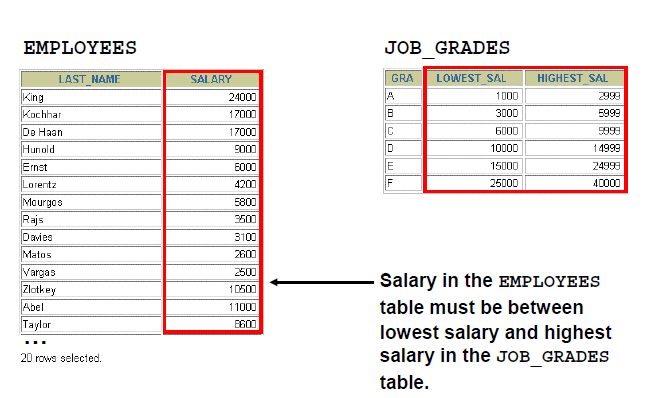
**WHERE d.department\_id = e.department\_id**

**AND d.location\_id = l.location\_id**

The rule is that the joins is 1 less than number of tables. In above there are 2 conditions for 3 tables. If joining 20 tables there would be 19 conditions.

NON-EQUI-JOINS

# A JOIN that is not using an equality operator



A relationship between the two tables is that the SALARY column in the EMPLOYEES table must be between the values in the LOWEST\_SALARY and HIGHEST\_SALARY columns of the JOB\_GRADES table.

SELECT e.last\_name, e.salary, j.grade\_level

FROM employees e, job\_grdes j

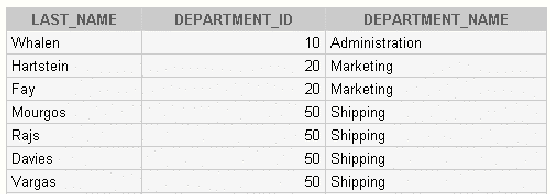
WHERE e.salary BETWEEN j.lowest\_sal AND j.highest\_sal

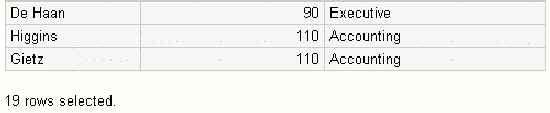
NOTE: Specify lowest first on a between

CROSS-JOIN – CARTESIAN PRODUCT

# This is what happens when you don't specify a join condition.







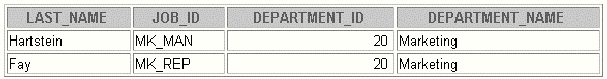
1. Write a query for the HR department to produce the addresses of all the departments.

Use the LOCATIONS and COUNTRIES tables. Show the location ID, street address, city, state or province, and country in the output. Use a NATURAL JOIN to produce the results.

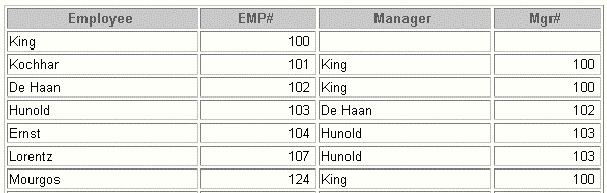
2. The HR department needs a report of all employees. Write a query to display the last name, department number, and department name for all employees.

…

3. The HR department needs a report of employees in Toronto. Display the last name, job, department number, and department name for all employees who work in Toronto.



4. Create a report to display employees’ last name and employee number along with their manager’s last name and manager number. Label the columns Employee, Emp#, Manager, and Mgr#, respectively. Place your SQL statement in a text file named



lab\_05\_04.sql.