Les05-Join tables

Chapter 5

Displaying data from

Multiple Tables

JOINS

Slide 176 in 10g

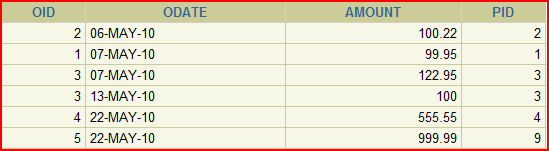
Alternate tables to use as samples or practice

These tables need to be loaded if you want to test the examples

CUSTOMERS



ORDERS



Here is the script to run. Just cut and paste it

drop table customers;

drop table orders;

CREATE TABLE CUSTOMERS (

PID NUMBER (1) NOT NULL,

PNAME VARCHAR2(20),

PEMAIL VARCHAR2(20)

);

LOAD THESE

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

Sometimes you need data from more than one table - possibly because the data has been normalized

**AFTER COMPLETING THIS LESSON**

🡪 Write SELECT statements to access data from

🡪 more than one table 🡪 equijoins etc …

🡪 Join table to itself 🡪 self-join

🡪 View data that does not normally meet a join condition

🡪 Use 🡪 Outer Joins

🡪 Generate a Cartesian Product of all rows to all rows from one or more tables

Obtaining data from Multiple Tables

PROBLEM: I want to know how all the programs at Seneca are doing. One of the things I want to look at is the

(a) program name and

(b) the number of students enrolled in the program.

Given these 2 tables: Give Program code and count of students in that program

Example Data

|  |  |
| --- | --- |
| ProgramCode | ProgramDescription |
| ACC | Accounting |
| BUS | Business |
| CPA | Computer Programming |
| MKT | Marketing |

Student Data

|  |  |  |
| --- | --- | --- |
| SID | NAME & OTHER DATA | PROGCODE |
| 1111 | first | ACC |
| 2222 | second | ACC |
| 3333 | third | BUS |
| 4444 | fourth | ACC |
| 5555 | fifth | MKT |

RESULT – if done manually

Accounting 3

Business 1

Computer Programming 0

Marketing 1

Example Data

|  |  |
| --- | --- |
| ProgramCode | ProgramDescription |
| ACC | Accounting |
| BUS | Business |
| CPA | Computer Programming |
| MKT | Marketing |

Student Data

|  |  |  |
| --- | --- | --- |
| SID | NAME & OTHER DATA | PROGCODE |
| 1111 | first | ACC |
| 2222 | second | ACC |
| 3333 | third | BUS |
| 4444 | fourth | ACC |
| 5555 | fifth | MKT |

**1 FIRST STEP data is found in more than 1 table – Find a common field and JOIN.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ACC | Accounting | 1111 | first | ACC |
| ACC | Accounting | 2222 | second | ACC |
| ACC | Accounting | 4444 | fourth | ACC |
| BUS | Business | 3333 | third | BUS |
| MKT | Marketing | 5555 | fifth | MKT |
|  |  |  |  |  |

**2 Then selecting rows to display if there is a where condition**

**3 and columns from the joined data based on the select**

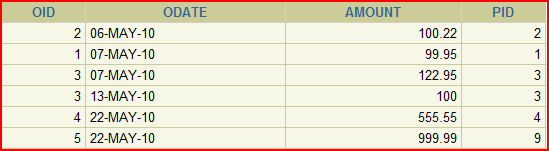
Another example of needing data from 2 tables

(these were the tables loaded at the start of the notes)

CUSTOMERS



ORDERS



**PROBLEM:** 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 to get the data from 2 or more tables.

The common field is PID (meaning Person ID of the Customer and the FK of PID in the order table)

**PROBLEM (re-stated)**

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

**SOLUTION:**

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

SELECT pname,

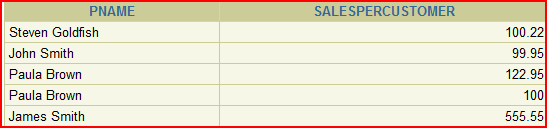
Amount AS "Sales Per Customer"

FROM Customers, Orders

WHERE Customers.pid = orders.pid;

The condition was to join on the common field

**RESULT:**



Natural Joins (personal suggestion – don’t use)

There are lots of suggestions that this should not be used. The reason later

• The NATURAL JOIN clause looks for all columns

in the 2 tables that share the same name

• It selects rows from the two tables that have

equal values in the matched rows

•If the columns having the same names have

**different data types, an error is returned**

SAMPLE: NEXT PAGE:

Sample joins – another example

Retrieve Department id, Department Name, Location id, city

The data is found in 2 tables 🡪 Department and Locations

Name Null Type

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

LOCATION\_ID NOT NULL NUMBER(4)

STREET\_ADDRESS VARCHAR2(40)

POSTAL\_CODE VARCHAR2(12)

CITY NOT NULL VARCHAR2(30)

STATE\_PROVINCE VARCHAR2(25)

COUNTRY\_ID CHAR(2)

Name Null Type

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

DEPARTMENT\_ID NOT NULL NUMBER(4)

DEPARTMENT\_NAME NOT NULL VARCHAR2(30)

MANAGER\_ID NUMBER(6)

LOCATION\_ID NUMBER(4

**EQUIJOIN – what you just did previous page**

**SELECT DEPARTMENT\_ID, DEPARTMENT\_NAME, D.LOCATION\_ID, CITY**

**FROM DEPARTMENTS D, LOCATIONS L**

**WHERE D.LOCATION\_ID = L.LOCATION\_ID**

NATURAL

**SELECT DEPARTMENT\_ID, DEPARTMENT\_NAME, LOCATION\_ID, CITY**

**FROM DEPARTMENTS**

**NATURAL JOIN LOCATIONS;**

## 🡪🡪🡪 NB It knows to look for the common named column

* Cannot use a qualifier on location\_id in the select as it is the common column

# Another example:

**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**

MUST HAVE A COMMON NAME

Do the same with an EQUIJOIN

**NATURAL JOIN locations**

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

NOTE: EQUIJOIN = SIMPLE JOIN = INNER JOIN

Problem with Natural Join – not standard

1. The common column might be another set of columns with matching names
2. Since don’t know what was chosen to be joined sometimes the number of rows might be different

Natural join is like a “shortcut” to save from typing and has been around for a long time.

NOTE: from Wikepedia

Most experts agree that NATURAL JOINs are dangerous and therefore strongly discourage their use.[[3]](http://en.wikipedia.org/wiki/Join_(SQL)#cite_note-3) The danger comes from inadvertently adding a new column, named the same as another column in the other table. An existing natural join might then "naturally" use the new column for comparisons, making comparisons/matches using different criteria (from different columns) than before. Thus an existing query could produce different results, even though the data in the tables have not been changed, but only augmented.

Note that natural is not mentioned as a STANDARD

ANSI standard SQL specifies four types of JOIN: INNER, OUTER, LEFT, and RIGHT.

As a special case, a table (base table, [view](http://en.wikipedia.org/wiki/View_(database)), or joined table) can JOIN to itself in a *self-join*.

Creating Joins with the USING Clause

What if several columns have the same NAMES and not the same DATA TYPES?

**The NATURAL JOIN clause can be modified with the USING clause to specify the columns that should be used for an equijoin**

• Use the USING clause to match only one column

**when more than one column matches.**

• Do not use a table name or alias in the referenced columns

• The NATURAL JOIN and USING clauses are mutually exclusive

Sample:

**VALID:**

SELECT L.city, D.department\_name

FROM locations L JOIN departments D USING (location\_id)

WHERE location\_id = 1400;

**NOT VALID**: The referenced column (location\_id can not be qualified anywhere)

SELECT L.city, D.department\_name

FROM locations L JOIN departments D USING (location\_id)

WHERE D.location\_id = 1400;

ORA-25154: column part of USING clause cannot have qualifier

# Joining Column Names --- USING

Find the employee id, employee name, department ID, location ID

This data is found in the EMPLOYEES and DEPARTMENTS tables

SELECT EMPLOYEES.EMPLOYEE\_ID,

EMPLOYEES.LAST\_NAME,

NOTE: Oracle did not like it if bracket missing

DEPARTMENTS.LOCATION\_ID,

DEPARTMENT\_ID

FROM EMPLOYEES JOIN DEPARTMENTS

USING (DEPARTMENT\_ID);

Use table name prefix to qualify an ambiguous column name

🡪 one that is found in both tables

Use table prefixes to improve performance

If no join type mentioned the default is called an INNER JOIN

Use column aliases to distinguish columns that have identical names but are in different tables

DO NOT use aliases on columns used with a USING clause and listed elsewhere also

NOTE: … very wordy

Can use aliases just not on the one used in the using statement

Let the system choose which one

Creating Joins with the ON Clause

**🡪** The join condition for the natural join is basically

**an equijoin of all columns with the same name**

**🡪** Use the ON clause to specify arbitrary conditions

**or specify columns to join**

**🡪** The join condition lets you separate the join from other WHERE conditions

**🡪🡪🡪 The ON clause makes the code easy to understand**

SAMPLE:

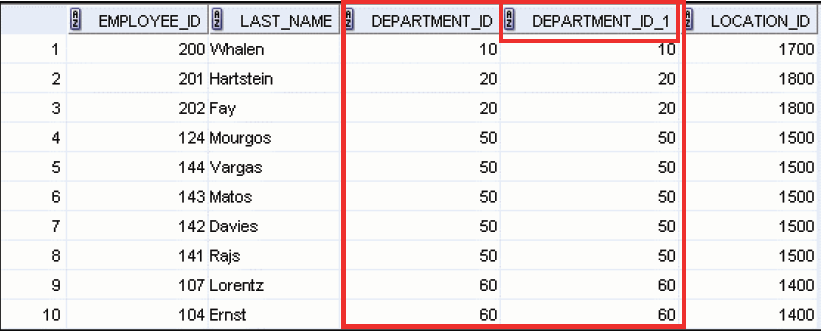
SELECT e.employee\_id, e.last\_name, e.department\_id,

d.department\_id, d.location\_id

Notice – no WHERE clause. Separates the 2 ideas of JOIN and WHERE

FROM employees e JOIN departments d

ON (e.department\_id = d.department\_id;



**NOTE:** Can also use ON for joins on different names

Three-Way Joins

Joining more than 2 tables

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

**FROM employees e**

**JOIN departments d**

Might be easier to read if indent

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

**JOIN locations l**

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

The order of the joins is from LEFT to RIGHT, or in this case first then second

This is mentioned so you will understand the condition in the ON must reference only columns in the tables being joined.

EQUIJOIN of same SQL

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

**FROM employees e, departments d**

Might be easier to read if indent

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

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

NEW PROBLEM to solve:

Find the last\_name of Lorentz's manager

Lorentz and the manager are all employees. There isn't a need for a manager table.

HOW TO DO IT

1) Find Lorentz in the employees table by looking up the name in the last\_names column

2) Find the manager number on the same row 🡪 103

3) Use the manager number to search back through the employee table to find a match for employee 103

You are looking in the same table twice

TABLE: Employees

EMPLOYEE\_ID LAST\_NAME MANAGER\_ID

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

100 King

101 Kochhar 100

102 De Haan 100

103 Hunold 102

104 Ernst 103

107 Lorentz 103

124 Mourgos 100

141 Rajs 124

142 Davies 124

143 Matos 124

144 Vargas 124

149 Zlotkey 100

174 Abel 149

176 Taylor 149

178 Grant 149

200 Whalen 101

201 Hartstein 100

202 Fay 201

205 Higgins 101

206 Gietz 205

20 rows selected

SELF JOIN(p190-slide)

USED TO SOLVE ABOVE PROBLEM:

**SELECT e.last\_name emp,**

**m.last\_name mgr**

**FROM employees e JOIN employees m**

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

**WHERE e.last\_name like 'Lorentz';**

The managers id in the employees E table matches the employee id in the managers table

SELF JOIN can have conditions

Show only those with manager 149

SELECT e.employee\_id, e.last\_name, e.department\_id,

d.department\_id, d.location\_id

FROM employees e JOIN departments d

ON (e.department\_id = d.department\_id)

AND e.manager\_id = 149;

Alternately you can use a WHERE clause

SELECT e.employee\_id, e.last\_name, e.department\_id,

d.department\_id, d.location\_id

FROM employees e JOIN departments d

ON (e.department\_id = d.department\_id)

WHERE e.manager\_id = 149;

This is what it meant when it separates the join from the where condition

NOTE: Equijoins are based on equality or = signs

# SELF JOIN PROBLEM: -- another example

Display Managers Last Name and the employees last name working for that manager.

**USING EQUIJOIN**

Select M.Last\_Name As Manager, W.last\_name AS Worker

From Employees M , Employees W

WHERE W.manager\_id = M.employee\_id (where the worker’s manager id is equal to the managers employee id)

Order By 1;

Result:

MANAGER WORKER

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

De Haan Hunold

Hartstein Fay

Higgins Gietz

Hunold Ernst

Hunold Lorentz

King De Haan

King Kochhar

King Hartstein

King Zlotkey

King Mourgos

Kochhar Higgins

Kochhar Whalen

Etc.…. 19 rows

USING ON method

Select M.Last\_Name As Manager, W.Last\_Name As Worker

From Employees M Join Employees W

ON W.Manager\_Id = M.Employee\_Id

order by M.last\_name

NOTE: The ON is like the WHERE

INNER joins

The most common join.

**Explicitly** defined inner join

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

**FROM employees INNER JOIN departments**

**ON employees.Department\_ID = departments.Department\_ID;**

EMPLOYEE\_ID LAST\_NAME DEPARTMENT\_NAME

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

200 Whalen Administration

201 Hartstein Marketing

202 Fay Marketing

124 Mourgos Shipping

141 Rajs Shipping

142 Davies Shipping

143 Matos Shipping

144 Vargas Shipping

103 Hunold IT

104 Ernst IT

107 Lorentz IT

149 Zlotkey Sales

174 Abel Sales

176 Taylor Sales

100 King Executive

101 Kochhar Executive

102 De Haan Executive

205 Higgins Accounting

206 Gietz Accounting

Note this looks very like this example which is the implicitly defined join

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

**FROM employees, departments**

**WHERE employees.Department\_ID = departments.Department\_ID;**

🡺🡺 The ON tends to make it more readable

Introduce another problem with a JOIN

Another look at the other set of tables

|  |  |
| --- | --- |
| CUSTOMERS | ORDERS |

**PROBLEM**

**PROVIDE A LIST OF CUSTOMERS AND THEIR SALES AMOUNTS –** shown before and just repeated here

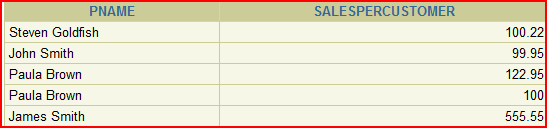
SELECT pname,

Amount AS SalesPerCustomer

FROM Customers, Orders

WHERE Customers.pid = orders.pid

The condition was to join on the common field



PROBLEM:

Notice Paula Brown has 2 orders

|  |  |
| --- | --- |
| CUSTOMERS | ORDERS |

Show only total amount from the customer

🡪 Need to use a SUM function

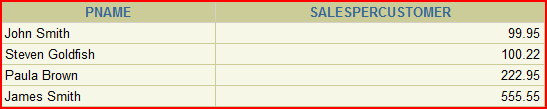
**SELECT pname, SUM(Amount) AS TotalSales**

**FROM Customers JOIN Orders**

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

**GROUP BY PNAME** 🡨 MUST HAVE THIS

SINGLE AND GROUP FUNCTIONS ON SELECT



??? This does not answer a question such as …

**List all customers** and what orders they have placed.

The join is based on finding a value in the joining columns. If no order has been placed then no data will show, but you want all customers

* 🡺🡺 🡺… leads to other types of joins ---

OUTER JOIN

|  |  |
| --- | --- |
| CUSTOMERS | ORDERS |

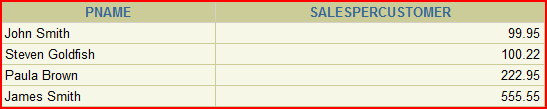
To find what customers ordered we used this SQL

**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



PROBLEM:

Uncle Joe, PID 5, does not show because there is no order for PID 5

(Note look at PID and not OID)

🡪 🡪🡪 Management wants ALL customers and the resulting sales totals.

LEADS TO ….

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

**The problem was to display ALL customers and their sales including the customers with no sales**

To solve this requires an OUTER JOIN

3 types of OUTER JOINS

🡪 LEFT

🡪 RIGHT

🡪 FULL

SQL:1999

1) Joins of 2 tables that return only matching rows 🡪 INNER JOIN

2) Joins between 2 tables that return

a) result of INNER join

b) any unmatched rows from the left (or right) tables

🡪 called an OUTER JOIN

3) Joins between2 tables that returns the result of

a) an INNER join and

b) all results of both left and right non-matching rows

🡪 called a FULL OUTER JOIN

LEFT JOIN

|  |  |
| --- | --- |
| CUSTOMERS | ORDERS |

SELECT pname,

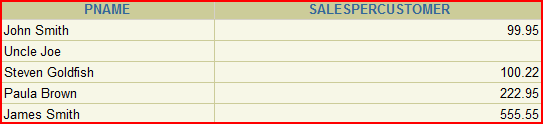
SUM(Amount) AS SalesPerCustomer

FROM Customers **LEFT** JOIN Orders

ON Customers.pid = orders.pid

GROUP BY PNAME

**Now there are 5**



**THIS SOLVES 🡪 Display ALL customers and their sales**

# RIGHT JOIN

|  |  |
| --- | --- |
| CUSTOMERS | ORDERS |

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

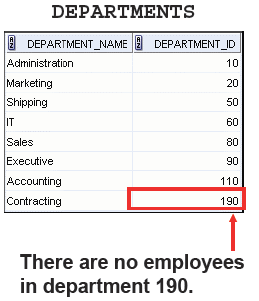
But it does allow us to demonstrate a RIGHT JOIN

MORE OUTER EXAMPLES

Using EMPLOYEES and DEPARTMENTS

Show last\_name, department ID, department\_name

Notice Contracting has no employees and Grant has no department



SELECT E.LAST\_NAME, D.DEPARTMENT\_ID, D.DEPARTMENT\_NAME

FROM EMPLOYEES E FULL OUTER JOIN DEPARTMENTS D

ON (E.DEPARTMENT\_ID = D.DEPARTMENT\_ID);

LAST\_NAME DEPARTMENT\_ID DEPARTMENT\_NAM

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

King 90 Executive

Kochhar 90 Executive

De Haan 90 Executive

Hunold 60 IT

Ernst 60 IT

Lorentz 60 IT

Mourgos 50 Shipping

Rajs 50 Shipping

Davies 50 Shipping

Matos 50 Shipping

Vargas 50 Shipping

Zlotkey 80 Sales

Abel 80 Sales

Taylor 80 Sales

Grant

Whalen 10 Administration

Hartstein 20 Marketing

Fay 20 Marketing

Higgins 110 Accounting

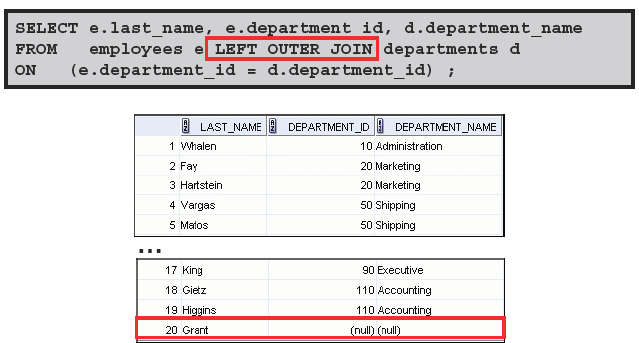
Gietz 110 Accounting

190 Contracting

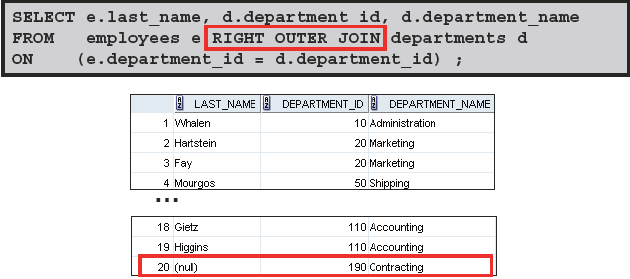
21 rows selected.

3 OUTER JOINS

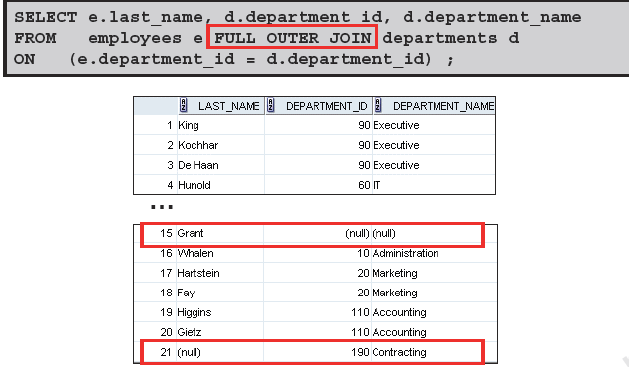
LEFT



RIGHT



FULL



NON-EQUIJOINS

PROBLEM:

Find the last name, the salary and what salary grade letter applies to their salary

Look at the 2 tables JOB\_GRADES and EMPLOYEES

|  |  |
| --- | --- |
| GRADE LOWEST\_SAL HIGHEST\_SAL  - ---------- -----------  A 1000 2999  B 3000 5999  C 6000 9999  D 10000 14999  E 15000 24999  F 25000 40000 | LAST\_NAME SALARY  ------------------------- ----------  King 24000  Kochhar 17000  De Haan 17000  Hunold 9000  Ernst 6000  Lorentz 4200  Mourgos 5800  Rajs 3500  Davies 3100  Matos 2600  … |

NOTE:

In the above tables salary for employee

King has a salary of 24000. It falls in the fifth JOB\_GRADE which is an E

Ernst with salary of 6000 will have a GRADE\_LEVEL of C

Non-equijoins simply mean a join on other than equals or equality

To do this we use an operator other than equals.

SELECT E.LAST\_NAME, E.SALARY, J.GRADE

FROM EMPLOYEES E JOIN JOB\_GRADES J

ON E.SALARY

BETWEEN J.LOWEST\_SAL AND J.HIGHEST\_SAL;

LAST\_NAME SALARY G

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

Vargas 2500 A

Matos 2600 A

Davies 3100 B

Rajs 3500 B

Lorentz 4200 B

Whalen 4400 B

Mourgos 5800 B

Ernst 6000 C

Fay 6000 C

Etc ….