Ellison styled his estimated $70 million Woodside, California, estate after feudal Japanese architecture, complete with a man-made 2.3-acre lake and an extensive seismic retrofit . In 2004 and 2005, Ellison purchased more than 12 properties in Malibu, California, worth more than $180 million. The $65 million Ellison spent on five contiguous lots on Malibu's Carbon Beach was the most costly residential transaction in United States history until Ron Perelman sold his Palm Beach, Florida compound for $70 million later that same year.His entertainment system cost $1 million, and includes a rock concert-sized video projector at one end of a drained swimming pool, using the gaping hole as a giant subwoofer

Chapter 9(SUBQUERIES)

"A friend is one who knows us, but loves us anyway."

Example 9a

```
DROP TABLE patient_disease;
DROP TABLE patient;
CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      Gender            CHAR,
      DOB         DATE,
      salary            NUMBER ,
               city                      VARCHAR2(20),
               state                   VARCHAR2(20)
);

INSERT INTO patient values (111,'john','Doe','m','11-FEB-1978',25000,
'Davis','CA');
INSERT INTO patient values (112,'john','Smith','m','01-MAR-1981',40000,
'Davis','CA');
INSERT INTO patient values (113,'jill','Crane','m','12-APR-
1999',NULL,'Reno','NV');
INSERT INTO patient values (114,'billy','Bob','f','05-MAY-1985',60000,'Las
Vegas','NV');
INSERT INTO patient values (115,'dove','Grime','f','04-JUN-
1960',20000,'Sacramento','CA');

DROP TABLE disease;
CREATE  TABLE disease
 (
      disease_id  NUMBER PRIMARY KEY,
      disease_desc      VARCHAR2(20)
);
INSERT INTO disease VALUES (11,'Cancer');
INSERT INTO disease VALUES (22,'Malaria');
INSERT INTO disease VALUES (33,'Flu');

CREATE  TABLE patient_disease
 (
               Patient_id          NUMBER REFERENCES patient,
      disease_id  NUMBER REFERENCES disease,
               PRIMARY KEY (patient_id, disease_id)
);
INSERT INTO patient_disease VALUES (111,11);
INSERT INTO patient_disease VALUES (111,22);
INSERT INTO patient_disease VALUES (113,11);
```

Patient

Patient_id	Fname	Lname	…
111	john	Doe	...
114	billy	Bob	...
112	john	Smith	...
113	jill	Crane	...

Patient_Disease

Patient_id	Disease_id
111	11
111	22
113	11

Disease

Disease_id	Disease_desc
11	Cancer
22	Malaria
33	Flu

A subquery is a SELECT statement used in another SQL command. Any type of action you can perform with a SELECT statement (such as filtering rows, filter-ing columns, and calculating aggregate amounts) can be performed when creating a table with a subquery.This first query is the subquery. The subquery's results are passed as input to the outer query (also called the parent query). The outer query incorporates this value into its calculations to determine the final output.

You can nest subqueries inside the FROM, WHERE, or HAVING clauses of other subqueries. In Oracle , subqueries in a WHERE clause can be nested to a depth of 255 subqueries, and there's no depth limit when subqueries are nested in a FROM clause. When nesting sub-queries, you might want to use the following strategy:

Determine exactly what you're trying to find— in other words, the goal of the query.
Write the innermost subquery first.

Next, look at the value you can pass to the outer query. If it isn't the value the outer query needs (for example, it references the wrong column), analyze how you need to convert the data to get the correct rows. If necessary, use another subquery between the outer query and the nested subquery.

Keep the following rules in mind when working with any type of subquery: A subquery must be a complete query in itself— in other words, it must have at least a SELECT and a FROM clause.

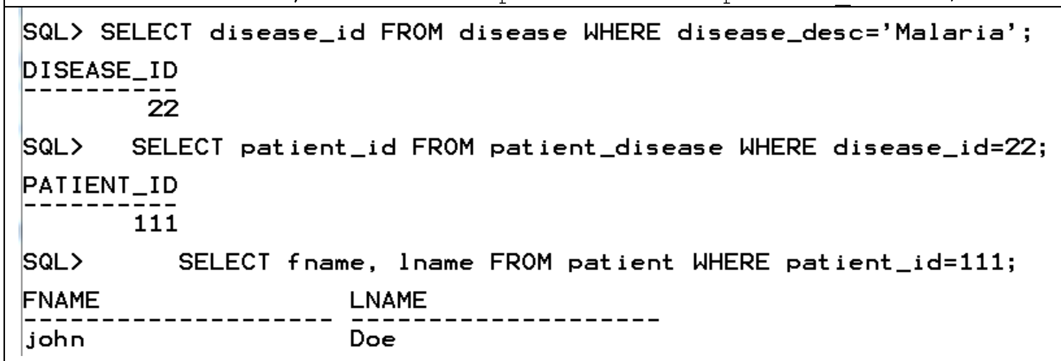
A subquery, except one in the FROM clause, can't have an ORDER BY clause. If you need to display output in a specific order, include an ORDER BY clause as the outer query's last clause.

A subquery must be enclosed in parentheses to separate it from the outer query.

If you place a subquery in the outer query's WHERE or HAVING clause, you can do so only on the right side of the comparison operator.

Example 9b (using separate queries)

```
--In this example, we want to know the names of all the people who are infected with Malaria.
--Using separate queries, we first have to find out what the disease_id is for malaria which
--is in the disease table. We then use the disease_id to find out which patient_id(s) have malaria.
--For this we need to go to the patient_disease table. Finally, we take the list of patient_ids and
--feed it into the patient table to get their names. Based on what we know so far, we would write
--three separate queries but in the next example, we will connect them into a single query. This
--would be done through the subquery mechanism.
SELECT disease_id FROM disease WHERE disease_desc='Malaria';
  SELECT patient_id FROM patient_disease WHERE disease_id=22;
     SELECT fname, lname FROM patient WHERE patient_id=111;
```

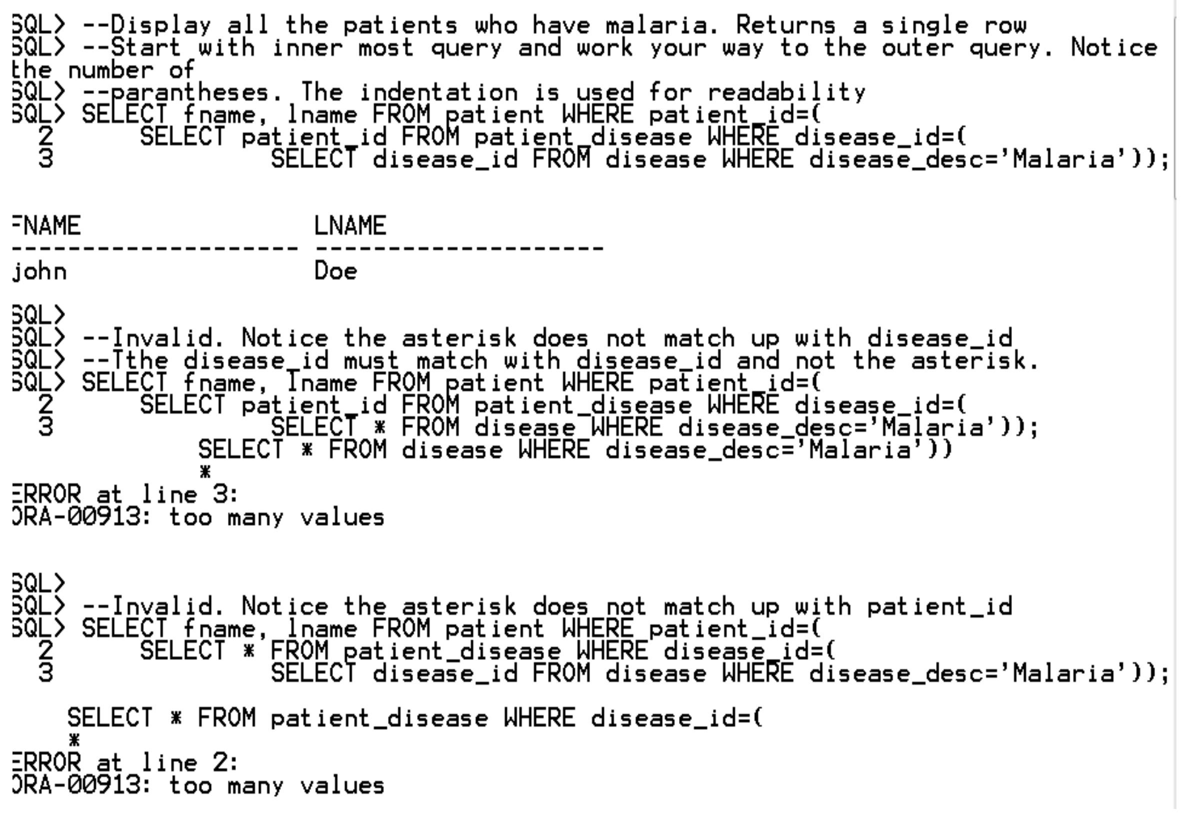


```
SQL> SELECT disease_id FROM disease WHERE disease_desc='Malaria';

DISEASE_ID
----------
        22

SQL>   SELECT patient_id FROM patient_disease WHERE disease_id=22;

PATIENT_ID
----------
       111

SQL>      SELECT fname, lname FROM patient WHERE patient_id=111;

FNAME                    LNAME
-------------------- --------------------
john                 Doe
```

Example 9c (Using subqueries-one single row)

A single- row subquery can return only one row of results consisting of only one column to the outer query. A single- row subquery can also be nested in the outer query's SELECT clause.

```
--Display all the patients who have malaria. Returns a single row.
--Start with inner most query and work your way to the outer query. Notice the number of
--parantheses. The indentation is used for readability.
SELECT fname, lname FROM patient WHERE patient_id=(
    SELECT patient_id FROM patient_disease WHERE disease_id=(
            SELECT disease_id FROM disease WHERE
disease_desc='Malaria'));


--Invalid. Notice the asterisk does not match up with disease_id.
--The disease_id must match with disease_id and not the asterisk.
SELECT fname, lname FROM patient WHERE patient_id=(
    SELECT patient_id FROM patient_disease WHERE disease_id=(
             SELECT * FROM disease WHERE disease_desc='Malaria'));


--Invalid. Notice the asterisk does not match up with patient_id.
SELECT fname, lname FROM patient WHERE patient_id=(
    SELECT * FROM patient_disease WHERE disease_id=(
            SELECT disease_id FROM disease WHERE
disease_desc='Malaria'));
```



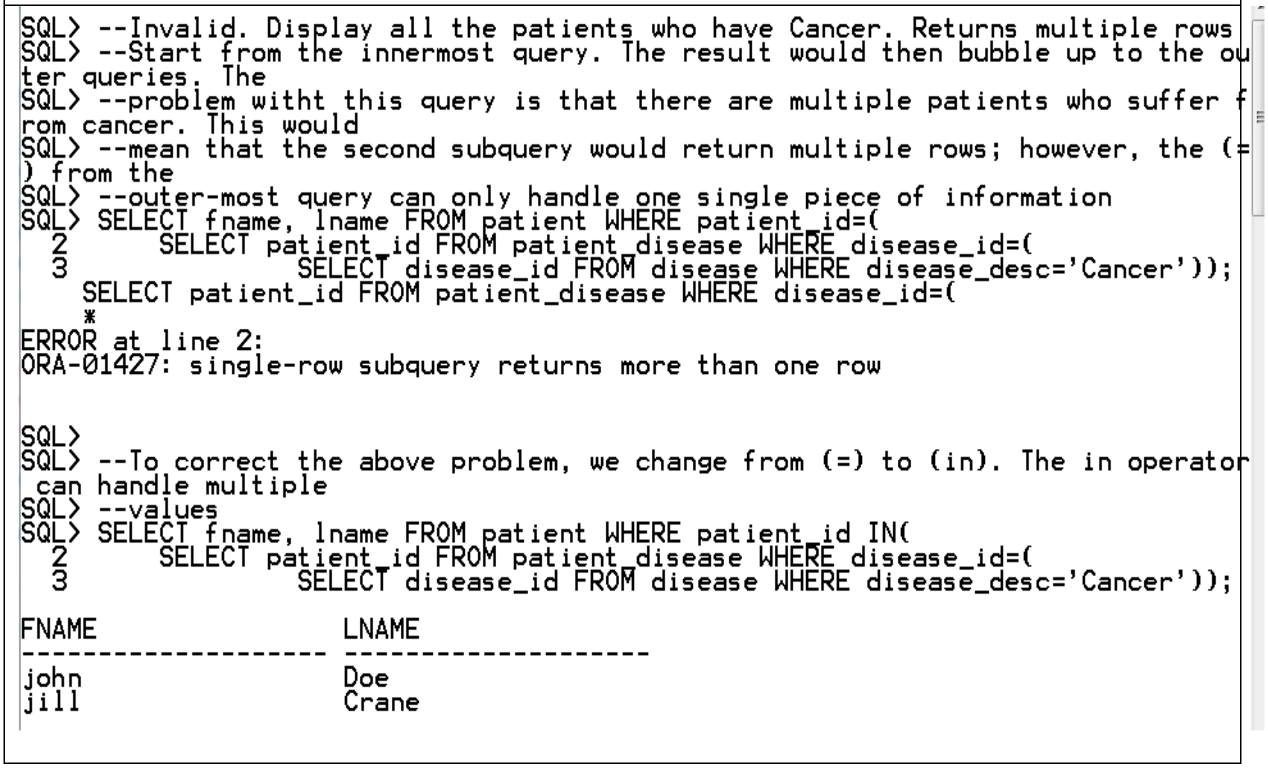
```
SQL> --Display all the patients who have malaria. Returns a single row
SQL> --Start with inner most query and work your way to the outer query. Notice
the number of
SQL> --parantheses. The indentation is used for readability
SQL> SELECT fname, lname FROM patient WHERE patient_id=(
  2      SELECT patient_id FROM patient_disease WHERE disease_id=(
  3              SELECT disease_id FROM disease WHERE disease_desc='Malaria'));

FNAME                LNAME
-------------------- --------------------
john                 Doe

SQL>
SQL> --Invalid. Notice the asterisk does not match up with disease_id
SQL> --Tthe disease_id must match with disease_id and not the asterisk.
SQL> SELECT fname, lname FROM patient WHERE patient_id=(
  2      SELECT patient_id FROM patient_disease WHERE disease_id=(
  3              SELECT * FROM disease WHERE disease_desc='Malaria'));
             SELECT * FROM disease WHERE disease_desc='Malaria'))
             *
ERROR at line 3:
ORA-00913: too many values


SQL>
SQL> --Invalid. Notice the asterisk does not match up with patient_id
SQL> SELECT fname, lname FROM patient WHERE patient_id=(
  2      SELECT * FROM patient_disease WHERE disease_id=(
  3              SELECT disease_id FROM disease WHERE disease_desc='Malaria'));

    SELECT * FROM patient_disease WHERE disease_id=(
    *
ERROR at line 2:
ORA-00913: too many values
```

Example 9d (Multiple rows)

Multiple- row subqueries are nested queries that can return more than one row of results to the parent query. The main rule to keep in mind when working with multiple- row subqueries is that you must use multiple- row operators. If a single- row operator is used with a subquery that returns more than one row of results, Oracle returns an error message. Valid multiple- row operators include IN, ALL, and ANY must be used. Of the three, the IN Operator is used most often.

```
--Invalid. Display all the patients who have Cancer. Returns multiple rows.
```

```
--Start from the inner-most query. The result would then bubble up to the outer queries. The
--problem with this query is that there are multiple patients who suffer from cancer. This
--would mean that the second subquery would return multiple rows; however, the (=) from the outer
--most query can only handle one single piece of information.
SELECT fname, lname FROM patient WHERE patient_id=(
    SELECT patient_id FROM patient_disease WHERE disease_id=(
             SELECT disease_id FROM disease WHERE disease_desc='Cancer'));


--To correct the above problem, we change from (=) to (in). The in operator can handle multiple values.
SELECT fname, lname FROM patient WHERE patient_id IN(
    SELECT patient_id FROM patient_disease WHERE disease_id=(
             SELECT disease_id FROM disease WHERE disease_desc='Cancer'));
```

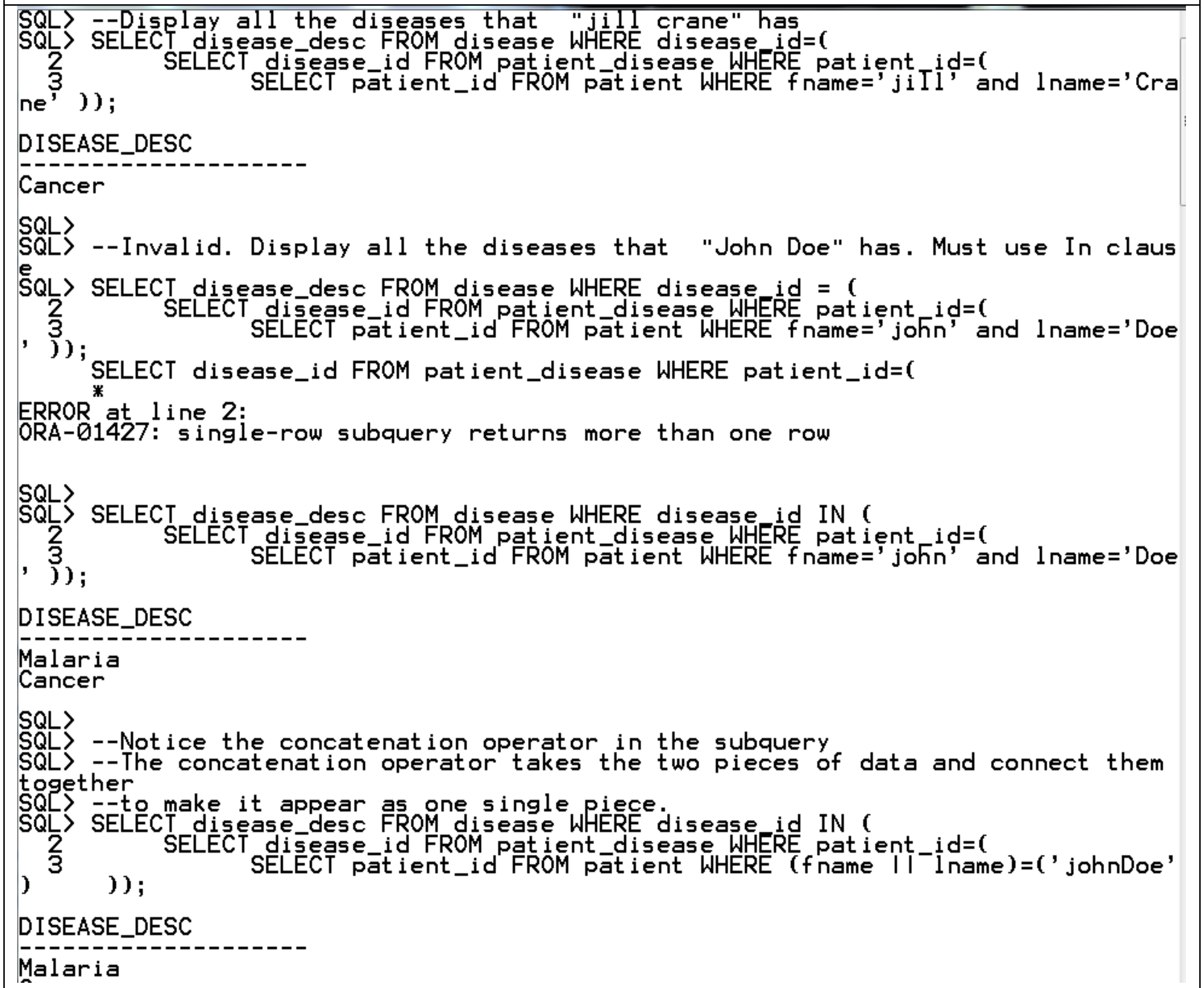


```
SQL> --Invalid. Display all the patients who have Cancer. Returns multiple rows
SQL> --Start from the innermost query. The result would then bubble up to the ou
ter queries. The
SQL> --problem witht this query is that there are multiple patients who suffer f
rom cancer. This would
SQL> --mean that the second subquery would return multiple rows; however, the (=
) from the
SQL> --outer-most query can only handle one single piece of information
SQL> SELECT fname, lname FROM patient WHERE patient_id=(
  2      SELECT patient_id FROM patient_disease WHERE disease_id=(
  3               SELECT disease_id FROM disease WHERE disease_desc='Cancer'));
    SELECT patient_id FROM patient_disease WHERE disease_id=(
    *
ERROR at line 2:
ORA-01427: single-row subquery returns more than one row


SQL>
SQL> --To correct the above problem, we change from (=) to (in). The in operator
 can handle multiple
SQL> --values
SQL> SELECT fname, lname FROM patient WHERE patient_id IN(
  2      SELECT patient_id FROM patient_disease WHERE disease_id=(
  3               SELECT disease_id FROM disease WHERE disease_desc='Cancer'));

FNAME                LNAME
-------------------- --------------------
john                 Doe
jill                 Crane
```

Example 9e (Single and multiple rows)

```
--Display all the diseases that ”jill crane” has.
SELECT disease_desc FROM disease WHERE disease_id=(
 SELECT disease_id FROM patient_disease WHERE patient_id=(
   SELECT patient_id FROM patient WHERE fname=’jill’ and lname=’Crane' ));




--Invalid. Display all the diseases that ”John Doe” has. Must use in clause.
SELECT disease_desc FROM disease WHERE disease_id = (
     SELECT disease_id FROM patient_disease WHERE patient_id=(
       SELECT patient_id FROM patient WHERE fname=’john’ and lname=’Doe' ));
SELECT disease_desc FROM disease WHERE disease_id IN (
     SELECT disease_id FROM patient_disease WHERE patient_id=(
      SELECT patient_id FROM patient WHERE fname=’john’ and lname=’Doe' ));
```

```
--Notice the concatenation operator in the subquery.
--The concatenation operator takes the two pieces of data and connect them together
--to make it appear as one single piece.
SELECT disease_desc FROM disease WHERE disease_id IN (
  SELECT disease_id FROM patient_disease WHERE patient_id=(
    SELECT patient_id FROM patient WHERE (fname || lname)=('johnDoe')));
```



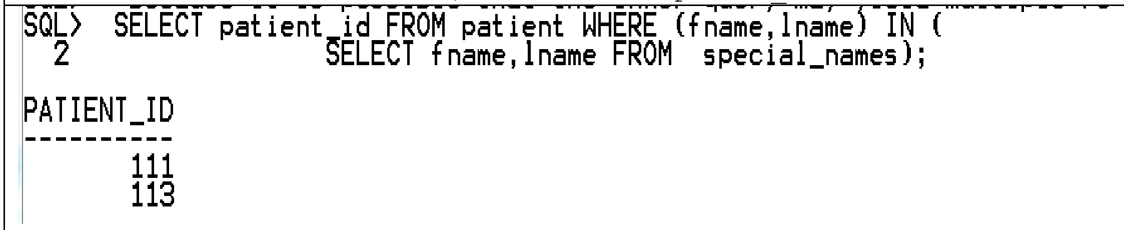
```
SQL> --Display all the diseases that  "jill crane" has
SQL> SELECT disease_desc FROM disease WHERE disease_id=(
  2         SELECT disease_id FROM patient_disease WHERE patient_id=(
  3               SELECT patient_id FROM patient WHERE fname='jill' and lname='Cra
ne' ));

DISEASE_DESC
--------------------
Cancer

SQL>
SQL> --Invalid. Display all the diseases that  "John Doe" has. Must use In claus
e
SQL> SELECT disease_desc FROM disease WHERE disease_id = (
  2         SELECT disease_id FROM patient_disease WHERE patient_id=(
  3               SELECT patient_id FROM patient WHERE fname='john' and lname='Doe
' ));
     SELECT disease_id FROM patient_disease WHERE patient_id=(
     *
ERROR at line 2:
ORA-01427: single-row subquery returns more than one row


SQL>
SQL> SELECT disease_desc FROM disease WHERE disease_id IN (
  2         SELECT disease_id FROM patient_disease WHERE patient_id=(
  3               SELECT patient_id FROM patient WHERE fname='john' and lname='Doe
' ));

DISEASE_DESC
--------------------
Malaria
Cancer

SQL>
SQL> --Notice the concatenation operator in the subquery
SQL> --The concatenation operator takes the two pieces of data and connect them
together
SQL> --to make it appear as one single piece.
SQL> SELECT disease_desc FROM disease WHERE disease_id IN (
  2         SELECT disease_id FROM patient_disease WHERE patient_id=(
  3               SELECT patient_id FROM patient WHERE (fname || lname)=('johnDoe'
)     ));

DISEASE_DESC
--------------------
Malaria
```

Example 9f (Multiple column subquery)

Multiple- column subquery returns more than one column to the outer query. The syntax of the outer WHERE clause is WHERE (columnname, columnname, ...) IN subquery.

Keep these rules in mind: Because the WHERE clause contains more than one column name, the column list must be enclosed in parentheses. Column names listed in the WHERE clause must be in the same order as they're listed in the subquery's SELECT clause.

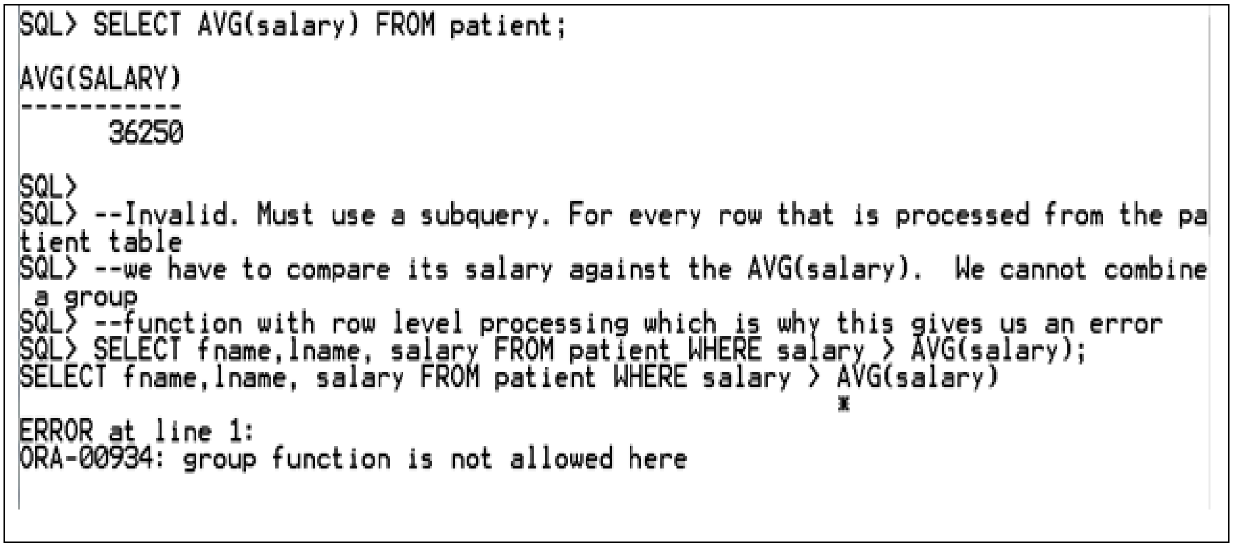
```
DROP TABLE special_names;
CREATE TABLE special_names (fname VARCHAR2(20), lname VARCHAR2(30));
INSERT INTO special_names VALUES ('john','Doe');
INSERT INTO special_names VALUES ('jill','Crane');

--Notice that fname and lname are enclosed in parantheses in the outer query. It works
--like the concatenation operator in that fname and lname become a single piece of data
--which are compared against fname and lname in the inner query.  The (IN) operator is used
--because it is possible that the inner query may yield multiple rows.
 SELECT patient_id FROM patient WHERE (fname,lname) IN (
                    SELECT fname,lname FROM  special_names);
```



```
SQL>  SELECT patient_id FROM patient WHERE (fname,lname) IN (
  2                     SELECT fname,lname FROM  special_names);

PATIENT_ID
----------
       111
       113
```

Example 9g (Group functions and subqueries)

```
SELECT AVG(salary) FROM patient;

--Invalid. Must use a subquery. For every row that is processed from the patient table
--we have to compare its salary against the AVG(salary).  We cannot combine a group
--function with row level processing which is why this gives us an error.
SELECT fname,lname, salary FROM patient WHERE salary > AVG(salary);
```



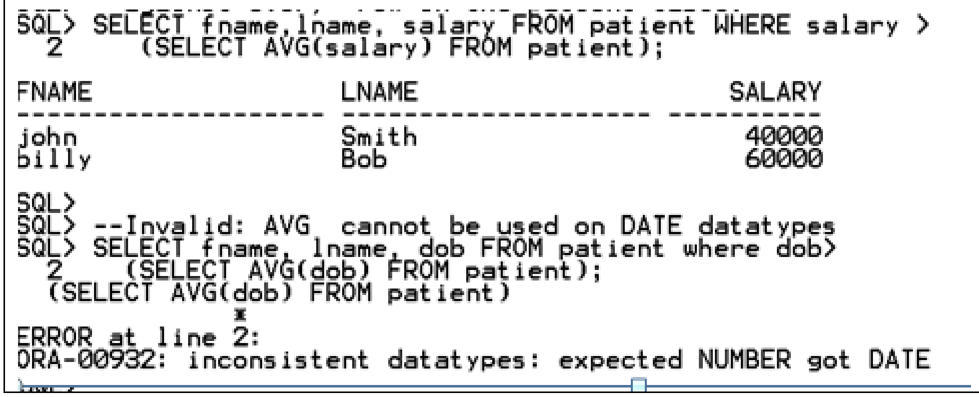
```
SQL> SELECT AVG(salary) FROM patient;

AVG(SALARY)
-----------
      36250

SQL>
SQL> --Invalid. Must use a subquery. For every row that is processed from the pa
tient table
SQL> --we have to compare its salary against the AVG(salary).  We cannot combine
 a group
SQL> --function with row level processing which is why this gives us an error
SQL> SELECT fname,lname, salary FROM patient WHERE salary > AVG(salary);
SELECT fname,lname, salary FROM patient WHERE salary > AVG(salary)
                                                       *
ERROR at line 1:
ORA-00934: group function is not allowed here
```

--In this case, the inner query will be executed which comes up with a single number.
--That single number will be fed to the outer query which can be used to compare
--against every row in the patient table.

```
SELECT fname,lname, salary FROM patient WHERE salary >
    (SELECT AVG(salary) FROM patient);
```

--Invalid: AVG cannot be used on DATE datatypes

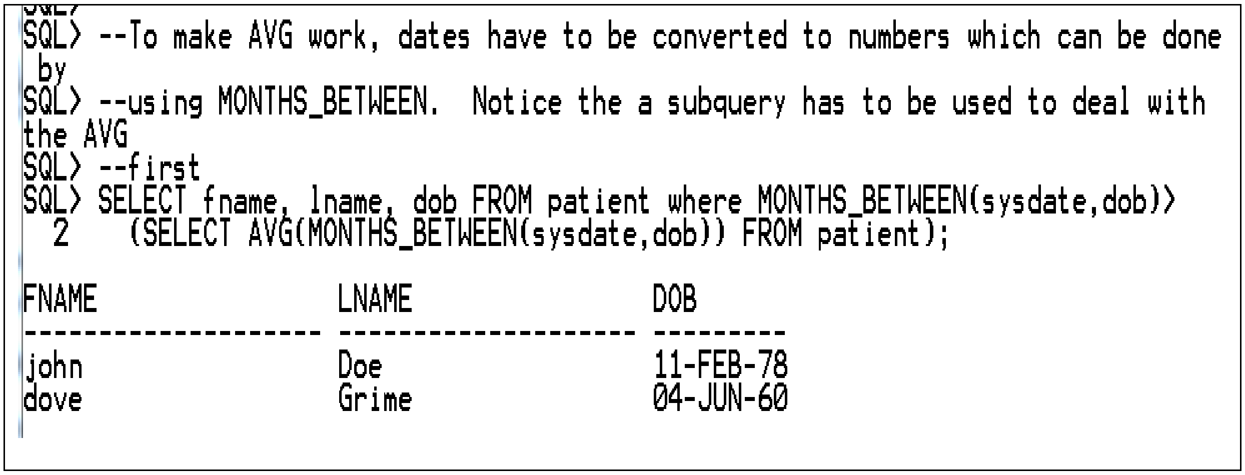
```
SELECT fname, lname, DOB FROM patient where DOB>
   (SELECT AVG(DOB) FROM patient);
```



```
SQL> SELECT fname,lname, salary FROM patient WHERE salary >
  2      (SELECT AVG(salary) FROM patient);

FNAME                LNAME                    SALARY
-------------------- -------------------- ----------
john                 Smith                     40000
billy                Bob                       60000

SQL>
SQL> --Invalid: AVG  cannot be used on DATE datatypes
SQL> SELECT fname, lname, dob FROM patient where dob>
  2     (SELECT AVG(dob) FROM patient);
   (SELECT AVG(dob) FROM patient)
               *
ERROR at line 2:
ORA-00932: inconsistent datatypes: expected NUMBER got DATE
```

--To make AVG work, dates have to be converted to numbers which can be done by using
--MONTHS_BETWEEN. Notice that a subquery has to be used to deal with the AVG first.

```
SELECT fname, lname, DOB FROM patient where MONTHS_BETWEEN(sysdate,DOB)>
   (SELECT AVG(MONTHS_BETWEEN(sysdate,DOB)) FROM patient);
```

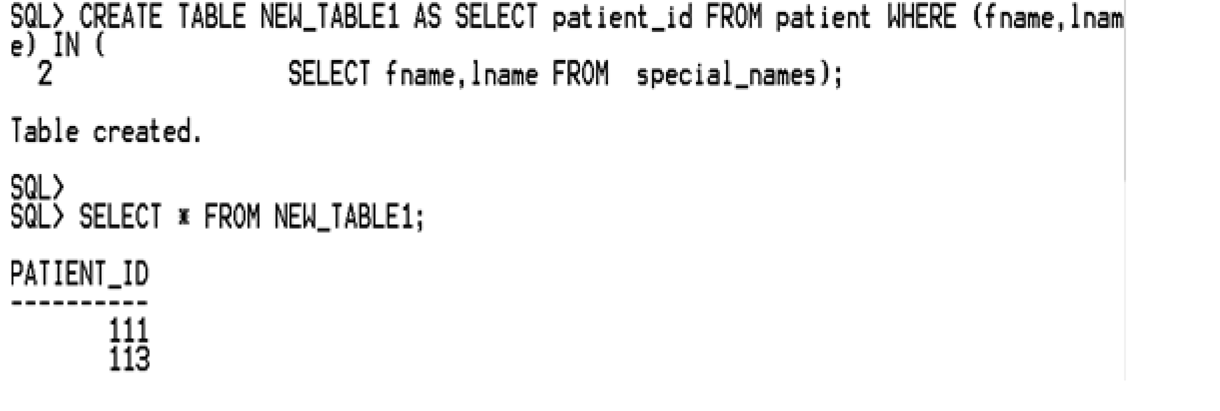


```
SQL> --To make AVG work, dates have to be converted to numbers which can be done
 by
SQL> --using MONTHS_BETWEEN.  Notice the a subquery has to be used to deal with
the AVG
SQL> --first
SQL> SELECT fname, lname, dob FROM patient where MONTHS_BETWEEN(sysdate,dob)>
  2    (SELECT AVG(MONTHS_BETWEEN(sysdate,dob)) FROM patient);

FNAME                LNAME                DOB
-------------------- -------------------- ---------
john                 Doe                  11-FEB-78
dove                 Grime                04-JUN-60
```

Example 9h (Create table and subqueries)

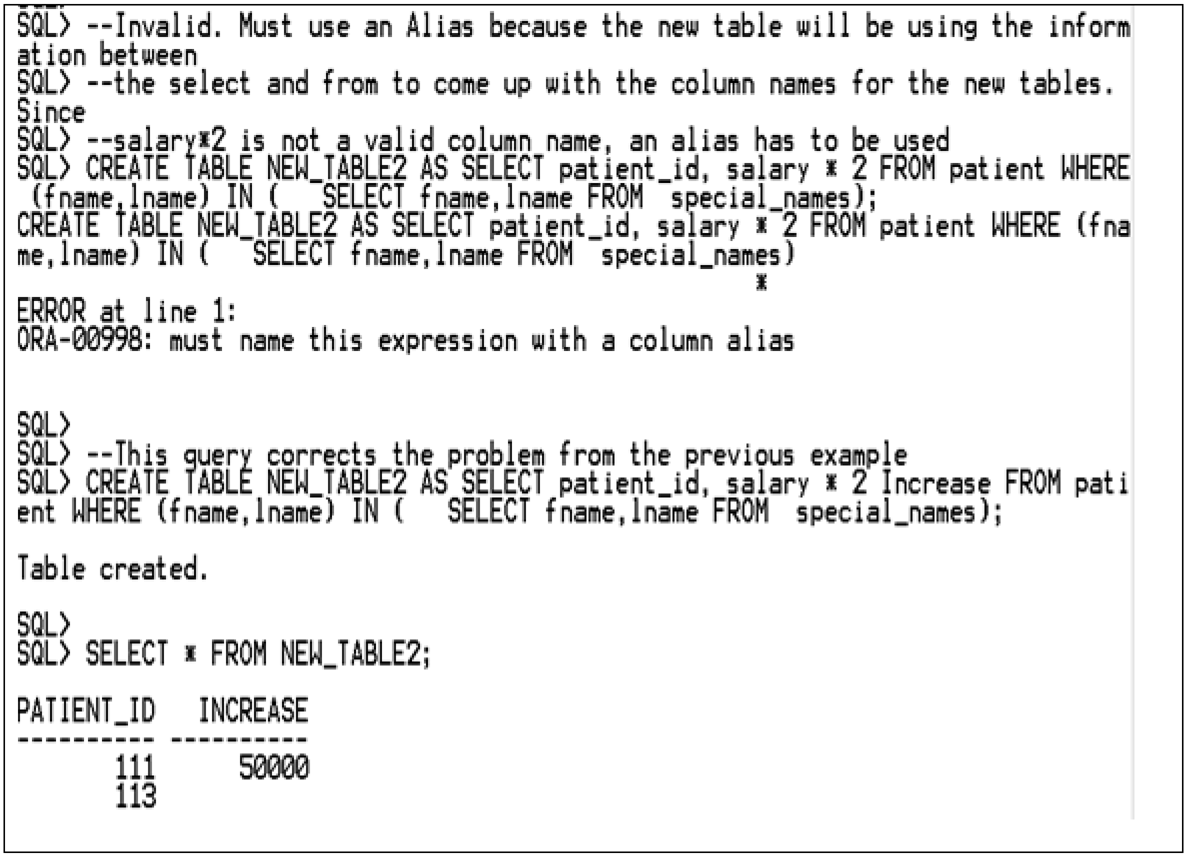
You can also perform CREATE TABLE AS by using subqueries.

```
--Instead of displaying the information on to the screen, it can be fed into a brand new table.
CREATE TABLE NEW_TABLE1 AS SELECT patient_id FROM patient WHERE
(fname,lname) IN (SELECT fname,lname FROM  special_names);

SELECT * FROM NEW_TABLE1;
```



```
SQL> CREATE TABLE NEW_TABLE1 AS SELECT patient_id FROM patient WHERE (fname,lnam
e) IN (
  2                     SELECT fname,lname FROM  special_names);

Table created.

SQL>
SQL> SELECT * FROM NEW_TABLE1;

PATIENT_ID
----------
       111
       113
```

```
--Invalid. Must use an alias because the new table will be using the information between
--the select and from to come up with the column names for the new tables. Since
--salary*2 is not a valid column name, an alias has to be used.
CREATE TABLE NEW_TABLE2 AS SELECT patient_id, salary * 2 FROM patient
WHERE (fname,lname) IN (SELECT fname,lname FROM  special_names);


--This query corrects the problem from the previous example.
CREATE TABLE NEW_TABLE2 AS SELECT patient_id, salary * 2 Increase FROM
patient WHERE (fname,lname) IN (   SELECT fname,lname FROM
special_names);


SELECT * FROM NEW_TABLE2;
```

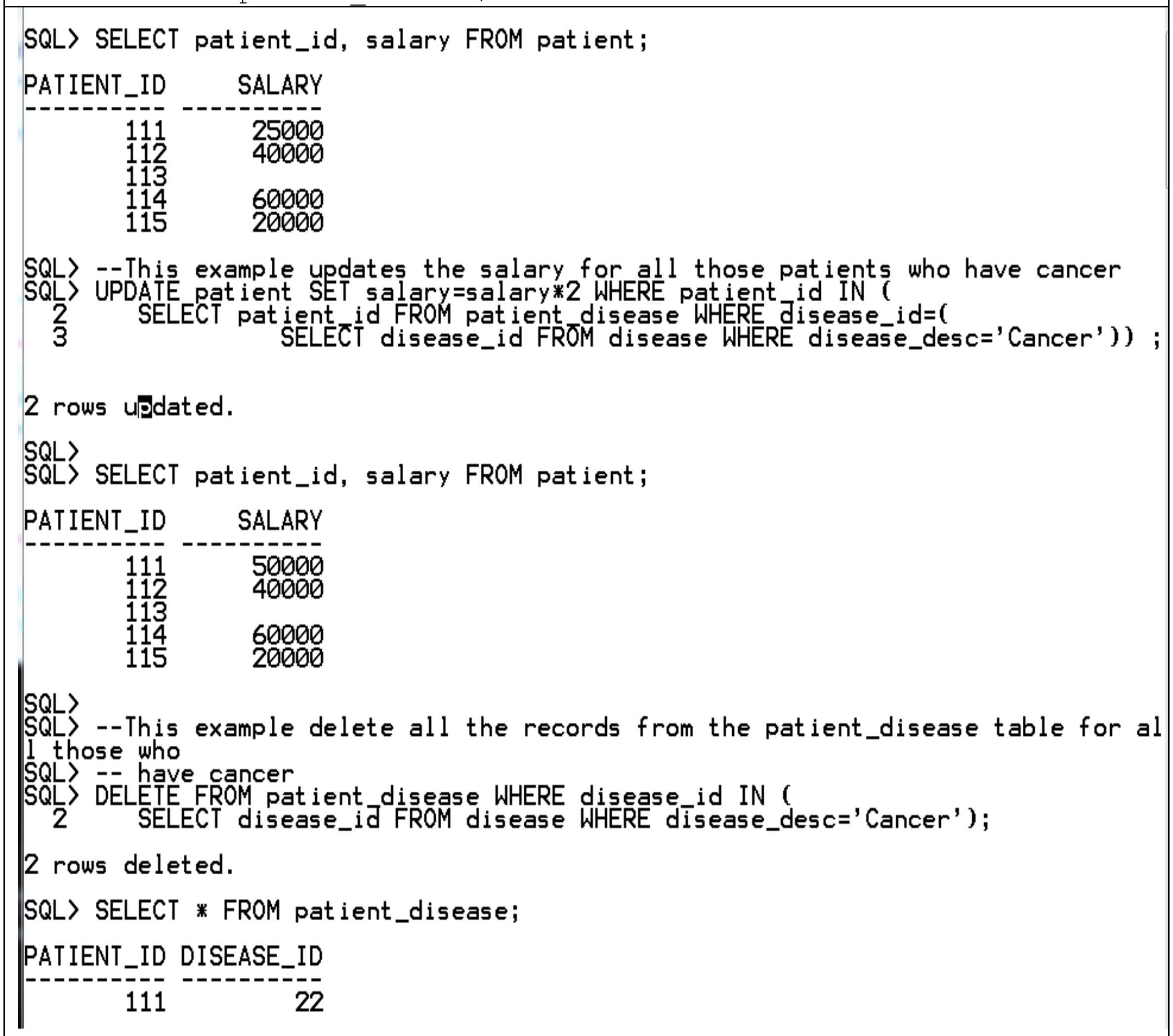


```
SQL> --Invalid. Must use an Alias because the new table will be using the inform
ation between
SQL> --the select and from to come up with the column names for the new tables.
Since
SQL> --salary*2 is not a valid column name, an alias has to be used
SQL> CREATE TABLE NEW_TABLE2 AS SELECT patient_id, salary * 2 FROM patient WHERE
 (fname,lname) IN (   SELECT fname,lname FROM  special_names);
CREATE TABLE NEW_TABLE2 AS SELECT patient_id, salary * 2 FROM patient WHERE (fna
me,lname) IN (   SELECT fname,lname FROM  special_names)
                                                     *
ERROR at line 1:
ORA-00998: must name this expression with a column alias


SQL>
SQL> --This query corrects the problem from the previous example
SQL> CREATE TABLE NEW_TABLE2 AS SELECT patient_id, salary * 2 Increase FROM pati
ent WHERE (fname,lname) IN (   SELECT fname,lname FROM  special_names);

Table created.

SQL>
SQL> SELECT * FROM NEW_TABLE2;

PATIENT_ID   INCREASE
---------- ----------
       111      50000
       113
```

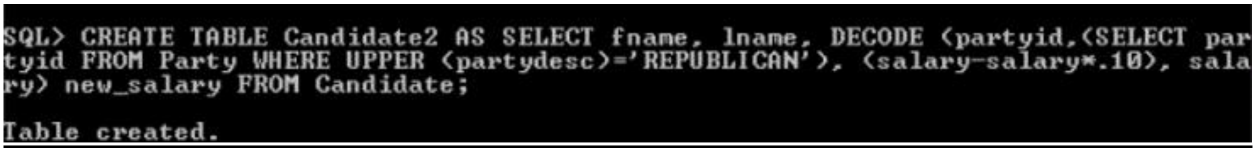
Example 9i (Update and delete using subqueries)

You can also perform UPDATE and DELETE statements

```
SELECT patient_id, salary FROM patient;

--This example updates the salary for all those patients who have cancer.
UPDATE patient SET salary=salary*2 WHERE patient_id IN (
   SELECT patient_id FROM patient_disease WHERE disease_id=(
         SELECT disease_id FROM disease WHERE disease_desc='Cancer')) ;

SELECT patient_id, salary FROM patient;

--This example deletes all the records from the patient_disease table for all those who have cancer.
DELETE FROM patient_disease WHERE disease_id IN (
   SELECT disease_id FROM disease WHERE disease_desc='Cancer');


SELECT * FROM patient_disease;
```



```
SQL> SELECT patient_id, salary FROM patient;

PATIENT_ID     SALARY
---------- ----------
       111      25000
       112      40000
       113
       114      60000
       115      20000

SQL> --This example updates the salary for all those patients who have cancer
SQL> UPDATE patient SET salary=salary*2 WHERE patient_id IN (
  2     SELECT patient_id FROM patient_disease WHERE disease_id=(
  3               SELECT disease_id FROM disease WHERE disease_desc='Cancer')) ;


2 rows updated.

SQL>
SQL> SELECT patient_id, salary FROM patient;

PATIENT_ID     SALARY
---------- ----------
       111      50000
       112      40000
       113
       114      60000
       115      20000

SQL>
SQL> --This example delete all the records from the patient_disease table for al
l those who
SQL> -- have cancer
SQL> DELETE FROM patient_disease WHERE disease_id IN (
  2     SELECT disease_id FROM disease WHERE disease_desc='Cancer');

2 rows deleted.

SQL> SELECT * FROM patient_disease;

PATIENT_ID DISEASE_ID
---------- ----------
       111         22
```



```
SQL> CREATE TABLE Candidate2 AS SELECT fname, lname, DECODE (partyid,(SELECT par
tyid FROM Party WHERE UPPER (partydesc)='REPUBLICAN'), (salary-salary*.10), sala
ry) new_salary FROM Candidate;

Table created.
```

✓

CHECK 9A

1. Display the salary of all those who are good (regardless of case).
2. Display only the personality description for those people who have a personality.
3. Display the name of all those who are making more than the average salary.
4. Delete all those who are making less than the average salary.
5. What is wrong with the following?

```
SELECT * FROM patient WHERE patient_id =
  (SELECT * FROM patient_disease WHERE disease_id =
    (SELECT disease_id FROM disease WHERE disease_desc=’MaLaRia’ ORDER BY fname);


SELECT * FROM patient WHERE (fname, lname) IN (SELECT fname FROM patient2);
```

“The true price of anything is the amount of life you exchange for it”

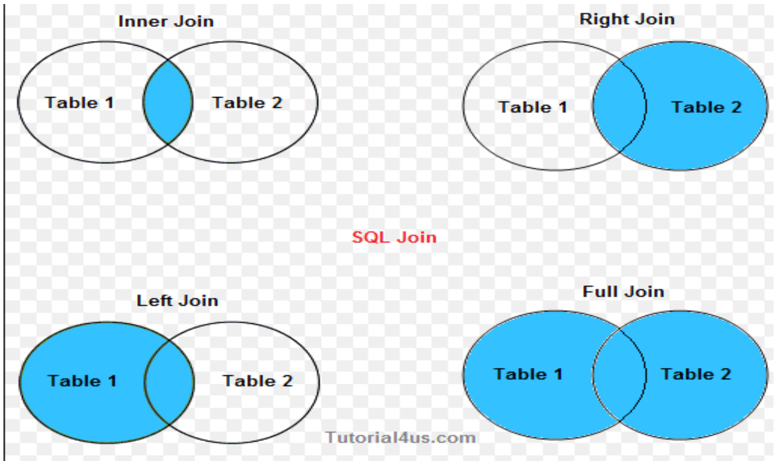
Summary Examples

```
--This example updates the salary for all those patients who have cancer.
--Notice the IN clause will have to be used to deal with multiple rows.
--Also, note that the where condition for the outer query filters on both the patient_id and the salary.
SELECTpatient_id, salary FROM patient WHERE patient_id IN (
   SELECT patient_id FROM patient_disease WHERE disease_id=(
     SELECT disease_id FROM disease WHERE to_lower(disease_desc)='cancer'))
AND salary IS NOT NULL ;




--Notice that the AVG function is enclosed in its own query. Salary>AVG(salary) is wrong.
SELECT patient_id, salary FROM patient WHERE salary >
   (SELECT AVG(salary) FROM Patient)
```

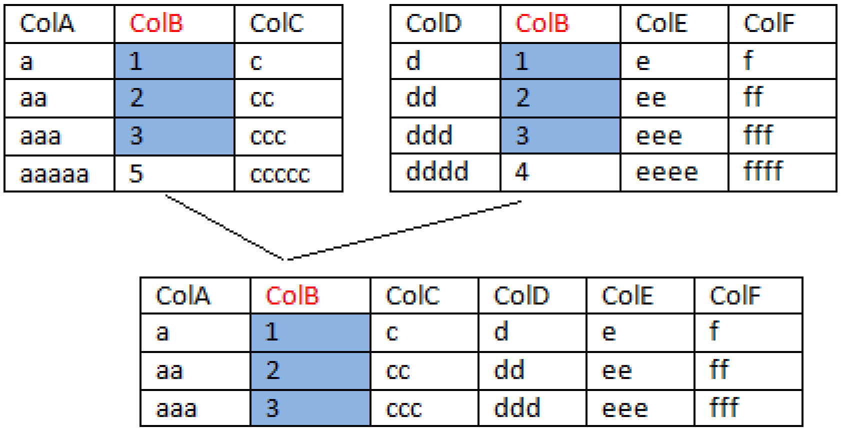
Warren Buffett, Bill Gates, Ted Turner, George Lucas, Larry Ellison and Mayor Bloomberg

Ellison wrote: "Many years ago, I put virtually all of my assets into a trust with the intent of giving away at least 95 percent of my wealth to charitable causes. I have already given hundreds of millions of dollars to medical research and education, and I will give billions more over time. Until now, I have done this giving quietly—because I have long believed that charitable giving is a personal and private matter.

Chapter 10 (Joins)



"Life is not the amount of breaths you take, it's the moments that take your breath away…"



ColA	ColB	ColC
a	1	c
aa	2	cc
aaa	3	ccc
aaaaa	5	ccccc

ColD	ColB	ColE	ColF
d	1	e	f
dd	2	ee	ff
ddd	3	eee	fff
dddd	4	eeee	ffff

ColA	ColB	ColC	ColD	ColE	ColF
a	1	c	d	e	f
aa	2	cc	dd	ee	ff
aaa	3	ccc	ddd	eee	fff

Cartesian product or CROSS JOIN	Replicates each row from the first table with every row from the second table
Equality join also known as equijoin, inner join or a simple join	Creates a join by using a commonly named and defined column
Non-equality join	Joins tables when there are no equivalent rows in the tables to be joined
Self-join	Joins a table to itself.
Outer join	Includes records of a table when there is no matching record in the other table.
Set operators UNION, UNION ALL, INTERSECT and MINUS	Combines results from multiple select statements

WHERE	In the traditional approach, the WHERE clause indicates which columns should be joined
NATURAL JOIN	The keywords are used in the FROM clause to join tables containing a common column with the same name and definition
JOIN … USING	The JOIN keyword is used in the FROM clause; combined with the USING clause, it identifies the common column used to join the tables.

JOIN … ON	The JOIN keyword is used in the FROM clause. The ON clause identifies the columns used to join the tables
OUTER JOIN can be used with LEFT, RIGHT, FULL	Indicates that at least one of the tables doesn’t have a matching row in the other table

```
DROP TABLE patient_disease;
DROP TABLE patient;
CREATE  TABLE Patient
(
      Patient_id  NUMBER PRIMARY KEY,
      Fname       VARCHAR2(20),
      Lname       VARCHAR2(20),
      Gender            CHAR,
      DOB         DATE,
      salary            NUMBER ,
               city                      VARCHAR2(20),
               state                   VARCHAR2(20)
);

INSERT INTO patient values (111,'john','Doe','m','11-FEB-1978',25000,
'Davis','CA');
INSERT INTO patient values (113,'jill','Crane','m','12-APR-
1999',NULL,'Reno','NV');
INSERT INTO patient values (114,'billy','Bob','f','05-MAY-1985',60000,'Las
Vegas','NV');

DROP TABLE disease;
CREATE  TABLE disease
 (
      disease_id  NUMBER PRIMARY KEY,
      disease_desc      VARCHAR2(20)
);
INSERT INTO disease VALUES (11,'Cancer');
INSERT INTO disease VALUES (22,'Malaria');
INSERT INTO disease VALUES (33,'Flu');

CREATE  TABLE patient_disease
 (
               Patient_id          NUMBER REFERENCES patient,
      disease_id  NUMBER REFERENCES disease,
               PRIMARY KEY (patient_id, disease_id)
);
INSERT INTO patient_disease VALUES (111,11);
INSERT INTO patient_disease VALUES (111,22);
INSERT INTO patient_disease VALUES (113,11);
```

Patient

Patient_id	Fname	Lname…	
111	john	Doe	…
114	billy	Bob	…
113	jill	Crane	…

Patient_Disease

Patient_id	Disease_id
111	11
111	22
113	11

Disease

Disease_id	Disease_desc
11	Cancer
22	Malaria
33	Flu

```
DROP TABLE heroes;
DROP TABLE skills;
CREATE TABLE heroes
(
  name VARCHAR(10),
  skill_code NUMBER
);
INSERT INTO heroes VALUES ('superman',1);
INSERT INTO heroes VALUES ('aqua-man',2);
INSERT INTO heroes VALUES ('flash',NULL);

CREATE TABLE skills
(
  skill_code NUMBER,
  skill_name VARCHAR(10)
);
INSERT INTO skills VALUES (1,'fly');
INSERT INTO skills VALUES (2,'swim');
INSERT INTO skills VALUES (3,'freeze');
```

Heroes

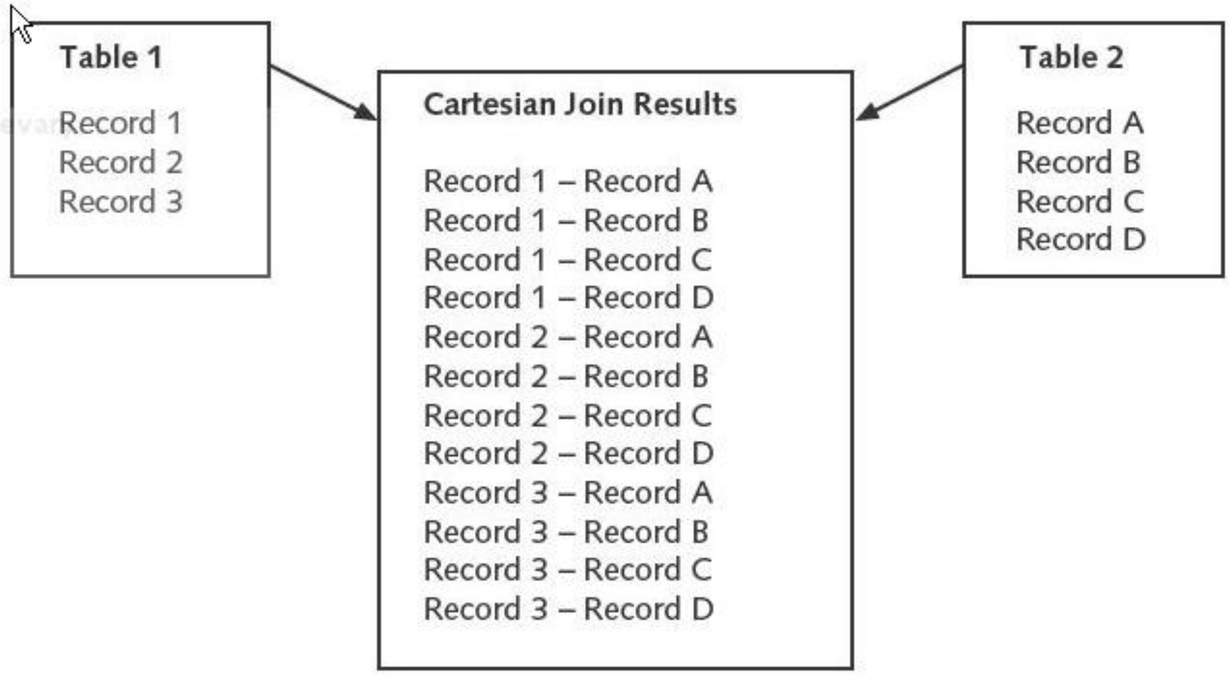
Name	skill_code
superman	1
Aquaman	2
Flash	NULL

Skills

Skill_code	skill_name
1	fly
2	swim
3	freeze

10.1 Cartesian/Cross Join

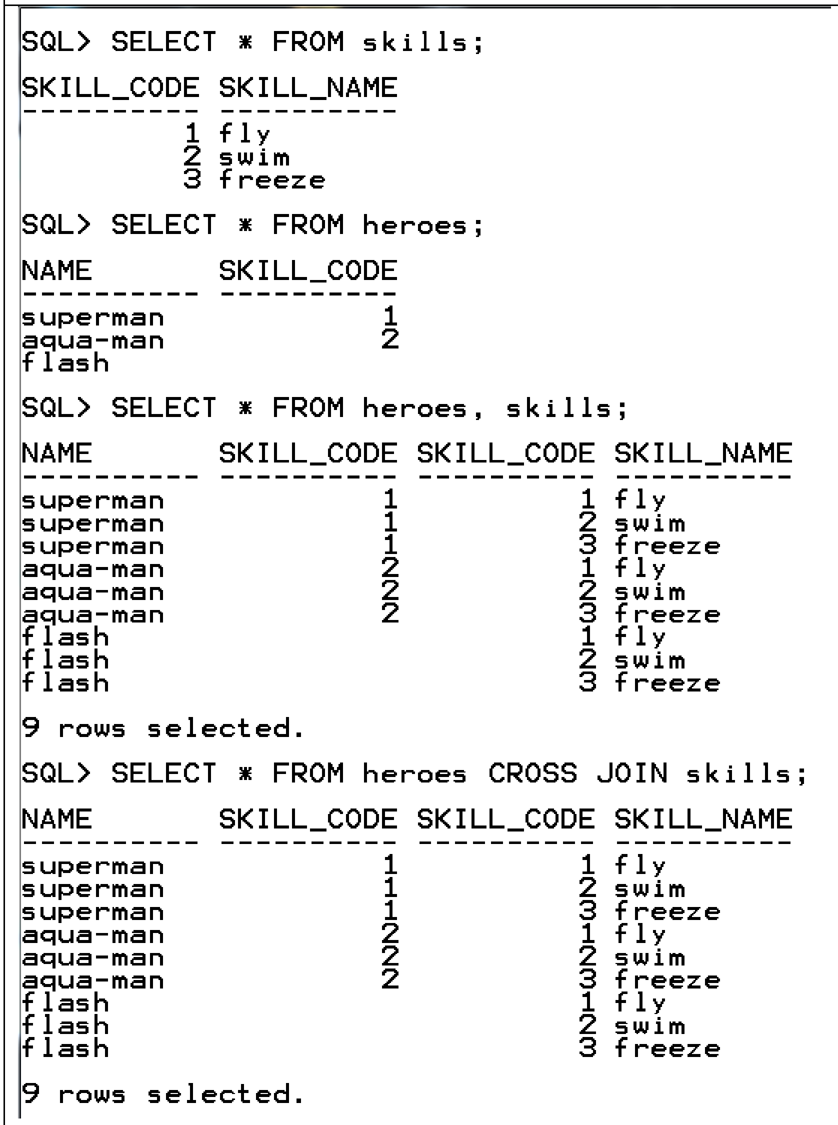
In a Cartesian join, also called a Cartesian product or cross join, each record in the first table is matched with each record in the second table. This type of join is useful when you’re performing certain statistical procedures for data analysis. Therefore, if you have three records in the first table and four in the second table, the first record from the first table is matched with each of the four records in the second table. Then the second record of the first table is matched with each of the four records from the second table, and so on.



The CROSS keyword, combined with the JOIN keyword, can be used in the FROM clause to explicitly instruct Oracle to create a Cartesian (cross) join. The CROSS JOIN keywords instruct the database system to create cross- products, using all records of the tables listed in the query.

Example 10.1a (Cartesian product, cross join)

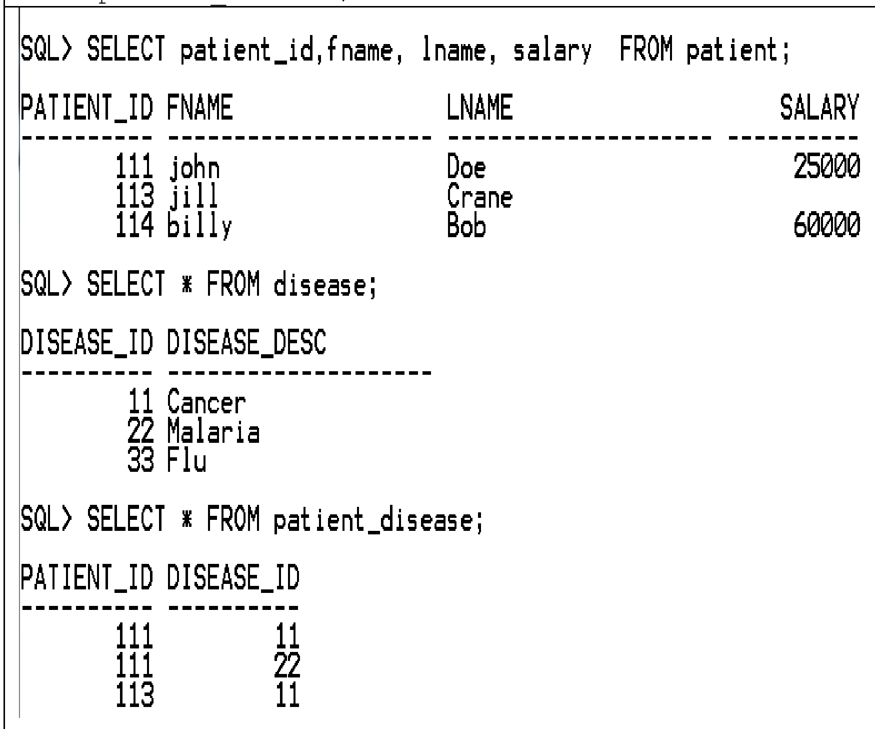
```
SELECT * FROM skills;
SELECT * FROM heroes;

--Cartesian product gives every combination.
SELECT * FROM heroes, skills;

--Alternative way to do a cartesian product is to use a cross join.
SELECT * FROM heroes CROSS JOIN skills;
```



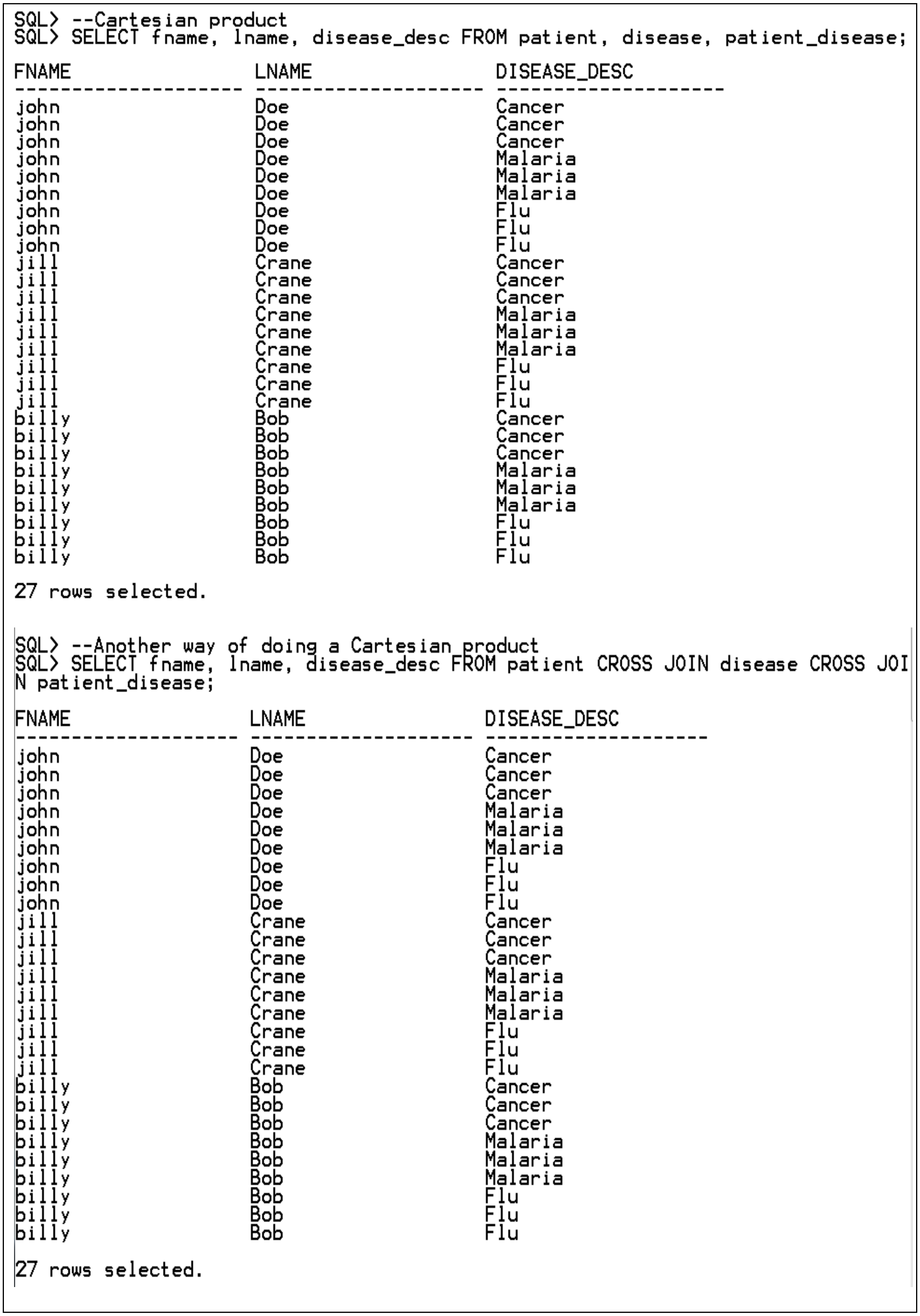
```
SQL> SELECT * FROM skills;

SKILL_CODE SKILL_NAME
---------- ----------
         1 fly
         2 swim
         3 freeze

SQL> SELECT * FROM heroes;

NAME       SKILL_CODE
---------- ----------
superman            1
aqua-man            2
flash

SQL> SELECT * FROM heroes, skills;

NAME       SKILL_CODE SKILL_CODE SKILL_NAME
---------- ---------- ---------- ----------
superman            1          1 fly
superman            1          2 swim
superman            1          3 freeze
aqua-man            2          1 fly
aqua-man            2          2 swim
aqua-man            2          3 freeze
flash                          1 fly
flash                          2 swim
flash                          3 freeze

9 rows selected.

SQL> SELECT * FROM heroes CROSS JOIN skills;

NAME       SKILL_CODE SKILL_CODE SKILL_NAME
---------- ---------- ---------- ----------
superman            1          1 fly
superman            1          2 swim
superman            1          3 freeze
aqua-man            2          1 fly
aqua-man            2          2 swim
aqua-man            2          3 freeze
flash                          1 fly
flash                          2 swim
flash                          3 freeze

9 rows selected.
```

Example 10.1b (Patient example)

```
SELECT patient_id,fname, lname, salary  FROM patient;
SELECT * FROM disease;
SELECT * FROM patient_disease;

--Cartesian product
SELECT fname, lname, disease_desc FROM patient, disease,
patient_disease;

--Another way of doing a cartesian product.
SELECT fname, lname, disease_desc FROM patient CROSS JOIN disease CROSS
JOIN patient_disease;
```



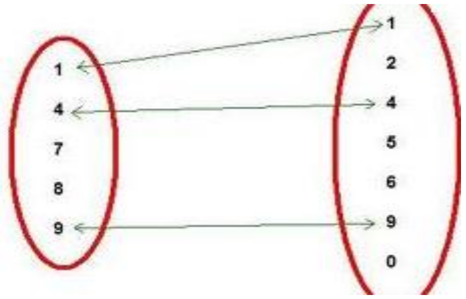
```
SQL> SELECT patient_id,fname, lname, salary  FROM patient;

PATIENT_ID FNAME                LNAME                    SALARY
---------- -------------------- -------------------- ----------
       111 john                 Doe                       25000
       113 jill                 Crane
       114 billy                Bob                       60000

SQL> SELECT * FROM disease;

DISEASE_ID DISEASE_DESC
---------- --------------------
        11 Cancer
        22 Malaria
        33 Flu

SQL> SELECT * FROM patient_disease;

PATIENT_ID DISEASE_ID
---------- ----------
       111         11
       111         22
       113         11
```



```
SQL> --Cartesian product
SQL> SELECT fname, lname, disease_desc FROM patient, disease, patient_disease;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Cancer
john                 Doe                  Cancer
john                 Doe                  Malaria
john                 Doe                  Malaria
john                 Doe                  Malaria
john                 Doe                  Flu
john                 Doe                  Flu
john                 Doe                  Flu
jill                 Crane                Cancer
jill                 Crane                Cancer
jill                 Crane                Cancer
jill                 Crane                Malaria
jill                 Crane                Malaria
jill                 Crane                Malaria
jill                 Crane                Flu
jill                 Crane                Flu
jill                 Crane                Flu
billy                Bob                  Cancer
billy                Bob                  Cancer
billy                Bob                  Cancer
billy                Bob                  Malaria
billy                Bob                  Malaria
billy                Bob                  Malaria
billy                Bob                  Flu
billy                Bob                  Flu
billy                Bob                  Flu

27 rows selected.
```

```
SQL> --Another way of doing a Cartesian product
SQL> SELECT fname, lname, disease_desc FROM patient CROSS JOIN disease CROSS JOI
N patient_disease;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Cancer
john                 Doe                  Cancer
john                 Doe                  Malaria
john                 Doe                  Malaria
john                 Doe                  Malaria
john                 Doe                  Flu
john                 Doe                  Flu
john                 Doe                  Flu
jill                 Crane                Cancer
jill                 Crane                Cancer
jill                 Crane                Cancer
jill                 Crane                Malaria
jill                 Crane                Malaria
jill                 Crane                Malaria
jill                 Crane                Flu
jill                 Crane                Flu
jill                 Crane                Flu
billy                Bob                  Cancer
billy                Bob                  Cancer
billy                Bob                  Cancer
billy                Bob                  Malaria
billy                Bob                  Malaria
billy                Bob                  Malaria
billy                Bob                  Flu
billy                Bob                  Flu
billy                Bob                  Flu

27 rows selected.
```

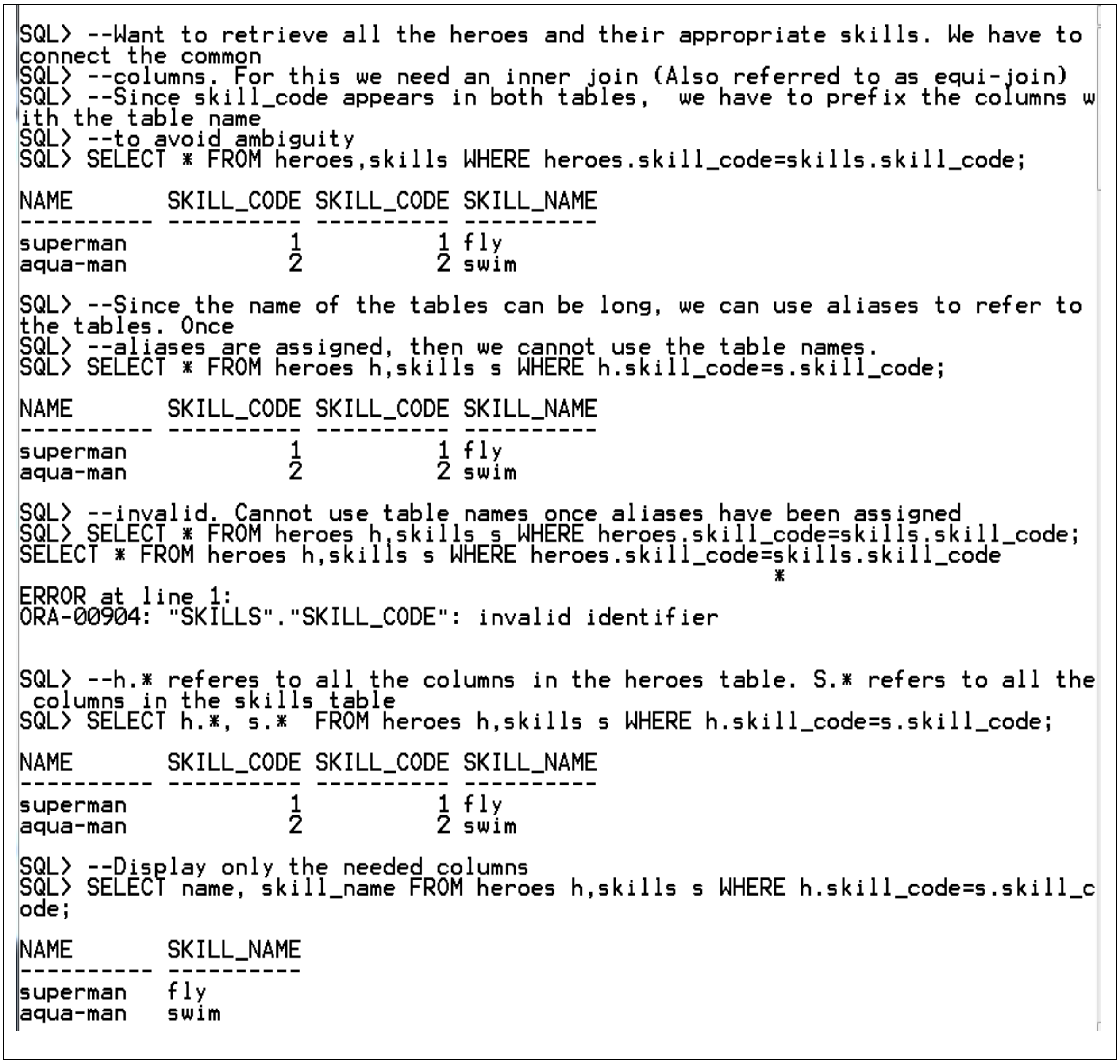
10.2 Inner Join



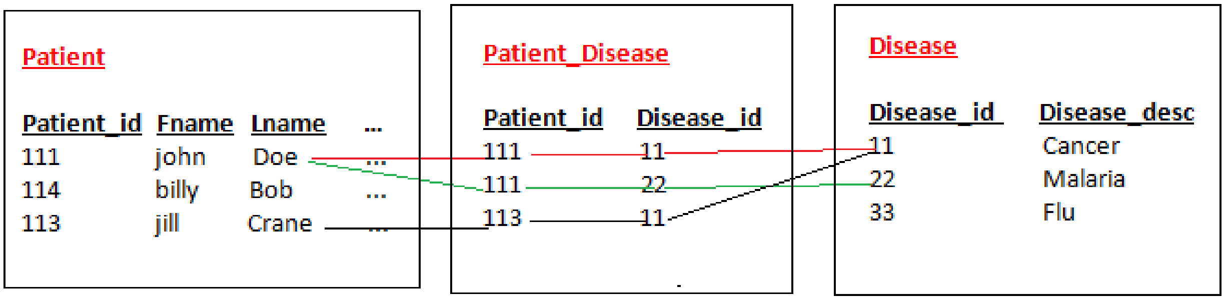
The most common type of join is based on two (or more) tables having equivalent data stored in a common column. These joins are called equality joins but are also referred to as equijoins, inner joins, or simple joins.The traditional way to include join conditions and avoid an unintended Cartesian result is to use the WHERE clause. The WHERE clause can perform two different activities: joining tables and providing conditions to limit or filter the rows that are affected.A column qualifier indicates the table containing the column being referenced.With the equality, non- equality, and self- joins, a row is returned only if a corresponding record in each table is queried. These types of joins can be categorized as inner joins because records are listed in the results only if a match is found in each table.

Example 10.2a (Simple join)

```
--We want to retrieve all the heroes and their corresponding skills. We have to connect the common
--columns. For this we need an inner join (Also referred to as equi-join).
--Since skill_code appears in both tables,  we have to prefix the columns with the table name
--to avoid ambiguity.
SELECT * FROM heroes,skills WHERE heroes.skill_code=skills.skill_code;

--Since the name of the tables can be long, we can use aliases to refer to the tables. Once
--aliases are assigned, we cannot use the table names.
SELECT * FROM heroes h,skills s WHERE h.skill_code=s.skill_code;

--Invalid: cannot use table names once aliases have been assigned.
SELECT * FROM heroes h,skills s WHERE heroes.skill_code=skills.skill_code;

--h.* referes to all the columns in the heroes table. S.* refers to all the columns in the skills table.
SELECT h.*, s.*  FROM heroes h,skills s WHERE h.skill_code=s.skill_code;

--Display only the needed columns.
SELECT name, skill_name FROM heroes h,skills s WHERE
h.skill_code=s.skill_code;
```



```
SQL> --Want to retrieve all the heroes and their appropriate skills. We have to
connect the common
SQL> --columns. For this we need an inner join (Also referred to as equi-join)
SQL> --Since skill_code appears in both tables,  we have to prefix the columns w
ith the table name
SQL> --to avoid ambiguity
SQL> SELECT * FROM heroes,skills WHERE heroes.skill_code=skills.skill_code;

NAME       SKILL_CODE SKILL_CODE SKILL_NAME
---------- ---------- ---------- ----------
superman            1          1 fly
aqua-man            2          2 swim

SQL> --Since the name of the tables can be long, we can use aliases to refer to
the tables. Once
SQL> --aliases are assigned, then we cannot use the table names.
SQL> SELECT * FROM heroes h,skills s WHERE h.skill_code=s.skill_code;

NAME       SKILL_CODE SKILL_CODE SKILL_NAME
---------- ---------- ---------- ----------
superman            1          1 fly
aqua-man            2          2 swim

SQL> --invalid. Cannot use table names once aliases have been assigned
SQL> SELECT * FROM heroes h,skills s WHERE heroes.skill_code=skills.skill_code;
SELECT * FROM heroes h,skills s WHERE heroes.skill_code=skills.skill_code
                                                        *
ERROR at line 1:
ORA-00904: "SKILLS"."SKILL_CODE": invalid identifier


SQL> --h.* referes to all the columns in the heroes table. S.* refers to all the
 columns in the skills table
SQL> SELECT h.*, s.*  FROM heroes h,skills s WHERE h.skill_code=s.skill_code;

NAME       SKILL_CODE SKILL_CODE SKILL_NAME
---------- ---------- ---------- ----------
superman            1          1 fly
aqua-man            2          2 swim

SQL> --Display only the needed columns
SQL> SELECT name, skill_name FROM heroes h,skills s WHERE h.skill_code=s.skill_c
ode;

NAME       SKILL_NAME
---------- ----------
superman   fly
aqua-man   swim
```



Patient

Patient_id	Fname	Lname	...
111	john	Doe	...
114	billy	Bob	...
113	jill	Crane	...

Patient_Disease

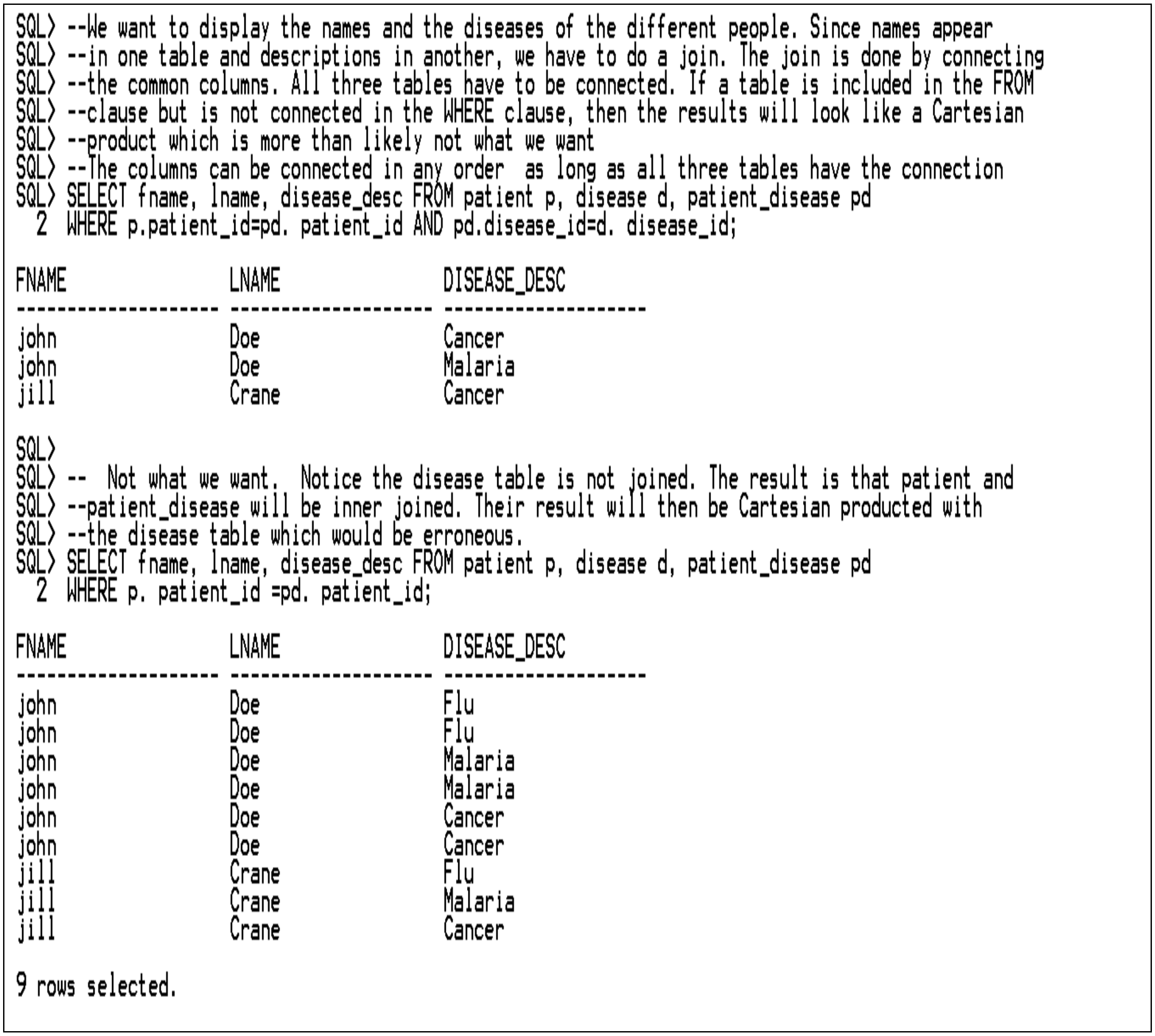
Patient_id	Disease_id
111	11
111	22
113	11

Disease

Disease_id	Disease_desc
11	Cancer
22	Malaria
33	Flu

Example 10.2b (Patient example)

```
--We want to display the names and the diseases of the different people. Since names appear
--in one table and descriptions in another, we have to do a join. The join is done by connecting
--the common columns. All three tables have to be connected. If a table is included in the FROM
--clause but is not connected in the WHERE clause, then the results will look like a Cartesian
--product, which is more than likely not what we want.
--The columns can be connected in any order as long as all three tables have the connection.
SELECT fname, lname,disease_desc FROM patient p, disease d, patient_disease pd
WHERE p.patient_id=pd. patient_id AND pd.disease_id=d. disease_id;


--This is not what we want.  Notice the disease table is not joined. The result is that patient and
--patient_disease will be inner joined. Their result will then be cross joined with
--the disease table, which would logically be erroneous.
SELECT fname, lname, disease_desc FROM patient p, disease d, patient_disease pd
WHERE p. patient_id=pd. patient_id;
```



```
SQL> --We want to display the names and the diseases of the different people. Since names appear
SQL> --in one table and descriptions in another, we have to do a join. The join is done by connecting
SQL> --the common columns. All three tables have to be connected. If a table is included in the FROM
SQL> --clause but is not connected in the WHERE clause, then the results will look like a Cartesian
SQL> --product which is more than likely not what we want
SQL> --The columns can be connected in any order  as long as all three tables have the connection
SQL> SELECT fname, lname, disease_desc FROM patient p, disease d, patient_disease pd
  2  WHERE p.patient_id=pd. patient_id AND pd.disease_id=d. disease_id;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer

SQL>
SQL> --  Not what we want.  Notice the disease table is not joined. The result is that patient and
SQL> --patient_disease will be inner joined. Their result will then be Cartesian producted with
SQL> --the disease table which would be erroneous.
SQL> SELECT fname, lname, disease_desc FROM patient p, disease d, patient_disease pd
  2  WHERE p. patient_id =pd. patient_id;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Flu
john                 Doe                  Flu
john                 Doe                  Malaria
john                 Doe                  Malaria
john                 Doe                  Cancer
john                 Doe                  Cancer
jill                 Crane                Flu
jill                 Crane                Malaria
jill                 Crane                Cancer

9 rows selected.
```

You can use three approaches to create an equality join that uses the JOIN keyword: NATURAL JOIN, JOIN . . . USING, and JOIN . . . ON:

- The NATURAL JOIN keywords create a join automatically between two tables, based on columns with matching names.

- The USING clause allows you to create joins based on a column that has the same name and definition in both tables.

- When the tables to be joined in a USING clause don't have a commonly named and defined field, you must add the ON clause to the JOIN keyword to specify how the tables are related.

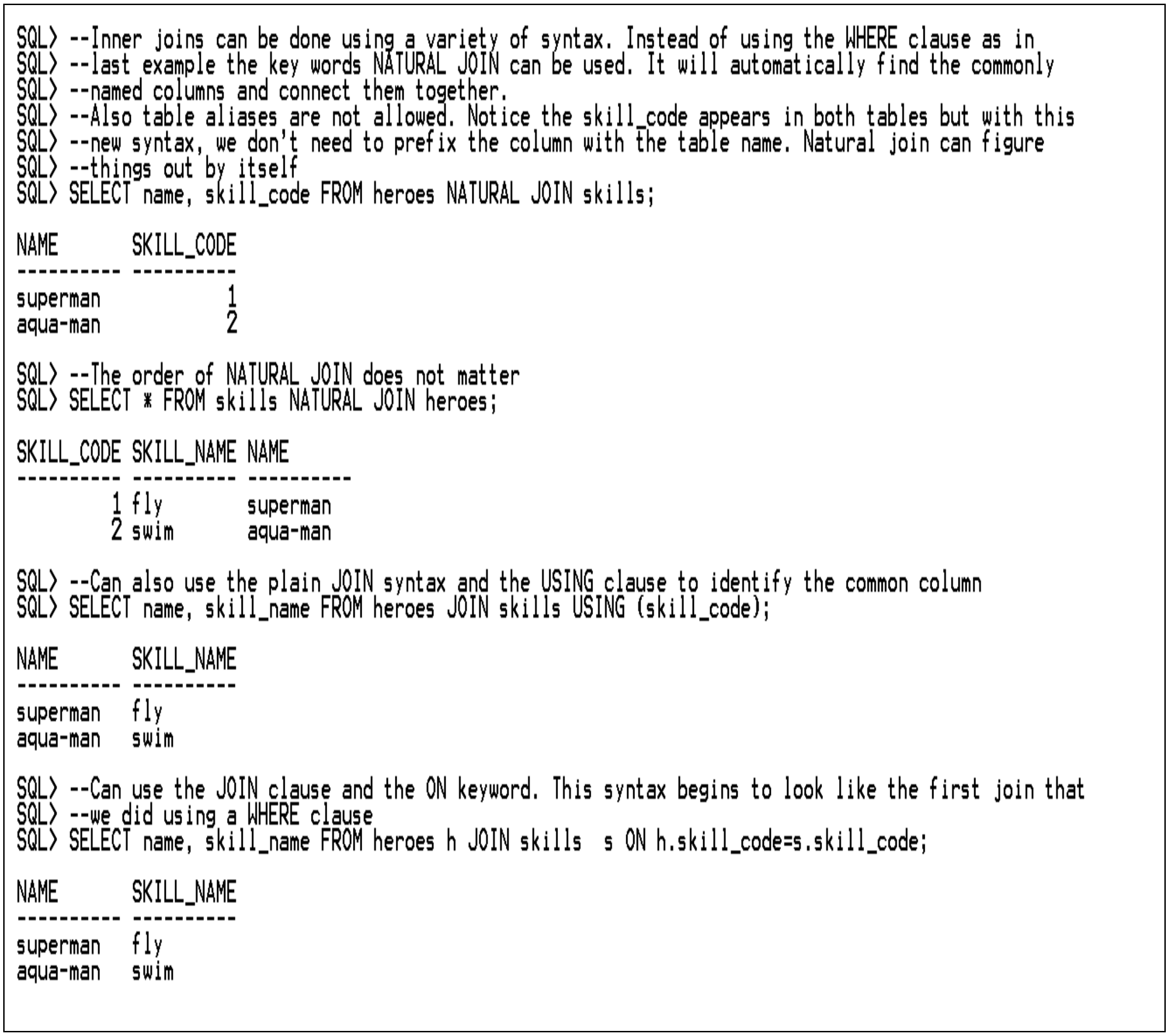
There are two main differences between using the USING and ON clauses with the JOIN keyword:

- The USING clause can be used only if the tables being joined have a common column with the same name. This rule isn't a requirement for the ON clause.

- A condition is specified in the ON clause; this isn't allowed in the USING clause. The USING clause can contain only the name of the common column.

Example 10.2c (Natural join)

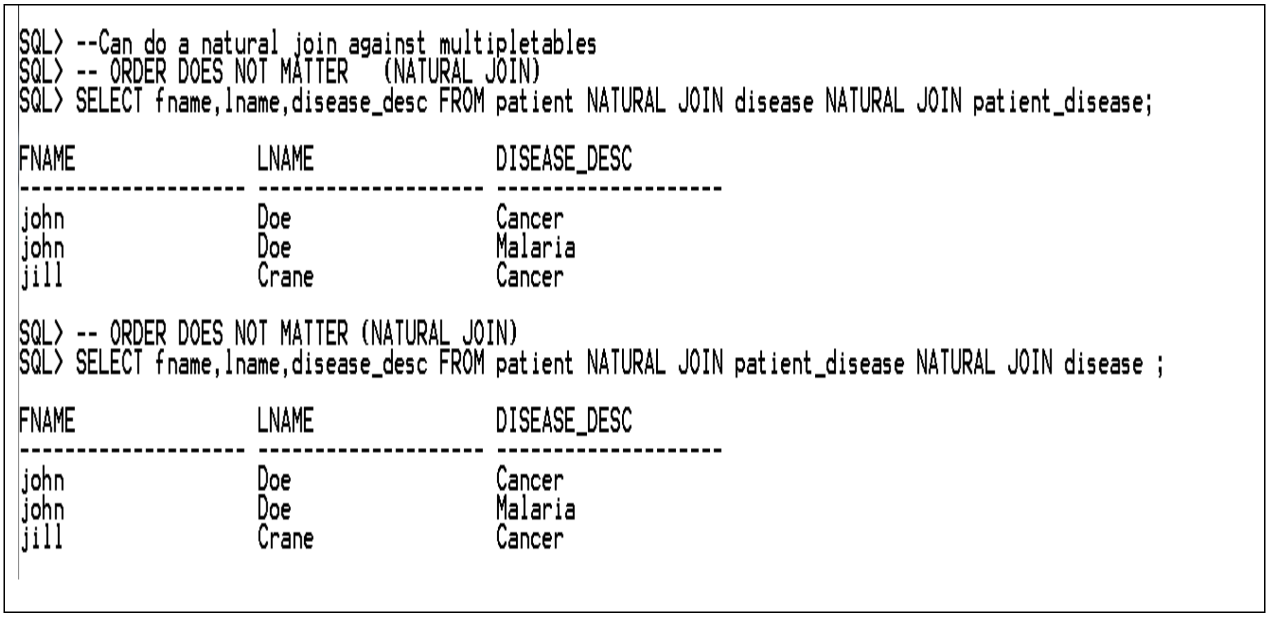
```
--Inner joins can be done using a variety of syntax. Instead of using the WHERE clause as in
--last example, the key words NATURAL JOIN can be used. It will automatically find the commonly
--named columns and connect them together.
--Also, table aliases are not allowed. Notice the skill_code appears in both tables but with this
--new syntax, we don't need to prefix the column with the table name. Natural join can figure
--things out by itself.
SELECT name, skill_code FROM heroes NATURAL JOIN skills;

--The order of NATURAL JOIN does not matter.
SELECT * FROM skills NATURAL JOIN heroes;

--Can also use the plain JOIN syntax and the USING clause to identify the common column.
SELECT name, skill_name FROM heroes JOIN skills USING (skill_code);

--Can use the JOIN clause and the ON keyword. This syntax begins to look like the first join that
--we did using a WHERE clause.
SELECT name, skill_nameFROM heroesh JOIN skills  s ON h.skill_code=s.skill_code;
```



```
SQL> --Inner joins can be done using a variety of syntax. Instead of using the WHERE clause as in
SQL> --last example the key words NATURAL JOIN can be used. It will automatically find the commonly
SQL> --named columns and connect them together.
SQL> --Also table aliases are not allowed. Notice the skill_code appears in both tables but with this
SQL> --new syntax, we don't need to prefix the column with the table name. Natural join can figure
SQL> --things out by itself
SQL> SELECT name, skill_code FROM heroes NATURAL JOIN skills;

NAME       SKILL_CODE
---------- ----------
superman            1
aqua-man            2

SQL> --The order of NATURAL JOIN does not matter
SQL> SELECT * FROM skills NATURAL JOIN heroes;

SKILL_CODE SKILL_NAME NAME
---------- ---------- ----------
         1 fly        superman
         2 swim       aqua-man

SQL> --Can also use the plain JOIN syntax and the USING clause to identify the common column
SQL> SELECT name, skill_name FROM heroes JOIN skills USING (skill_code);

NAME       SKILL_NAME
---------- ----------
superman   fly
aqua-man   swim

SQL> --Can use the JOIN clause and the ON keyword. This syntax begins to look like the first join that
SQL> --we did using a WHERE clause
SQL> SELECT name, skill_name FROM heroes h JOIN skills  s ON h.skill_code=s.skill_code;

NAME       SKILL_NAME
---------- ----------
superman   fly
aqua-man   swim
```

Example 10.2d (Patient Example)

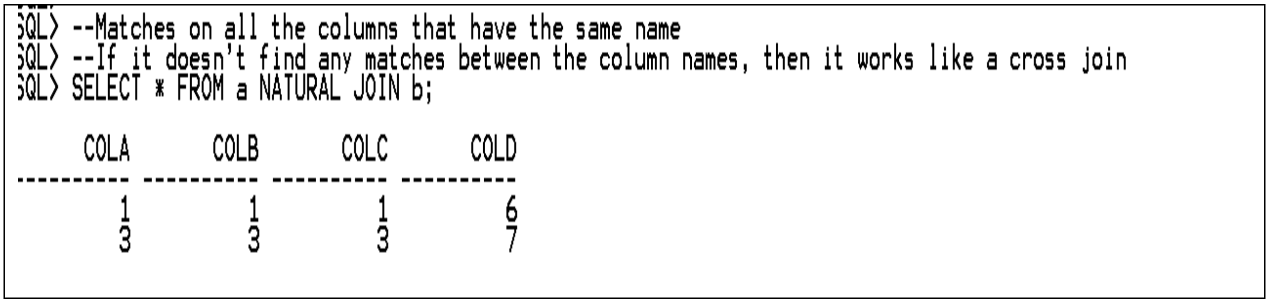
```
--Can do a natural join against multiple tables.
--Order does not matter  when using natural join.
SELECT fname,lname,disease_descFROM patient NATURAL JOIN disease NATURAL
JOIN patient_disease;

-- ORDER DOES NOT MATTER (NATURAL JOIN)
SELECT fname,lname,disease_desc FROM patient NATURAL JOIN patient_disease
NATURAL JOIN disease ;
```



```
SQL> --Can do a natural join against multipletables
SQL> -- ORDER DOES NOT MATTER   (NATURAL JOIN)
SQL> SELECT fname,lname,disease_desc FROM patient NATURAL JOIN disease NATURAL JOIN patient_disease;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer

SQL> -- ORDER DOES NOT MATTER (NATURAL JOIN)
SQL> SELECT fname,lname,disease_desc FROM patient NATURAL JOIN patient_disease NATURAL JOIN disease ;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
```

Example 10.2e (Natural join with multiple columns)

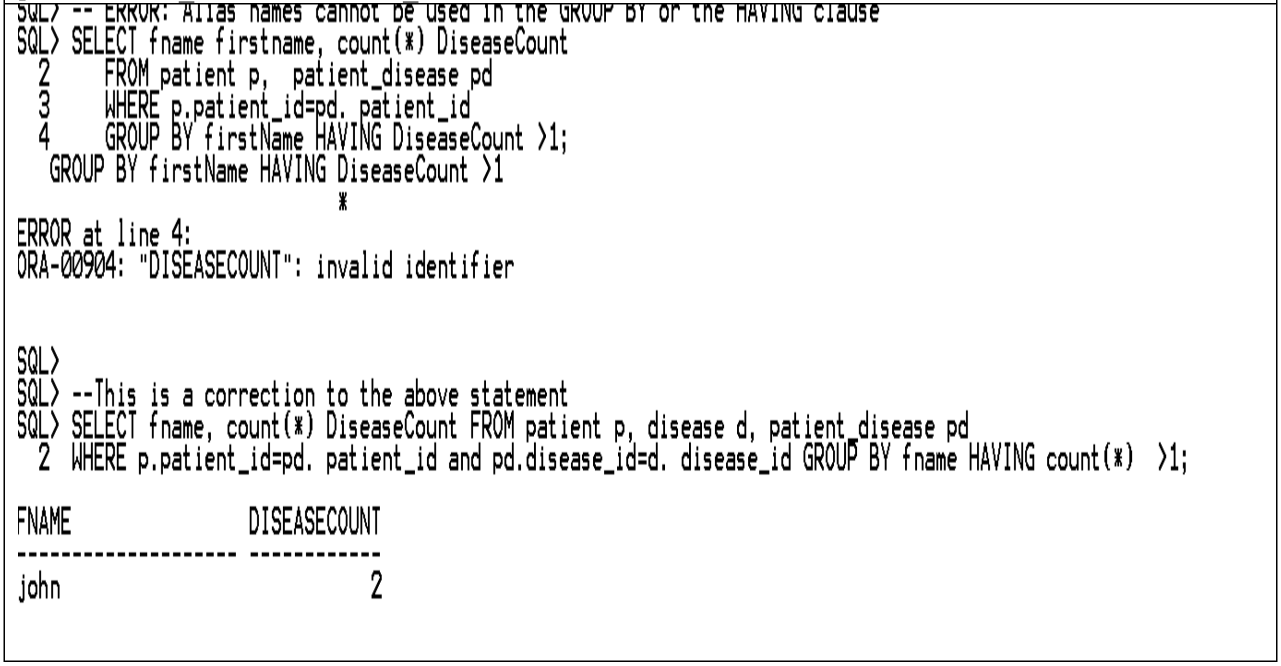
```
Drop TABLE a;
DROP TABLE b;
CREATE taBLE A
(
   COLA NUMBER,
   COLB NUMBER,
   COLC NUMBER
);

INSERT INTO a VALUES (1,1,1);
INSERT INTO a VALUES (2,2,2);
INSERT INTO a VALUES (3,3,3);
INSERT INTO a VALUES (4,4,4);

CREATE TABLE B (COLd NUMBER, COLB NUMBER, COLA NUMBER);
INSERT INTO b VALUES (6,1,1);
INSERT INTO b VALUES (9,3,2);
INSERT INTO b VALUES (7,3,3);
INSERT INTO b VALUES (5,5,5);



--Matches on all the columns that have the same name.
--If it doesn’t find any matches between the column names, then it works like a cross join.
SELECT * FROM a NATURAL JOIN b;
```



```
SQL> --Matches on all the columns that have the same name
SQL> --If it doesn't find any matches between the column names, then it works like a cross join
SQL> SELECT * FROM a NATURAL JOIN b;

      COLA       COLB       COLC       COLD
---------- ---------- ---------- ----------
         1          1          1          6
         3          3          3          7
```

Example 10.2f (Patient Example with GROUP BY)

```
--In this example, we are trying to find the number of diseases each person has as long as they
--they have more than one disease.  Since the name comes from the patient table but the actual disease
--association comes from patient_disease, we have to do an inner join.
--ERROR: Alias names cannot be used in the GROUP BY or the HAVING clause.
SELECT fname firstname, count(*) DiseaseCount
   FROM patient p,  patient_disease pd
   WHERE p.patient_id=pd. patient_id
   GROUP BY firstName HAVING DiseaseCount >1;



--This is a correction to the above statement.
SELECT fname, count(*) DiseaseCount FROM patient p, disease d,
patient_disease pd WHERE p.patient_id=pd. patient_id and
pd.disease_id=d. disease_idGROUP BY fname HAVING count(*)  >1;
```



```
SQL> -- ERROR: Alias names cannot be used in the GROUP BY or the HAVING clause
SQL> SELECT fname firstname, count(*) DiseaseCount
  2     FROM patient p,  patient_disease pd
  3     WHERE p.patient_id=pd. patient_id
  4     GROUP BY firstName HAVING DiseaseCount >1;
   GROUP BY firstName HAVING DiseaseCount >1
                             *
ERROR at line 4:
ORA-00904: "DISEASECOUNT": invalid identifier


SQL>
SQL> --This is a correction to the above statement
SQL> SELECT fname, count(*) DiseaseCount FROM patient p, disease d, patient_disease pd
  2  WHERE p.patient_id=pd. patient_id and pd.disease_id=d. disease_id GROUP BY fname HAVING count(*)  >1;

FNAME                DISEASECOUNT
-------------------- ------------
john                            2
```

A **non- equality join** is used when the related columns can't be joined with an equal sign— meaning there are no equivalent rows in the tables to be joined.

```
DROP TABLE students;
DROP TABLE grade_range;

CREATE TABLE students
(
name VARCHAR(10),
score NUMBER
);
INSERT INTO students VALUES ('jack',80);
INSERT INTO students VALUES ('scott',73);


CREATE TABLE grade_range
(
   beg_score NUMBER,
   end_score NUMBER,
   grade char
);

INSERT INTO grade_range VALUES (90,100,'A');
INSERT INTO grade_range VALUES (80,89,'B');
INSERT INTO grade_range VALUES (70,79,'C');
INSERT INTO grade_range VALUES (60,69,'D');
```

students

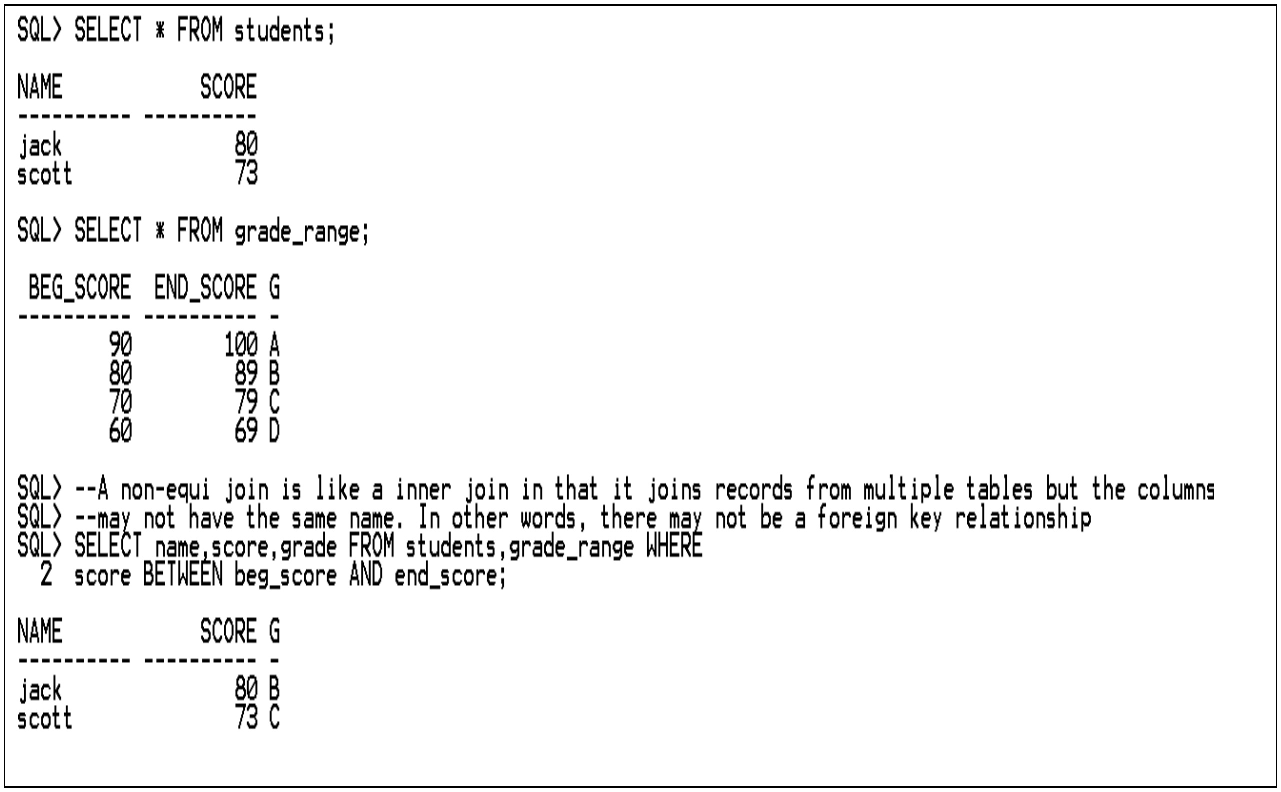
Name	score
Jack	80
Scott	73

Grade_range

Beg_score	end_score	grade
90	100	A
80	89	B
70	79	C
60	69	D

Example 10.2g (Non-equi join)

```
SELECT * FROM students;
SELECT * FROM grade_range;

--A non-equi join is like a inner join in that it joins records from multiple tables but the columns
--may not have the same name. In other words, there may not be a foreign key relationship.
SELECT name,score,grade FROM students,grade_range WHERE
score BETWEEN beg_score AND end_score;
```



```
SQL> SELECT * FROM students;

NAME            SCORE
---------- ----------
jack               80
scott              73

SQL> SELECT * FROM grade_range;

 BEG_SCORE  END_SCORE G
---------- ---------- -
        90        100 A
        80         89 B
        70         79 C
        60         69 D

SQL> --A non-equi join is like a inner join in that it joins records from multiple tables but the columns
SQL> --may not have the same name. In other words, there may not be a foreign key relationship
SQL> SELECT name,score,grade FROM students,grade_range WHERE
  2  score BETWEEN beg_score AND end_score;

NAME            SCORE G
---------- ---------- -
jack               80 B
scott              73 C
```

Example 10.2h (Numbering each line)

```
CREATE TABLE student
(
Name VARCHAR2(10),
Class VARCHAR2(10)
);

INSERT INTO student VALUES ('abdul','philosophy');
INSERT INTO student VALUES ('bob','philosophy');
INSERT INTO student VALUES ('dole','philosophy');
INSERT INTO student VALUES ('jack','Religion');
INSERT INTO student VALUES ('kennedy','Religion');
INSERT INTO student VALUES ('jim','Science');
INSERT INTO student VALUES ('jones','Science');
INSERT INTO student VALUES ('harry','Science');
INSERT INTO student VALUES ('potter','Science');
```

/*Below is the desired result set that we are looking for. We want to get a count sequence for each of the different categories. Notice this is not like a group by in that we are not trying to suppress any information. We want to number our records. */

1	**abdul**	**philosophy**

2	**bob**	**philosophy**
3	**dole**	**philosophy**
1	jack	Religion
2	kennedy	Religion
1	**jim**	**Science**
2	**jones**	**Science**
3	**harry**	**Science**
4	**potter**	**Science**

/* Rownum is a pseudo-column that is available to us. It is a number that Oracle assigns to each record in the order in which the records were either physically or virtually inserted into the table. Create a table (temp1) with new rownumber. */

```
CREATE TABLE temp1 AS
SELECT rownum AS line_number, name, class
FROM student;
```

1	**abdul**	**philosophy**
2	**bob**	**philosophy**
3	**dole**	**philosophy**
4	jack	Religion
5	kennedy	Religion
6	**jim**	**Science**
7	**jones**	**Science**
8	**harry**	**Science**
9	**potter**	**Science**

--Create a table (temp2) that identifies the beginning point.

```
CREATE TABLE temp2 AS
SELECT min(rownum) AS beg_line_number, class
```

```
FROM student GROUP BY class
```

```
  beg_line_number      class

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

    6                        Science
    4                        Religion
    1                        Philosophy
```

Temp1			Temp2	
6	**jim**	**Science**	**6**	**Science**
7	**jones**	**Science**		
8	**harry**	**Science**		
9	**potter**	**Science**		
4	jack	Religion	**4**	**Religion**
5	kennedy	Religion		
1	**abdul**	**philosophy**	**1**	**philosophy**
2	**bob**	**philosophy**		
3	**dole**	**philosophy**		

```
SELECT (temp1.line_number - temp2.beg_line_number)+1 , name,  temp1.class
FROM temp1, temp2
WHERE  temp1.class=temp2.class
ORDER BY 3,1;
```

```
Num   Name             Class

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

1        abdul            philosophy
```

2	**bob**	**philosophy**
3	**dole**	**philosophy**
1	jack	Religion
2	kennedy	Religion
1	**jim**	**Science**
2	**jones**	**Science**
3	**harry**	**Science**
4	**potter**	**Science**

✓ *CHECK 10A*

1. Display all the people and all the potential personality types that they can have. Display name and personality description
   a. With and without CROSS JOIN
2. Display the name and personailty description of all those people who have a personality
   a. Use both the old and new Oracle syntax

"I am not young enough to know everything"

10.3 Self Join

Sometimes data in one column of a table has a relationship with another column in the same table.This type of join is known as a self- join.

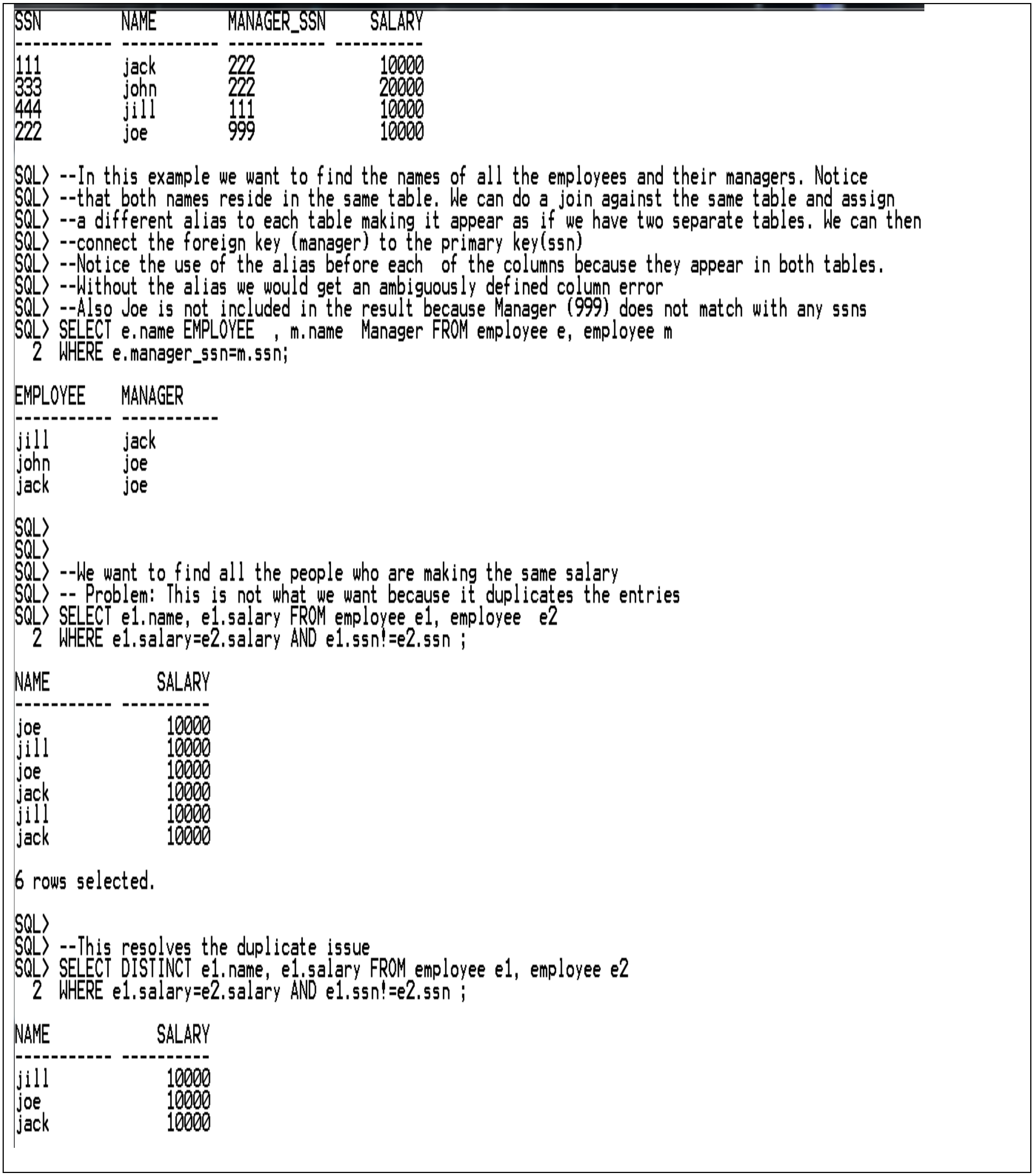
```
DROP TABLE employee;
CREATE TABLE employee
(
      ssn      VARCHAR2(11),
      name     VARCHAR2(11),
      manager  VARCHAR2(11),
      salary   NUMBER
);
INSERT INTO employee VALUES ('111','jack','222',10000);
INSERT INTO employee VALUES ('333','john','222',20000);
INSERT INTO employee VALUES ('444','jill','111',10000);
INSERT INTO employee VALUES ('222’,'joe','999',10000);
```

e	M

SSN	Name	Manager	Salary
111	jack	222	10000
333	john	222	20000
444	jill	111	10000
222	joe	999	10000

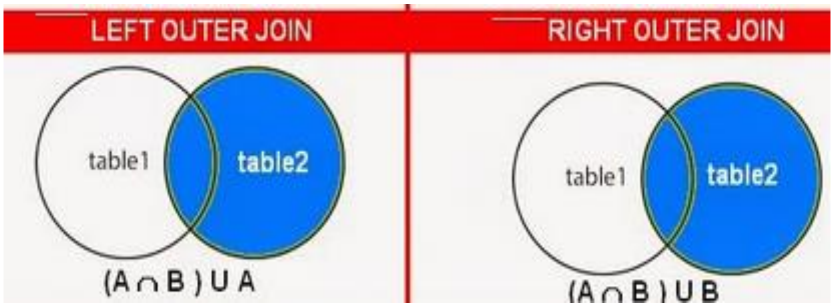
SSN	Name	Manager	Salary
111	jack	222	10000
333	john	222	20000
444	jill	111	10000
222	joe	999	10000

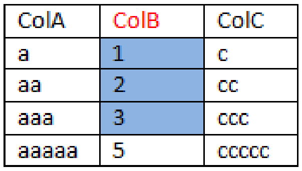
```
SELECT * FROM employee;
/*In this example, we want to find the names of all the employees and their managers. Notice that both names reside
in the same table. We can do a join against the same table and assign a different alias to each table making it
appear as if we have two separate tables. We can then connect the foreign key (manager) to the primary key (ssn).
Notice the use of the alias before each  of the columns because they appear in both tables. Without the alias we
would get an ambiguously defined column error. Also Joe is not included in the result because Manager (999) does
not match with any ssns. */
SELECT e.name EMPLOYEE, m.name  Manager FROM employee e, employee m
WHERE e.manager=m.ssn;


--We want to find all the people who are making the same salary.
-- Problem: This is not what we want because it duplicates the entries.
SELECT e1.name, e1.salary FROM employee e1, employee e2
WHERE e1.salary=e2.salary AND e1.ssn!=e2.ssn ;

--This resolves the duplicate issue.
SELECT DISTINCT e1.name, e1.salary FROM employee e1, employee e2
WHERE e1.salary=e2.salary AND e1.ssn!=e2.ssn ;
```



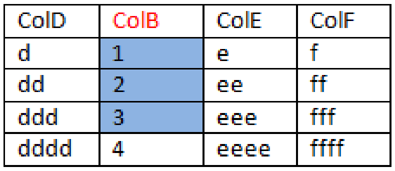
```
SSN         NAME        MANAGER_SSN     SALARY
----------- ----------- ----------- ----------
111         jack        222              10000
333         john        222              20000
444         jill        111              10000
222         joe         999              10000

SQL> --In this example we want to find the names of all the employees and their managers. Notice
SQL> --that both names reside in the same table. We can do a join against the same table and assign
SQL> --a different alias to each table making it appear as if we have two separate tables. We can then
SQL> --connect the foreign key (manager) to the primary key(ssn)
SQL> --Notice the use of the alias before each  of the columns because they appear in both tables.
SQL> --Without the alias we would get an ambiguously defined column error
SQL> --Also Joe is not included in the result because Manager (999) does not match with any ssns
SQL> SELECT e.name EMPLOYEE  , m.name  Manager FROM employee e, employee m
  2  WHERE e.manager_ssn=m.ssn;

EMPLOYEE    MANAGER
----------- -----------
jill        jack
john        joe
jack        joe

SQL>
SQL>
SQL> --We want to find all the people who are making the same salary
SQL> -- Problem: This is not what we want because it duplicates the entries
SQL> SELECT e1.name, e1.salary FROM employee e1, employee  e2
  2  WHERE e1.salary=e2.salary AND e1.ssn!=e2.ssn ;

NAME            SALARY
----------- ----------
joe              10000
jill             10000
joe              10000
jack             10000
jill             10000
jack             10000

6 rows selected.

SQL>
SQL> --This resolves the duplicate issue
SQL> SELECT DISTINCT e1.name, e1.salary FROM employee e1, employee e2
  2  WHERE e1.salary=e2.salary AND e1.ssn!=e2.ssn ;

NAME            SALARY
----------- ----------
jill             10000
joe              10000
jack             10000
```

10.4 Outer Join

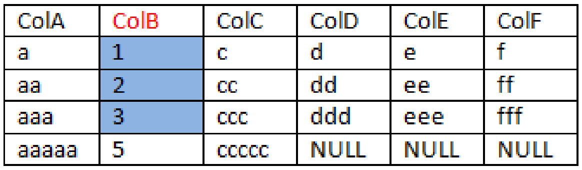




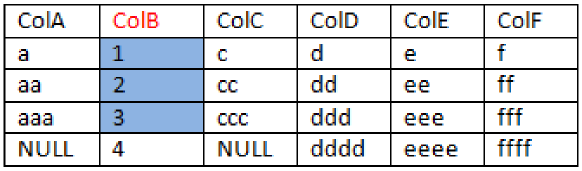
ColA	ColB	ColC
a	1	c
aa	2	cc
aaa	3	ccc
aaaaa	5	ccccc



ColD	ColB	ColE	ColF
d	1	e	f
dd	2	ee	ff
ddd	3	eee	fff
dddd	4	eeee	ffff



ColA	ColB	ColC	ColD	ColE	ColF
a	1	c	d	e	f
aa	2	cc	dd	ee	ff
aaa	3	ccc	ddd	eee	fff
aaaaa	5	ccccc	NULL	NULL	NULL



ColA	ColB	ColC	ColD	ColE	ColF
a	1	c	d	e	f
aa	2	cc	dd	ee	ff
aaa	3	ccc	ddd	eee	fff
NULL	4	NULL	dddd	eeee	ffff

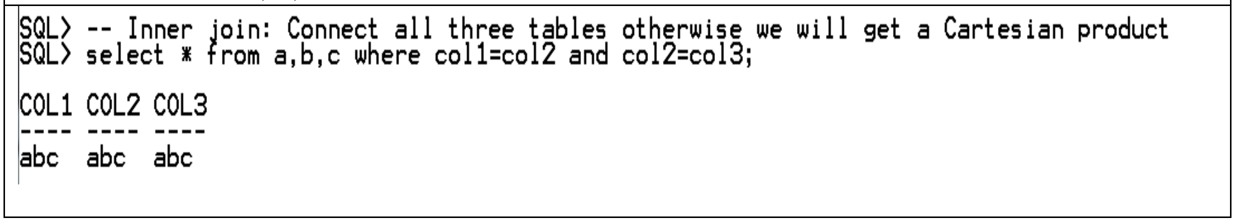
To tell Oracle to create NULL rows for records that don’t have a matching row, use an outer join operator, which looks like this: (+). It’s placed in the WHERE clause immediately after the column name from the table that’s missing the matching row. It tells Oracle to create a NULL row in that table to join with the row in the other table.You need to be aware of some limitations when using the traditional approach to outer joins: The outer join operator can be used for only one table in the joining condition. In other words, you can’t create NULL rows in both tables at the same time. A condition that includes the outer join operator can’t use the IN or OR operator.

```
drop table a;
drop table b;
drop table c;

create table a( col1 char(3) );
create table b( col2 char(3) );
create table c( col3 char(3) );

insert into a values ('a');
insert into a values ('ab');
insert into a values ('ac');
insert into a values ('abc');

insert into b values ('b');
insert into b values ('ab');
insert into b values ('bc');
insert into b values ('abc');

insert into c values ('c');
insert into c values ('bc');
insert into c values ('ac');
insert into c values ('abc');
```

A	B	C
COL1	COL2	COL3
-------	-------	-------
a	b	c
ab	ab	bc
ac	bc	ac
abc	abc	abc

Example 10.4a (Inner join)

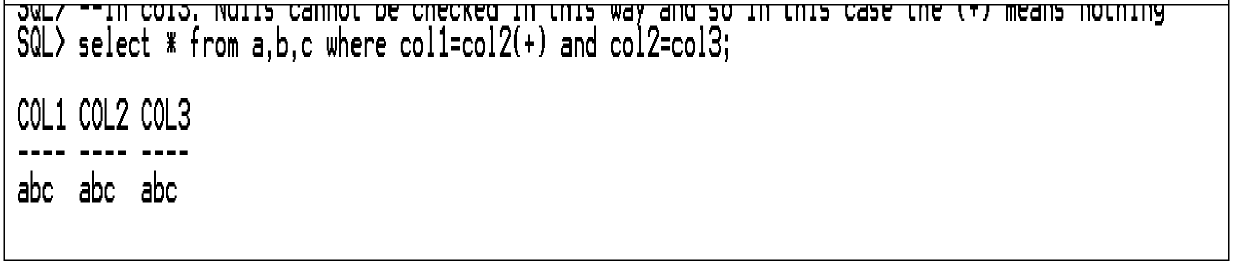
```
-- Inner join: Connect all three tables otherwise we will get a Cartesian product.
select * from a,b,c where col1=col2 and col2=col3;
```



```
SQL> -- Inner join: Connect all three tables otherwise we will get a Cartesian product
SQL> select * from a,b,c where col1=col2 and col2=col3;

COL1 COL2 COL3
---- ---- ----
abc  abc  abc
```

Example 10.4b (One outer join condition)

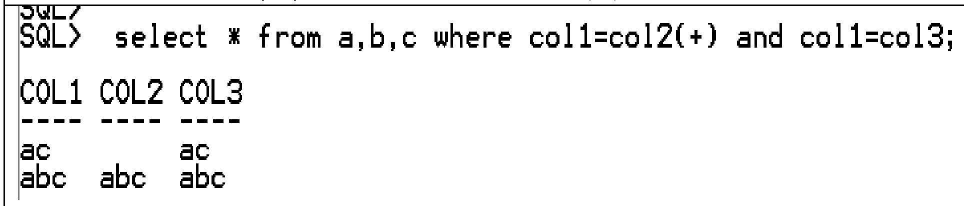
```
--The (+) is used for outer join which means (You don't really care). When you don't care,  it
--creates a virtual NULL record behind the scenes. An outer join is first and foremost an inner join.
--Then if there is a record for which there is not a match, the (+) says, it is okay and will allow it to
--go through.
--In this case, if there is something in col1 for which there is no match in col2, it will automatically
--create a NULL record in col2. The problem is that then the NULL record will be compared to the data
--in col3. NULLs cannot be checked in this way and so in this case the (+) means nothing.
select * from a,b,c where col1=col2(+) and col2=col3;
```



```
SQL> --in col3. Nulls cannot be checked in this way and so in this case the (+) means nothing
SQL> select * from a,b,c where col1=col2(+) and col2=col3;

COL1 COL2 COL3
---- ---- ----
abc  abc  abc
```

Example 10.4c (One outer join condition)

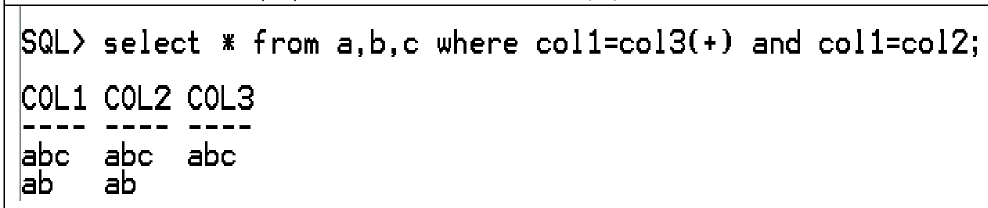
```
--In this example, we find all the common records to a, b and c. In addition, we find all the
--records that are common to (a) and (c).
--It checks (ac) against matches in col2 but it cannot find any. So it sets it to NULL because of (+). It then
--checks for (ac) in col3 and it finds a match.
select * from a,b,c where col1=col2(+) and col1=col3;
```



```
SQL>
SQL>  select * from a,b,c where col1=col2(+) and col1=col3;

COL1 COL2 COL3
---- ---- ----
ac        ac
abc  abc  abc
```

Example 10.4d (One outer join condition)

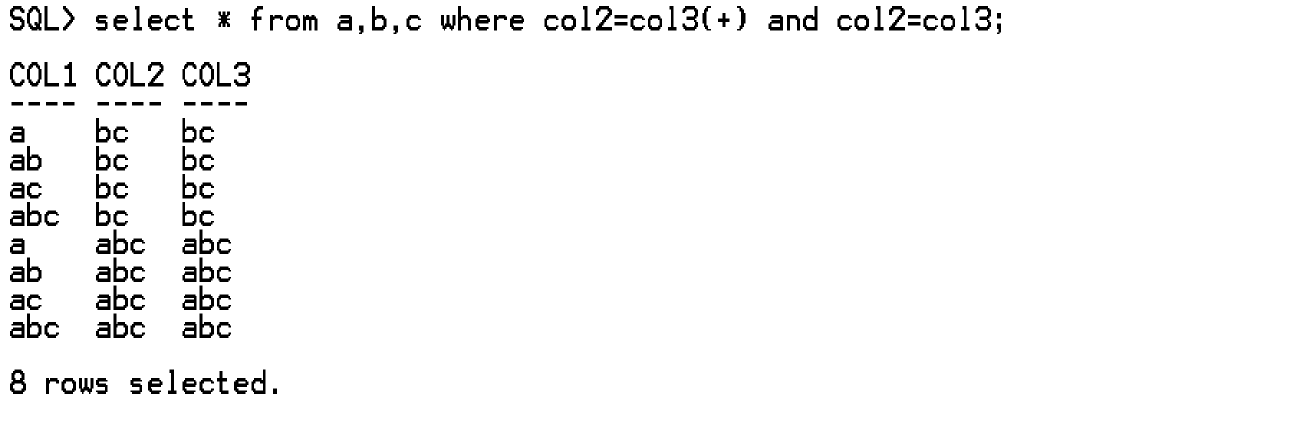
```
--Common to all and also to (a) and (b).
--It looks for (ab) in col3 but doesn't find a match. So it sets it to NULL. It checks (ab) to col2 and
--does find a match.
select * from a,b,c where col1=col3(+) and col1=col2;
```



```
SQL> select * from a,b,c where col1=col3(+) and col1=col2;

COL1 COL2 COL3
---- ---- ----
abc  abc  abc
ab   ab
```

Example 10.4e (One outer join condition)

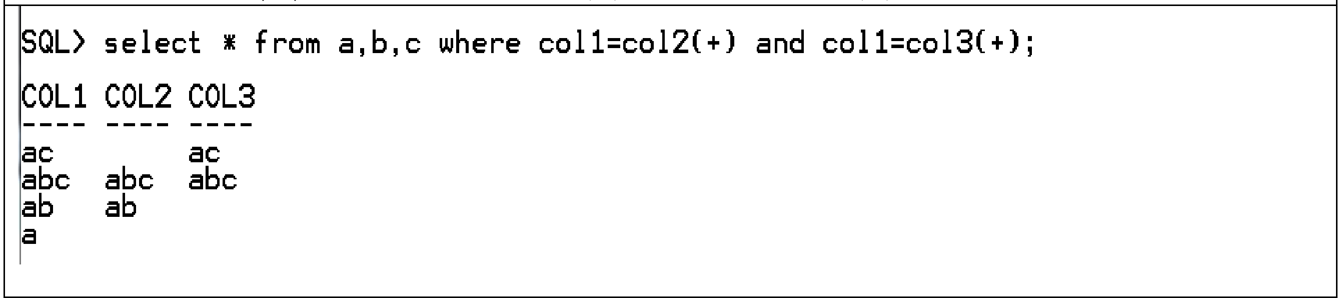
```
--Common to all and also to b and c.
--Notice that col1 is not included, which means that we will have a Cartesian product .
--(abc) in col2 will be found in col3 and also (bc) will be found in col3. The other records that are
--in col2, which are not in col3, will go through because of the (+). However, they would be ignored
--because it is looking to match that data with col3 again. The results (ab, abc) will be cross joined
-- with every record in table (a).
select * from a,b,c where col2=col3(+) and col2=col3;
```



```
SQL> select * from a,b,c where col2=col3(+) and col2=col3;

COL1 COL2 COL3
---- ---- ----
a    bc   bc
ab   bc   bc
ac   bc   bc
abc  bc   bc
a    abc  abc
ab   abc  abc
ac   abc  abc
abc  abc  abc

8 rows selected.
```

Example 10.4f (Two outer join conditions)

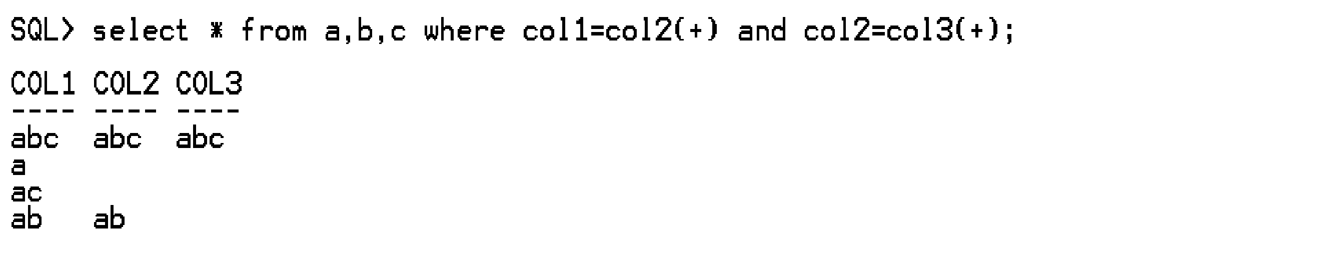
```
--All the intersecting points are displayed.
--If there is something that is col1 but not in col2, the (+) says it is okay. It then creates a NULL record.
--If there is something in col1 but not in col3, the (+) says it is okay.  It also creates a NULL record.
select * from a,b,c where col1=col2(+) and col1=col3(+);
```



```
SQL> select * from a,b,c where col1=col2(+) and col1=col3(+);

COL1 COL2 COL3
---- ---- ----
ac        ac
abc  abc  abc
ab   ab
a
```

Example 10.4g (Two outer join conditions)

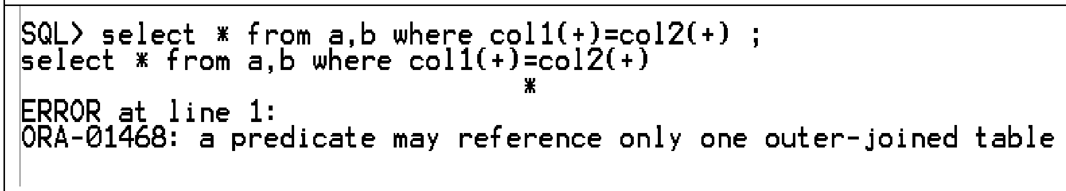
```
--Displays all the intersecting points.
--If there is something that is in col1 but not in col2, the (+) says it is okay and it creates a NULL record.
--If there is something in col2 but not in col3, the (+) says it is okay and also creates a NULL record.
select * from a,b,c where col1=col2(+) and col2=col3(+);
```



```
SQL> select * from a,b,c where col1=col2(+) and col2=col3(+);

COL1 COL2 COL3
---- ---- ----
abc  abc  abc
a
ac
ab   ab
```

Example 10.4h (Plus sign on only one side)

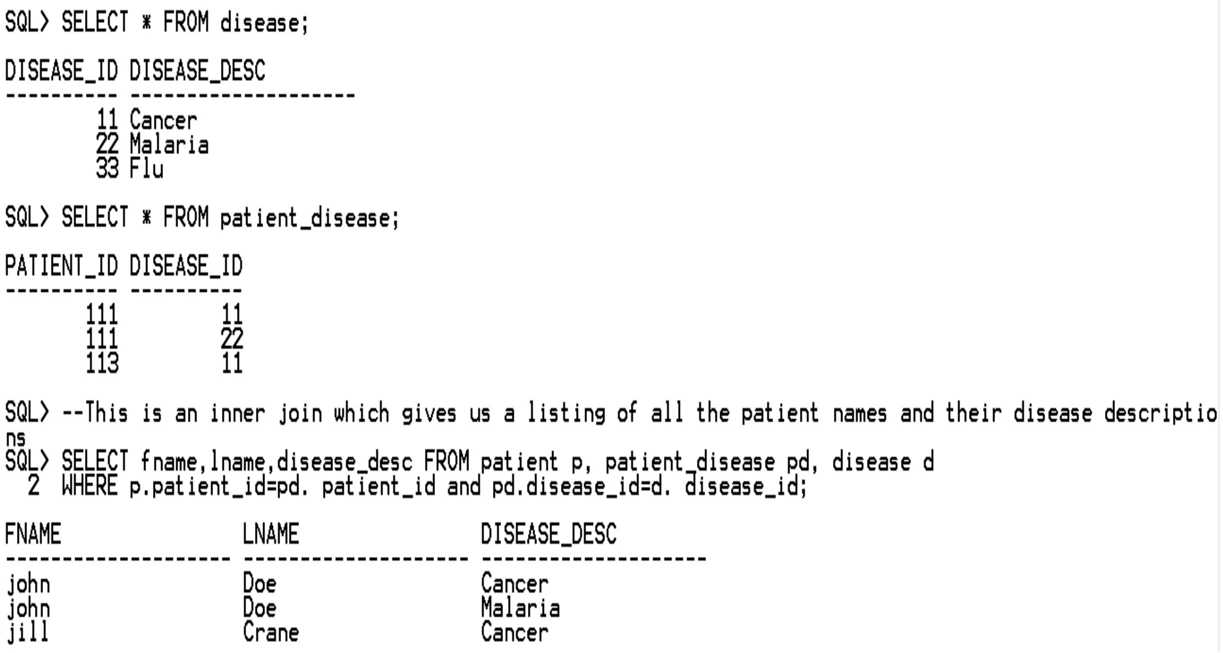
```
-- Invalid: Only one + sign can be used.
select * from a,b where col1(+)=col2(+) ;
```



```
SQL> select * from a,b where col1(+)=col2(+) ;
select * from a,b where col1(+)=col2(+)
                               *
ERROR at line 1:
ORA-01468: a predicate may reference only one outer-joined table
```

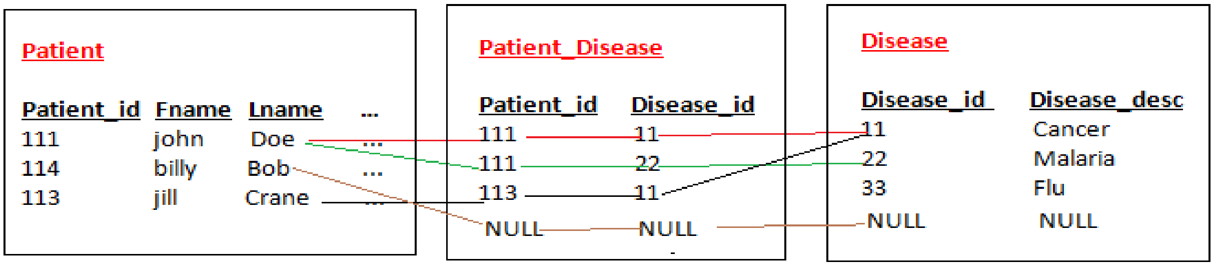
Example 10.4i (Inner join)

```
SELECT * FROM patient;
SELECT * FROM disease;
SELECT * FROM patient_disease;

--This is an inner join, which gives us a listing of all the patient names and their disease descriptions.
SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
WHERE p.patient_id=pd. patient_id and pd.disease_id=d. disease_id;
```



```
SQL> SELECT * FROM disease;

DISEASE_ID DISEASE_DESC
---------- --------------------
        11 Cancer
        22 Malaria
        33 Flu

SQL> SELECT * FROM patient_disease;

PATIENT_ID DISEASE_ID
---------- ----------
       111         11
       111         22
       113         11

SQL> --This is an inner join which gives us a listing of all the patient names and their disease descriptio
ns
SQL> SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
  2  WHERE p.patient_id=pd. patient_id and pd.disease_id=d. disease_id;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
```

+ +



Patient

Patient_id	Fname	Lname	...
111	john	Doe	...
114	billy	Bob	...
113	jill	Crane	...

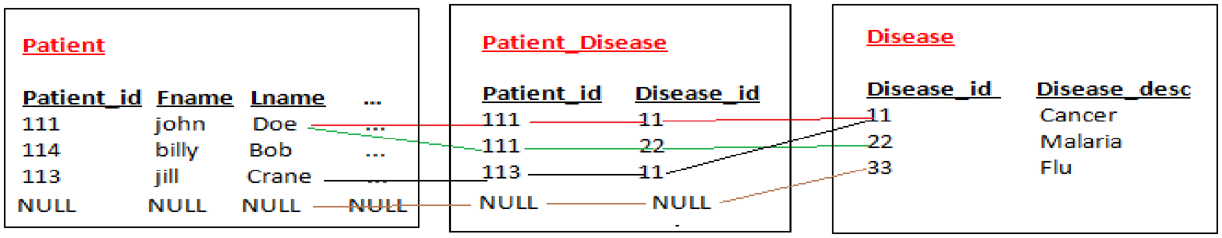
Patient_Disease

Patient_id	Disease_id
111	11
111	22
113	11
NULL	NULL

Disease

Disease_id	Disease_desc
11	Cancer
22	Malaria
33	Flu
NULL	NULL

+ +



Patient

Patient_id	Fname	Lname	...
111	john	Doe	...
114	billy	Bob	...
113	jill	Crane	...
NULL	NULL	NULL	NULL

Patient_Disease

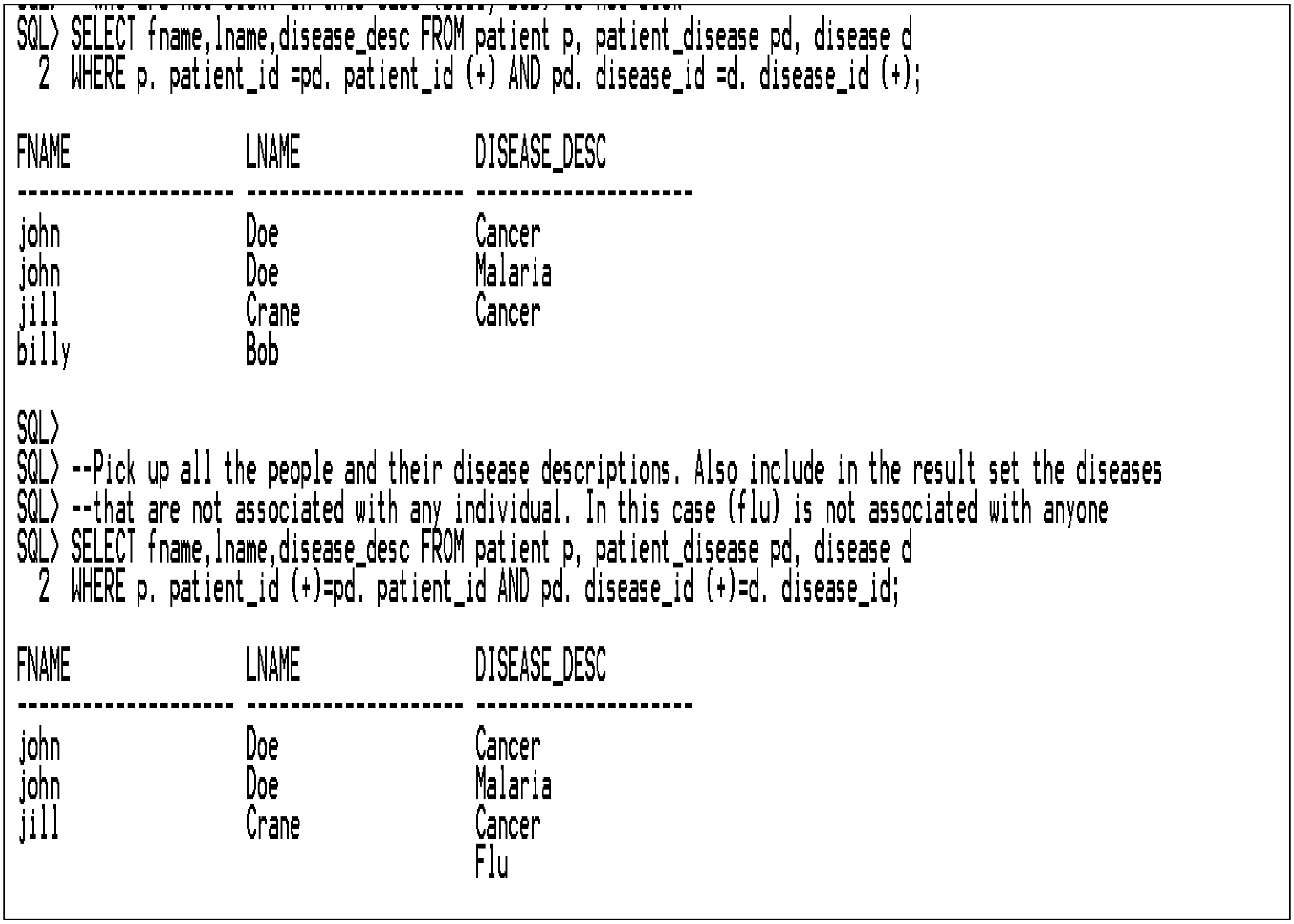
Patient_id	Disease_id
111	11
111	22
113	11
NULL	NULL

Disease

Disease_id	Disease_desc
11	Cancer
22	Malaria
33	Flu

Example 10.4j (Including records that don't match with anything else)

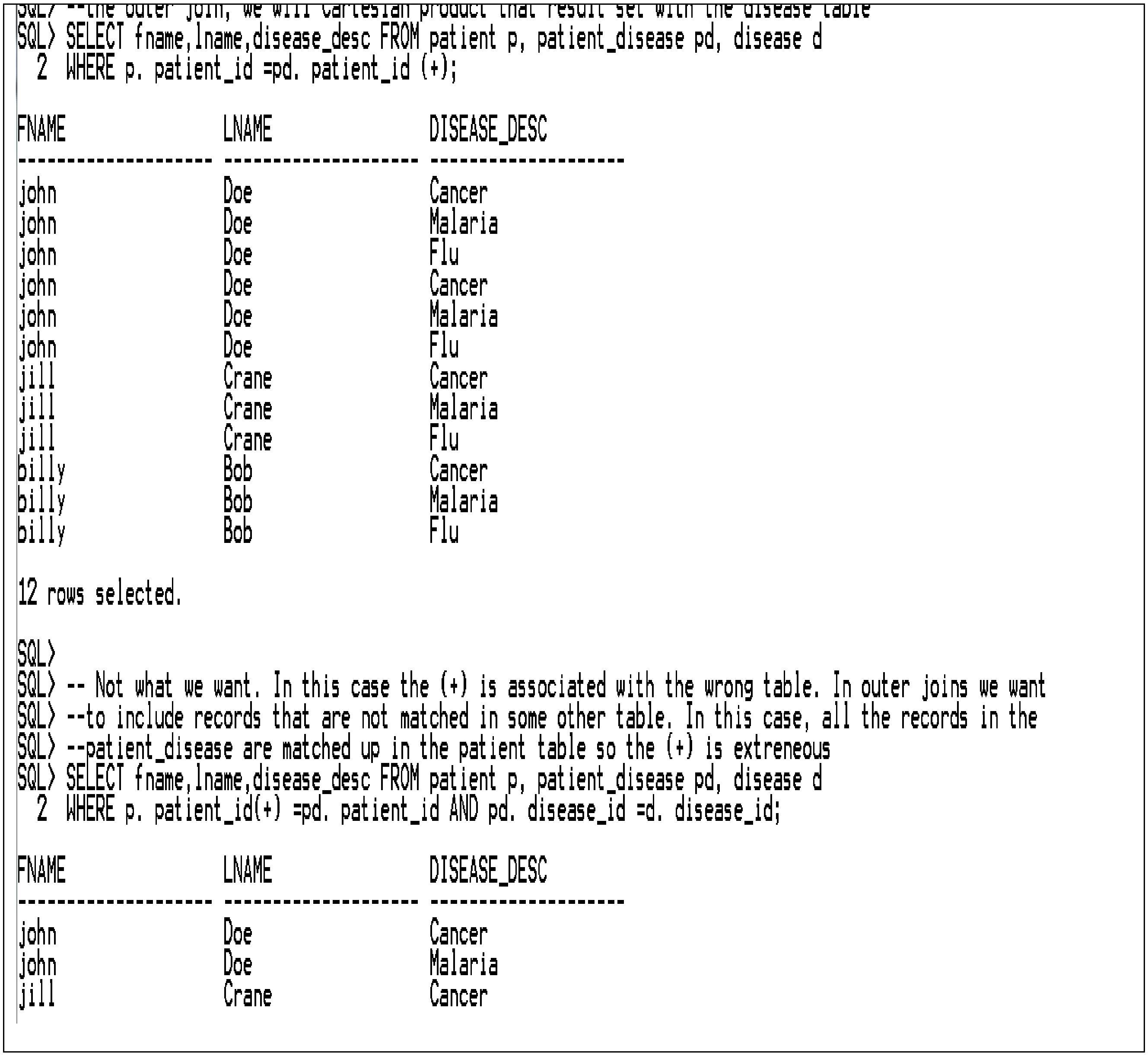
```
--Select all the people and their disease descriptions. Also, include in the result set the individuals
--who are not sick. In this case (billy bob) is not sick.
SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
WHERE p. patient_id =pd. patient_id (+) AND pd. disease_id =d. disease_id (+);


--Select all the people and their disease descriptions. Also, include in the result set the diseases
--that are not associated with any individual. In this case (flu) is not associated with anyone.
SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
WHERE p. patient_id (+)=pd. patient_id AND pd. disease_id (+)=d. disease_id;
```



```
SQL> SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
  2  WHERE p. patient_id =pd. patient_id (+) AND pd. disease_id =d. disease_id (+);

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
billy                Bob

SQL>
SQL> --Pick up all the people and their disease descriptions. Also include in the result set the diseases
SQL> --that are not associated with any individual. In this case (flu) is not associated with anyone
SQL> SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
  2  WHERE p. patient_id (+)=pd. patient_id AND pd. disease_id (+)=d. disease_id;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
                                          Flu
```

Example 10.4k (All join conditions must be included)

```
--This is not what we want. It is missing a join condition. Given the inner join between the patient and
--patient disease, along with the individuals who are not associated with any diseases, which is
--the outer join, we will Cartesian product the result set with the disease table.
SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
WHERE p. patient_id=pd. patient_id(+);


--This is not what we want. In this case the (+) is associated with the wrong table. In outer joins we want
--to include records that are not matched in some other table. In this case, all the records in the
--patient_disease are matched up in the patient table. The (+) is extreneous.
SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
WHERE p. patient_id(+)=pd. patient_id AND pd. disease_id =d. disease_id;
```

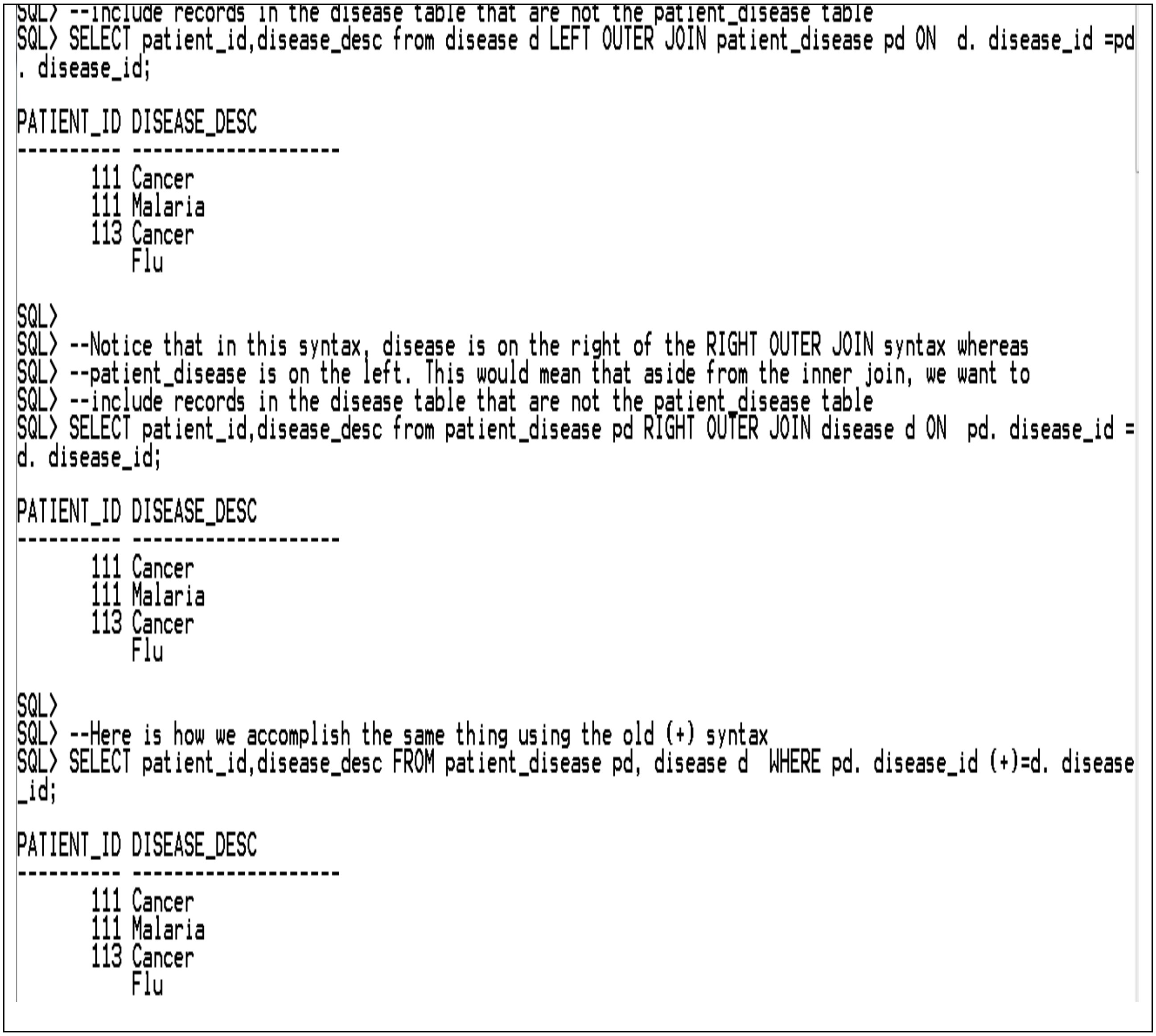


```
SQL> --the outer join, we will Cartesian product that result set with the disease table
SQL> SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
  2  WHERE p. patient_id =pd. patient_id (+);

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
john                 Doe                  Flu
john                 Doe                  Cancer
john                 Doe                  Malaria
john                 Doe                  Flu
jill                 Crane                Cancer
jill                 Crane                Malaria
jill                 Crane                Flu
billy                Bob                  Cancer
billy                Bob                  Malaria
billy                Bob                  Flu

12 rows selected.

SQL>
SQL> -- Not what we want. In this case the (+) is associated with the wrong table. In outer joins we want
SQL> --to include records that are not matched in some other table. In this case, all the records in the
SQL> --patient_disease are matched up in the patient table so the (+) is extreneous
SQL> SELECT fname,lname,disease_desc FROM patient p, patient_disease pd, disease d
  2  WHERE p. patient_id(+) =pd. patient_id AND pd. disease_id =d. disease_id;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
```

When creating a traditional outer join with the outer join operator, the join can be applied to only one table— not both. However, with the JOIN keyword, you can specify which table the join should be applied to by using a left, right, or full outer join. Left and right outer joins specify which table the outer join should be applied to, based on the table's location in the join condition. For example, a left outer join instructs Oracle to keep any rows in the table listed on the left side of the join condition, even if no matches are found with the table listed on the right. A full outer join keeps all rows from both tables in the results, no matter which table is deficient when matching rows. (That is, it performs a combination of left and right outer joins.)

Example 10.4L (Using join syntax)

```
--Can use the alternate syntax of LEFT or RIGHT OUTER JOIN to replace the old (+).
--Notice that in this syntax, disease is on the left of the LEFT OUTER JOIN syntax whereas
--patient_disease is on the right. This would mean that aside from the inner join, we want to
--include records in the disease table that are not the patient_disease table.
SELECT patient_id,disease_desc from disease d LEFT OUTER JOIN patient_disease
pd ON  d. disease_id =pd. disease_id;


--Notice that in this syntax, disease is on the right of the RIGHT OUTER JOIN syntax whereas
--patient_disease is on the left. This would mean that aside from the inner join, we want to
--include records in the disease table that are not in the patient_disease table.
SELECT patient_id,disease_desc from patient_disease pd RIGHT OUTER JOIN
disease d ON  pd. disease_id =d. disease_id;

--Here is how we accomplish the same thing using the old (+) syntax.
SELECT patient_id,disease_desc FROM patient_disease pd, disease d  WHERE pd.
disease_id (+)=d. disease_id;
```

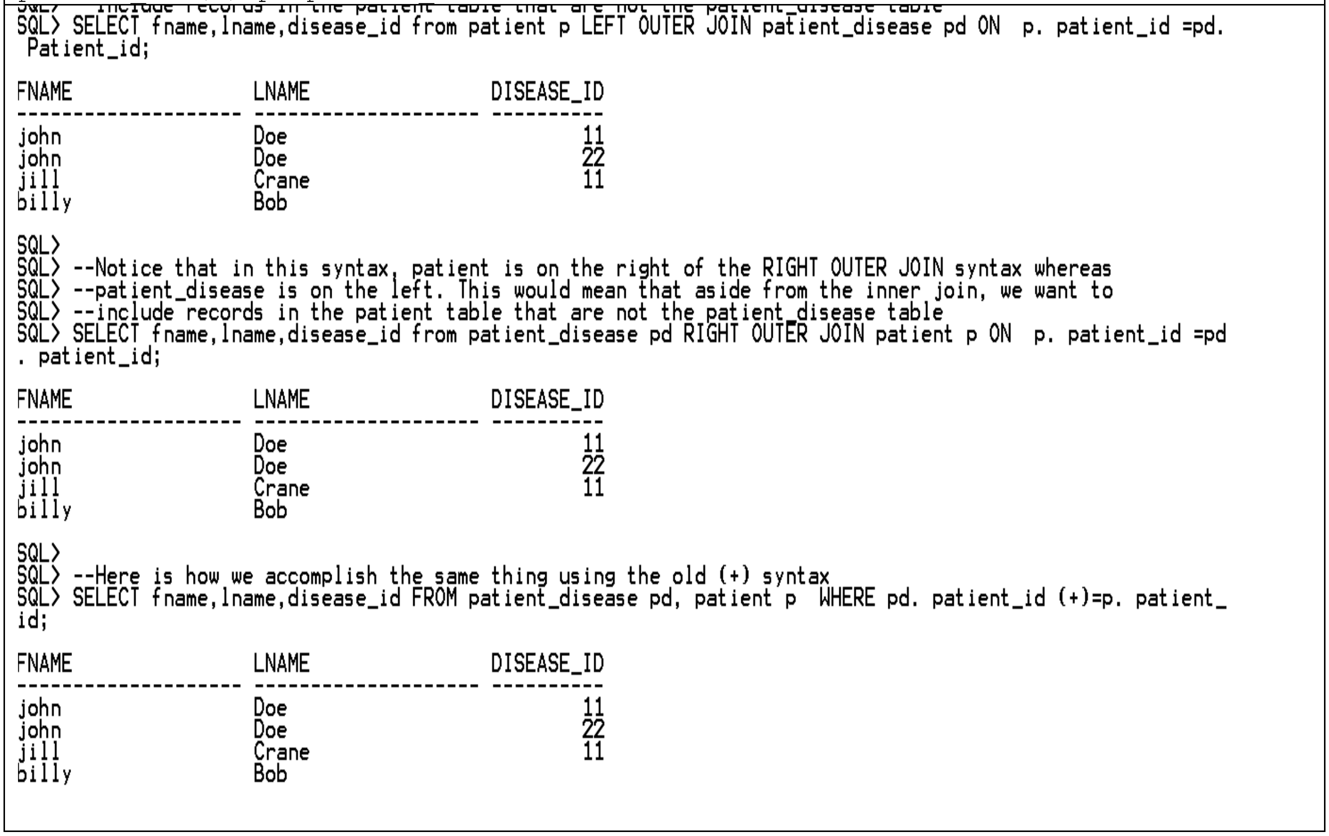


```
SQL> --include records in the disease table that are not the patient_disease table
SQL> SELECT patient_id,disease_desc from disease d LEFT OUTER JOIN patient_disease pd ON  d. disease_id =pd
. disease_id;

PATIENT_ID DISEASE_DESC
---------- --------------------
       111 Cancer
       111 Malaria
       113 Cancer
           Flu

SQL>
SQL> --Notice that in this syntax, disease is on the right of the RIGHT OUTER JOIN syntax whereas
SQL> --patient_disease is on the left. This would mean that aside from the inner join, we want to
SQL> --include records in the disease table that are not the patient_disease table
SQL> SELECT patient_id,disease_desc from patient_disease pd RIGHT OUTER JOIN disease d ON  pd. disease_id =
d. disease_id;

PATIENT_ID DISEASE_DESC
---------- --------------------
       111 Cancer
       111 Malaria
       113 Cancer
           Flu

SQL>
SQL> --Here is how we accomplish the same thing using the old (+) syntax
SQL> SELECT patient_id,disease_desc FROM patient_disease pd, disease d  WHERE pd. disease_id (+)=d. disease
_id;

PATIENT_ID DISEASE_DESC
---------- --------------------
       111 Cancer
       111 Malaria
       113 Cancer
           Flu
```

Example 10.4m (Left and right outer join syntax)

```
--Notice that in this syntax, patient is on the left of the LEFT OUTER JOIN syntax whereas
--patient_disease is on the right. This would mean that aside from the inner join, we want to
--include records in the patient table that are not the patient_disease table.
SELECT fname,lname,disease_idfrom patient p LEFT OUTER JOIN patient_disease pd
ON  p. patient_id =pd. Patient_id;


--Notice that in this syntax, patient is on the right of the RIGHT OUTER JOIN syntax whereas
--patient_disease is on the left. This would mean that aside from the inner join, we want to
--include records in the patient table that are not the patient_disease table.
SELECT fname,lname,disease_idfrom patient_disease pd RIGHT OUTER JOIN patient p
ON  p. patient_id=pd. patient_id;


--Here is how we accomplish the same thing using the old (+) syntax.
```

```
SELECT fname,lname,disease_idFROM patient_disease pd, patient p  WHERE pd.
patient_id (+)=p. patient_id;
```

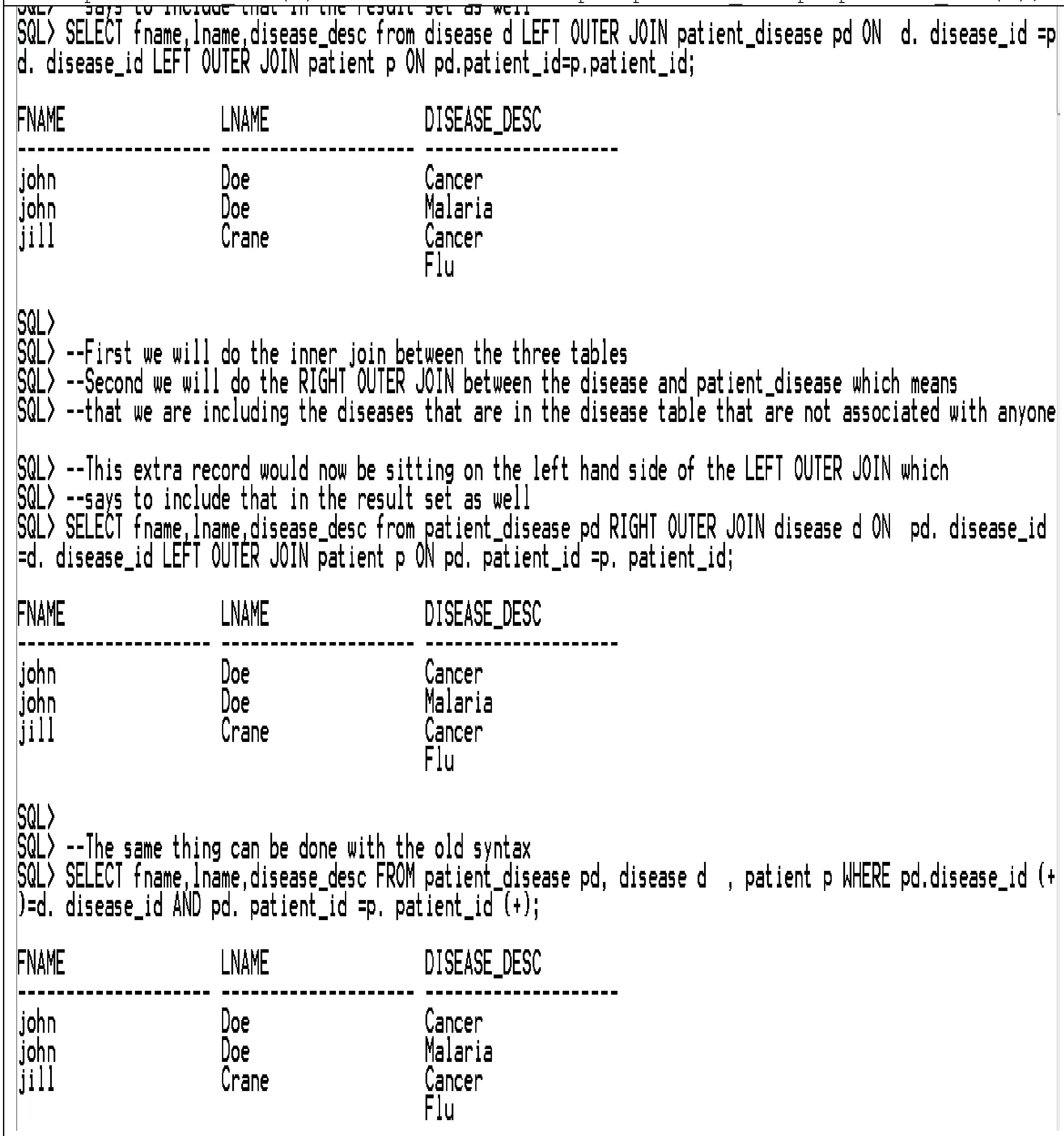


```
SQL> --include records in the patient table that are not the patient_disease table
SQL> SELECT fname,lname,disease_id from patient p LEFT OUTER JOIN patient_disease pd ON  p. patient_id =pd.
 Patient_id;

FNAME                LNAME                DISEASE_ID
-------------------- -------------------- ----------
john                 Doe                          11
john                 Doe                          22
jill                 Crane                        11
billy                Bob

SQL>
SQL> --Notice that in this syntax, patient is on the right of the RIGHT OUTER JOIN syntax whereas
SQL> --patient_disease is on the left. This would mean that aside from the inner join, we want to
SQL> --include records in the patient table that are not the patient_disease table
SQL> SELECT fname,lname,disease_id from patient_disease pd RIGHT OUTER JOIN patient p ON  p. patient_id =pd
. patient_id;

FNAME                LNAME                DISEASE_ID
-------------------- -------------------- ----------
john                 Doe                          11
john                 Doe                          22
jill                 Crane                        11
billy                Bob

SQL>
SQL> --Here is how we accomplish the same thing using the old (+) syntax
SQL> SELECT fname,lname,disease_id FROM patient_disease pd, patient p  WHERE pd. patient_id (+)=p. patient_
id;

FNAME                LNAME                DISEASE_ID
-------------------- -------------------- ----------
john                 Doe                          11
john                 Doe                          22
jill                 Crane                        11
billy                Bob
```

Example 10.4n (Left and right outer join syntax)

```
--First we will do the inner join between the three tables.
--Second we will do the LEFT OUTER JOIN between the disease and patient_disease which means
--that we are including the diseases that are in the disease table that are not associated with anyone.
--This extra record would now be sitting on the left hand side of the second LEFT OUTER JOIN which
--says to include that in the result set as well.
SELECT fname,lname,disease_desc from disease d LEFT OUTER JOIN patient_disease pd ON  d. disease_id
=pd.disease_id LEFT OUTER JOIN patient p ON pd.patient_id=p.patient_id;



--First we will do the inner join between the three tables.
--Second we will do the RIGHT OUTER JOIN between the disease and patient_disease which means
--that we are including the diseases that are in the disease table that are not associated with anyone.
--This extra record would now be sitting on the left hand side of the LEFT OUTER JOIN which
--says to include that in the result set as well.
SELECT fname,lname,disease_desc from patient_disease pd RIGHT OUTER JOIN disease
d ON  pd. disease_id=d. disease_idLEFT OUTER JOIN patient p ON pd. patient_id=p.
patient_id;



--The same thing can be done with the old syntax.
SELECT fname,lname,disease_desc FROM patient_disease pd, disease d  , patient p
```

```
WHEREpd.disease_id (+)=d. disease_id AND pd. patient_id =p. patient_id (+);
```



```
SQL> --says to include that in the result set as well
SQL> SELECT fname,lname,disease_desc from disease d LEFT OUTER JOIN patient_disease pd ON  d. disease_id =p
d. disease_id LEFT OUTER JOIN patient p ON pd.patient_id=p.patient_id;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
                                          Flu

SQL>
SQL> --First we will do the inner join between the three tables
SQL> --Second we will do the RIGHT OUTER JOIN between the disease and patient_disease which means
SQL> --that we are including the diseases that are in the disease table that are not associated with anyone

SQL> --This extra record would now be sitting on the left hand side of the LEFT OUTER JOIN which
SQL> --says to include that in the result set as well
SQL> SELECT fname,lname,disease_desc from patient_disease pd RIGHT OUTER JOIN disease d ON  pd. disease_id
=d. disease_id LEFT OUTER JOIN patient p ON pd. patient_id =p. patient_id;

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
                                          Flu

SQL>
SQL> --The same thing can be done with the old syntax
SQL> SELECT fname,lname,disease_desc FROM patient_disease pd, disease d  , patient p WHERE pd.disease_id (+
)=d. disease_id AND pd. patient_id =p. patient_id (+);

FNAME                LNAME                DISEASE_DESC
-------------------- -------------------- --------------------
john                 Doe                  Cancer
john                 Doe                  Malaria
jill                 Crane                Cancer
                                          Flu
```

Example 10.4o (Full outer join conditions)

ColA	ColB	ColC
a	1	c
aa	2	cc
aaa	3	ccc
aaaaa	5	ccccc

ColD	ColB	ColE	ColF
d	1	e	f
dd	2	ee	ff
ddd	3	eee	fff
dddd	4	eeee	ffff

Inner Join

ColA	ColB	ColC	ColD	ColE	ColF
a	1	c	d	e	f
aa	2	cc	dd	ee	ff
aaa	3	ccc	ddd	eee	fff

Left Outer Join

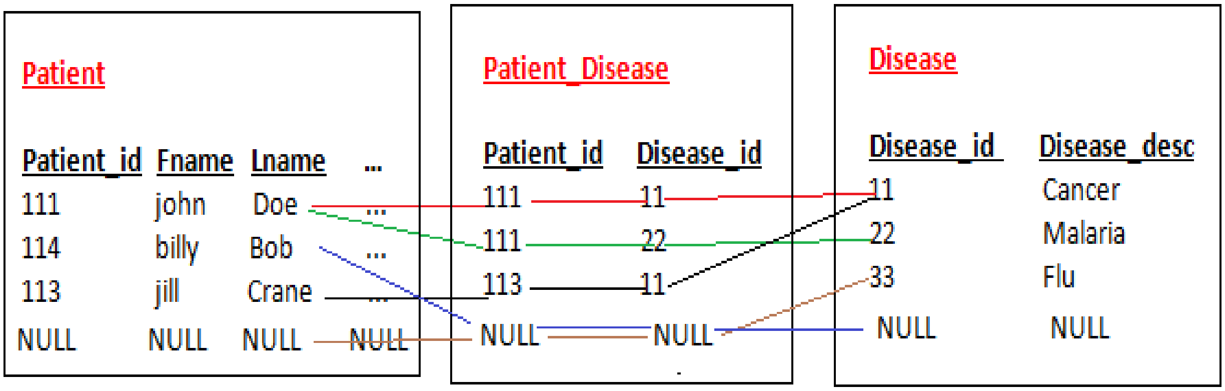
ColA	ColB	ColC	ColD	ColE	ColF
a	1	c	d	e	f
aa	2	cc	dd	ee	ff
aaa	3	ccc	ddd	eee	fff
aaaaa	5	ccccc	NULL	NULL	NULL

Right Outer Join

ColA	ColB	ColC	ColD	ColE	ColF
a	1	c	d	e	f
aa	2	cc	dd	ee	ff
aaa	3	ccc	ddd	eee	fff
NULL	4	NULL	dddd	eeee	ffff

Full Outer Join

ColA	ColB	ColC	ColD	ColE	ColF
a	1	c	d	e	f
aa	2	cc	dd	ee	ff
aaa	3	ccc	ddd	eee	fff
NULL	4	NULL	dddd	eeee	ffff
aaaaa	5	ccccc	NULL	NULL	NULL



Patient

Patient_id	Fname	Lname	...
111	john	Doe	...
114	billy	Bob	...
113	jill	Crane	...
NULL	NULL	NULL	NULL

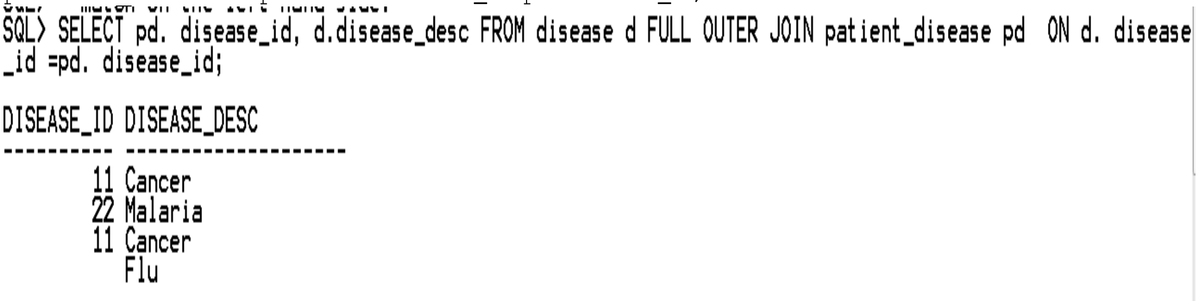
Patient_Disease

Patient_id	Disease_id
111	11
111	22
113	11
NULL	NULL

Disease

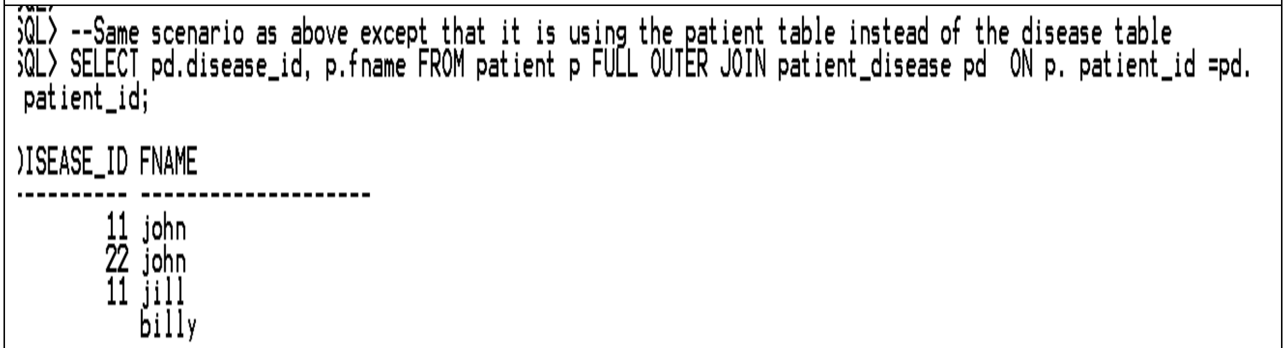
Disease_id	Disease_desc
11	Cancer
22	Malaria
33	Flu
NULL	NULL

```
--Full outer is equivalent to a LEFT and RIGHT OUTER JOIN at the same time. This means that
--in addition to an inner join, include the stuff on the left hand side for which there is no match
--no the right hand side. Also include the stuff on the right hand side for which there is no
--match on the left hand side.
SELECT pd. disease_id, d.disease_desc FROM disease d FULL OUTER JOIN
patient_disease pd ON d. disease_id=pd. disease_id;
```



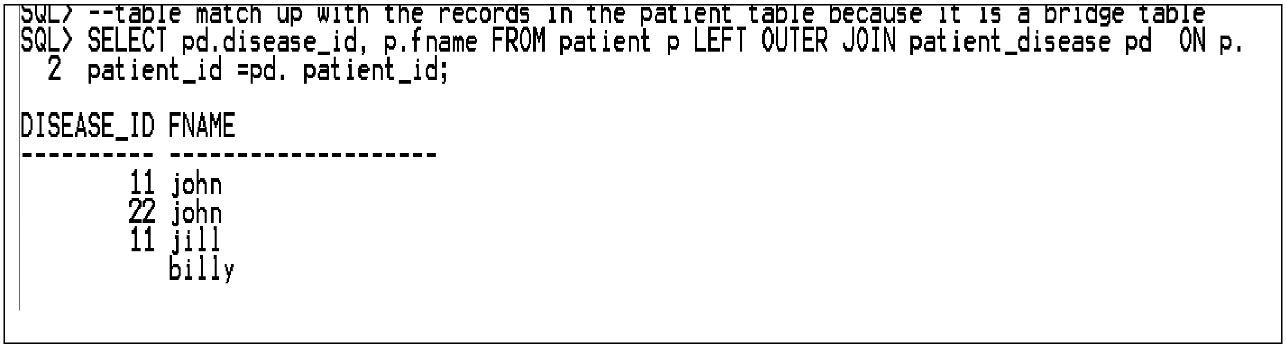
```
SQL> SELECT pd. disease_id, d.disease_desc FROM disease d FULL OUTER JOIN patient_disease pd  ON d. disease
_id =pd. disease_id;

DISEASE_ID DISEASE_DESC
---------- --------------------
        11 Cancer
        22 Malaria
        11 Cancer
           Flu
```

```
--This is the same as above because all the records that are in the patient_disease table are matched
--against the records in the disease table because it is a bridge table.
--SELECT pd. disease_id, d.disease_desc FROM disease d LEFT OUTER JOIN patient_disease pd  ON
--d. disease_id =pd. disease_id;

--Same scenario as above except that it is using the patient table instead of the disease table.
SELECT pd.disease_id, p.fname FROM patient p FULL OUTER JOIN patient_disease
pd  ON p. patient_id =pd. patient_id;
```



```
SQL> --Same scenario as above except that it is using the patient table instead of the disease table
SQL> SELECT pd.disease_id, p.fname FROM patient p FULL OUTER JOIN patient_disease pd  ON p. patient_id =pd.
patient_id;

DISEASE_ID FNAME
---------- --------------------
        11 john
        22 john
        11 jill
           billy
```

```
--This is the same as above. Don't need the FULL OUTER JOIN because all the records in the patient_disease
--table match up with the records in the patient table because it is a bridge table.
SELECT pd.disease_id, p.fname FROM patient p LEFT OUTER JOIN patient_disease
pd  ON p.patient_id =pd. patient_id;
```



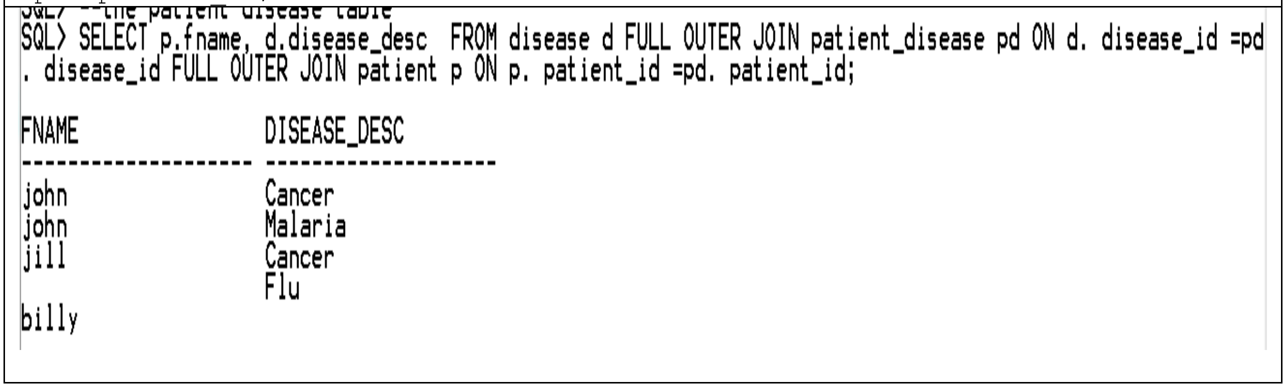
```
SQL> --table match up with the records in the patient table because it is a bridge table
SQL> SELECT pd.disease_id, p.fname FROM patient p LEFT OUTER JOIN patient_disease pd  ON p.
  2  patient_id =pd. patient_id;

DISEASE_ID FNAME
---------- --------------------
        11 john
        22 john
        11 jill
           billy
```

--Include all the data from the disease table in the FULL OUTER JOIN. Take that result set which
--includes the common records and the orphan records in the disease table and include them with
--all the records in the patient table, even the records in the patient table that don't match with
--the patient disease table.

```
SELECT p.fname, d.disease_desc  FROM disease d FULL OUTER JOIN patient_disease
pd ON d. disease_id =pd. disease_id FULL OUTER JOIN patient p ON p. patient_id
=pd. patient_id;
```

--This is the same as above. Because the patient_diseaes table is a bridge table, we can do a LEFT OUTER JOIN
--to include all the stuff from the disease table. Next, we can do a full outer join to include those
--results along with all the records from the patient table.

```
SELECT p.fname, d.disease_desc  FROM disease d LEFT OUTER JOIN patient_disease
pd ON d.disease_id =pd. disease_id FULL OUTER JOIN patient p ON p. patient_id
=pd. patient_id;
```



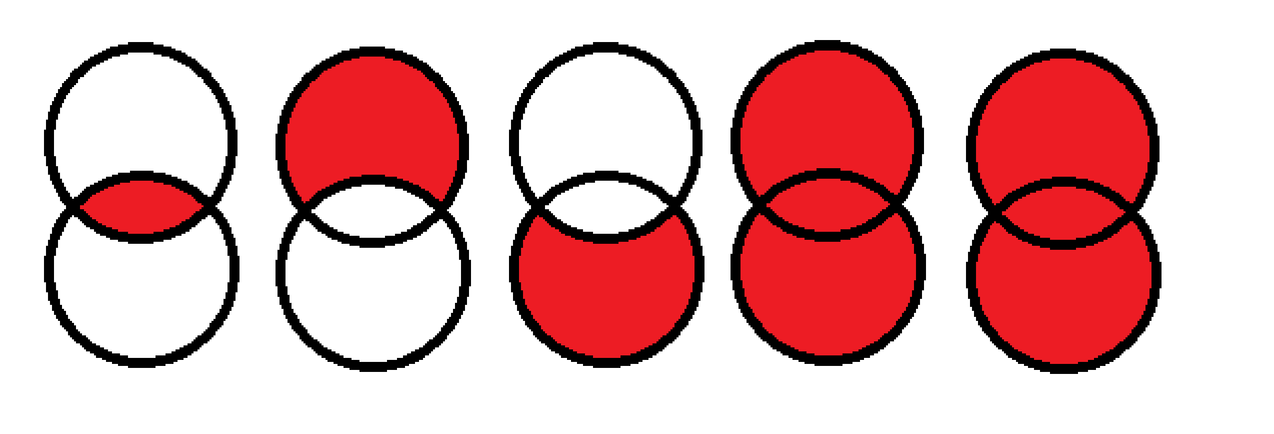
```
SQL> --the patient disease table
SQL> SELECT p.fname, d.disease_desc  FROM disease d FULL OUTER JOIN patient_disease pd ON d. disease_id =pd
. disease_id FULL OUTER JOIN patient p ON p. patient_id =pd. patient_id;

FNAME                DISEASE_DESC
-------------------- --------------------
john                 Cancer
john                 Malaria
jill                 Cancer
                     Flu
billy
```

✓ *<u>CHECK 10B</u>*

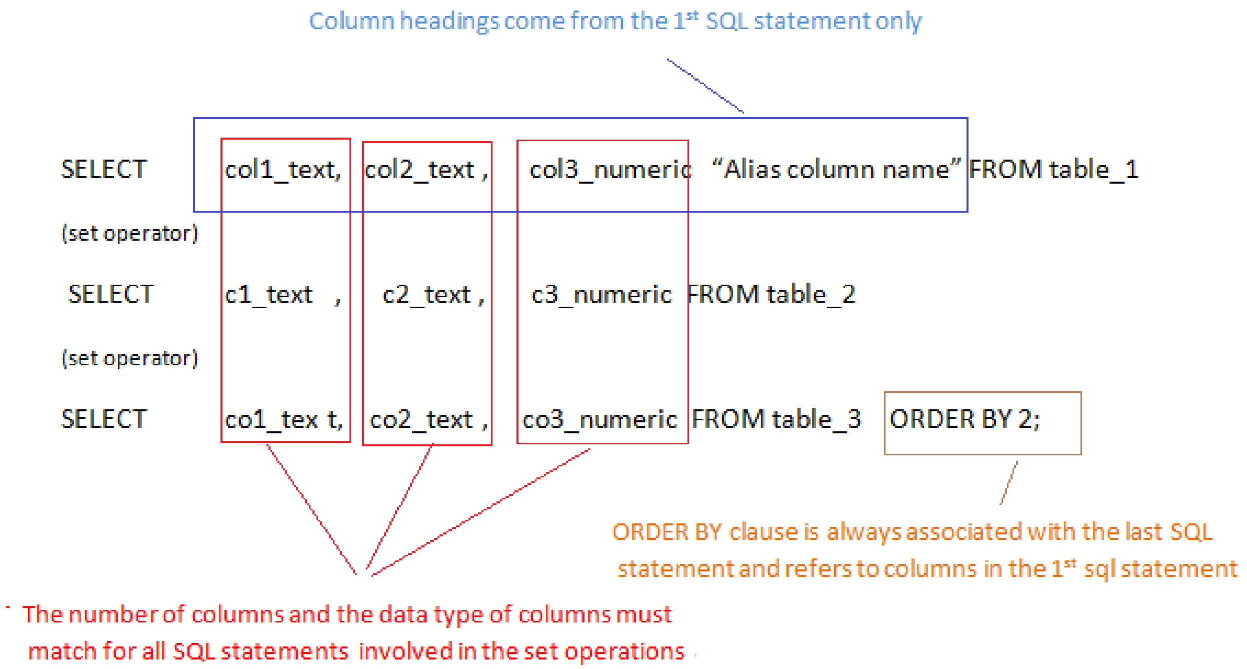
1. Display name and description of all those who have a personality and also those who don't. For those who don't have a personality, display "Bland" for description.
   a. Use LEFT, RIGHT and + operator syntax
2. Display the name and description of all those who have a personality, all those who don't have a personality and all personalities that are not associated with anyone. (Display fname, lname, personality description)

“Some drink deeply from the river of knowledge. Others only gargle”

10.5 Set Operators



top INTERSECT bottom	top MINUS bottom	bottom MINUS top	top UNION bottom (No duplicates)	top UNIONALL bottom (Includes duplicates)



Column headings come from the 1st SQL statement only

```
SELECT           col1_text,  col2_text ,   col3_numeric  "Alias column name" FROM table_1
(set operator)
SELECT           c1_text   ,  c2_text ,    c3_numeric  FROM table_2
(set operator)
SELECT           co1_tex t,  co2_text ,   co3_numeric  FROM table_3   ORDER BY 2;
```

ORDER BY clause is always associated with the last SQL statement and refers to columns in the 1st sql statement

- The number of columns and the data type of columns must match for all SQL statements involved in the set operations

Set operators are used to combine the results of two (or more) SELECT statements. Valid set operators in Oracle are UNION, UNION ALL, INTERSECT, and MINUS. When used with two SELECT statements, the UNION set operator returns the results of both queries. However, if there are any duplicates, they are removed, and the duplicated record is listed only once. To include duplicates in the results, use the UNION ALL set operator. INTERSECT lists only records that are returned by both queries; the MINUS set operator removes the second query's results from the output if they are also found in the first query's results. INTERSECT and MINUS set operations produce unduplicated results.

UNION	Returns the results of both queries and removes duplicates
UNION ALL	Returns the results of both queries but includes duplicates
INTERSECT	Returns only the rows included in the results of both queries
MINUS	Subtracts the second query's results if they're also returned in the first query's result

Keep in mind some guidelines for multiple- column set operations: All columns are included to perform the set comparison. Each query must contain the same number of columns, which are compared positionally. Column names can be different in the queries.

```
CREATE TABLE hourly
(
    Lname          VARCHAR2(20),
    Hrly_wage      NUMBER
);

INSERT INTO hourly VALUES ('Smith',10.25);
INSERT INTO hourly VALUES ('Wesseon',30.50);
INSERT INTO hourly VALUES ('Smith',10.25);
INSERT INTO hourly VALUES ('Wesseon',30.50);

CREATE TABLE salaried
(
   Lname    VARCHAR2(20),
   salary   NUMBER
);

INSERT INTO salaried VALUES ('Smith',500);
INSERT INTO salaried VALUES ('Jones',600);
```

Hourly

LnameHrly_wage

Smith   10.25

Wesseon   30.50

Smith   10.25

Wesseon   30.50

Salaried

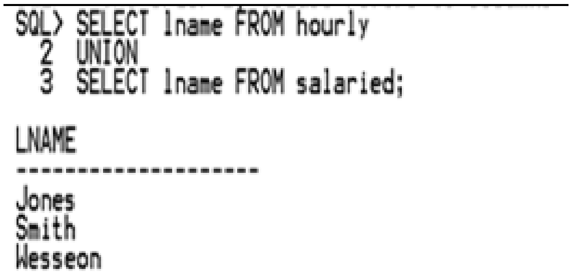
LnameSalary

Smith   500

```
Jones        600
```

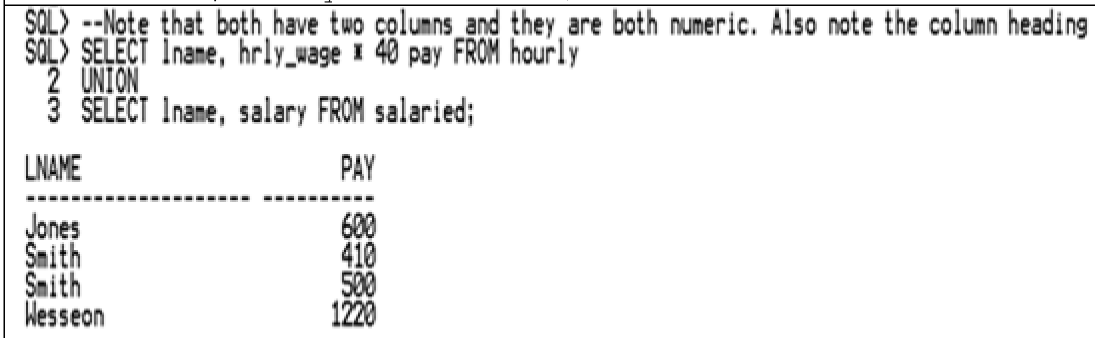
Example 10.5a (UNION)

```
--Appends the result from the second query to the first query. Make sure the number of
--columns in both queries are the same. Also their types must be the same. The column
--headings come from the first query and ORDER BY clause appears at the end of the last query.
--The order by clause refers to columns from the first query
SELECT lname FROM hourly
UNION
SELECT lname FROM salaried;
```



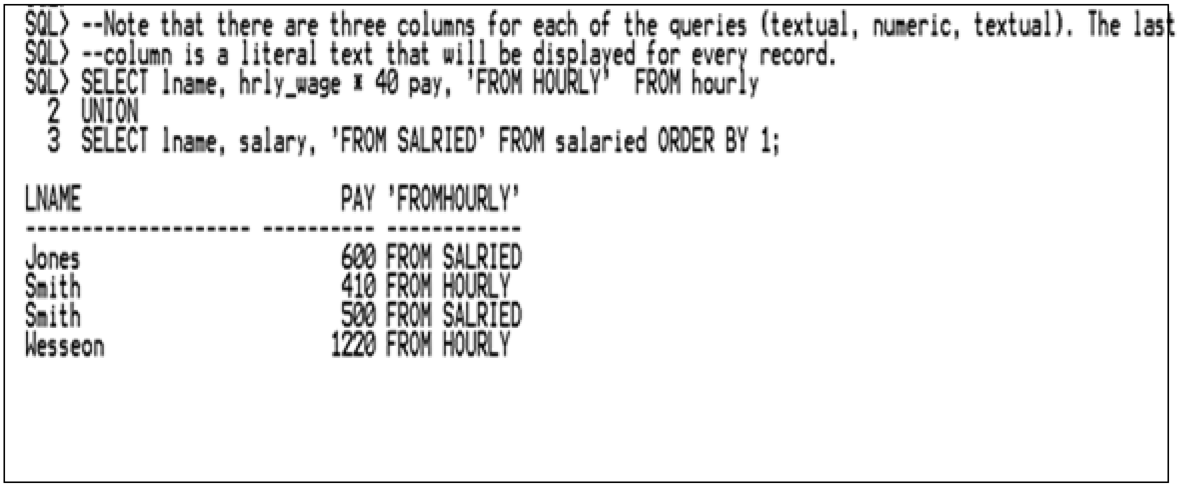
```
SQL> SELECT lname FROM hourly
  2  UNION
  3  SELECT lname FROM salaried;

LNAME
--------------------
Jones
Smith
Wesseon
```

```
--Note that both have two columns and they are both numeric. Also note the column heading
SELECT lname, hrly_wage * 40 pay FROM hourly
UNION
SELECT lname, salary FROM salaried;
```



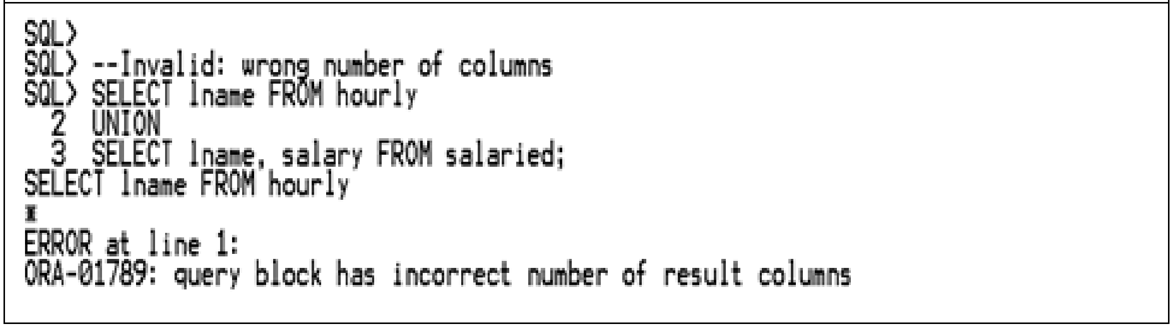
```
SQL> --Note that both have two columns and they are both numeric. Also note the column heading
SQL> SELECT lname, hrly_wage * 40 pay FROM hourly
  2  UNION
  3  SELECT lname, salary FROM salaried;

LNAME                       PAY
-------------------- ----------
Jones                       600
Smith                       410
Smith                       500
Wesseon                    1220
```

```
--Note that there are three columns for each of the queries (textual, numeric, textual). The last
--column is a literal text that will be displayed for every record.
SELECT lname, hrly_wage * 40 pay, 'FROM HOURLY'  FROM hourly
UNION
SELECT lname, salary, 'FROM SALRIED' FROM salaried ORDER BY 1;
```



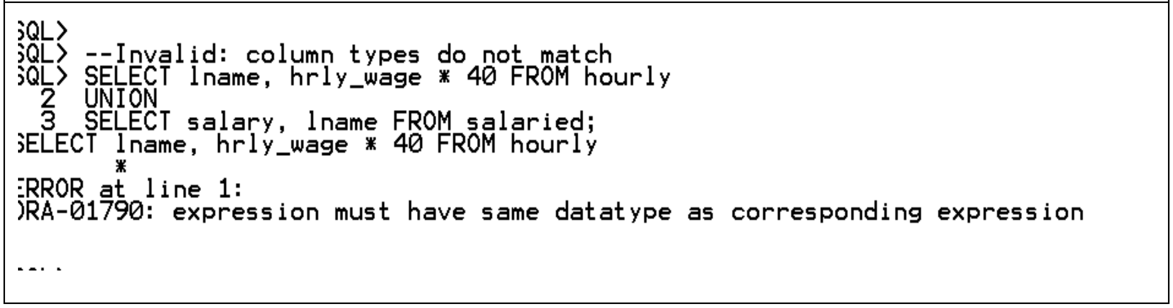
```
SQL> --Note that there are three columns for each of the queries (textual, numeric, textual). The last
SQL> --column is a literal text that will be displayed for every record.
SQL> SELECT lname, hrly_wage * 40 pay, 'FROM HOURLY'  FROM hourly
  2  UNION
  3  SELECT lname, salary, 'FROM SALRIED' FROM salaried ORDER BY 1;

LNAME                       PAY 'FROMHOURLY'
-------------------- ---------- ------------
Jones                       600 FROM SALRIED
Smith                       410 FROM HOURLY
Smith                       500 FROM SALRIED
Wesseon                    1220 FROM HOURLY
```

```
--Invalid: wrong number of columns
SELECT lname FROM hourly
UNION
SELECT lname, salary FROM salaried;
```



```
SQL>
SQL> --Invalid: wrong number of columns
SQL> SELECT lname FROM hourly
  2  UNION
  3  SELECT lname, salary FROM salaried;
SELECT lname FROM hourly
*
ERROR at line 1:
ORA-01789: query block has incorrect number of result columns
```

```
--Invalid: column types do not match
SELECT lname, hrly_wage * 40 FROM hourly
UNION
SELECT salary, lname FROM salaried;
```

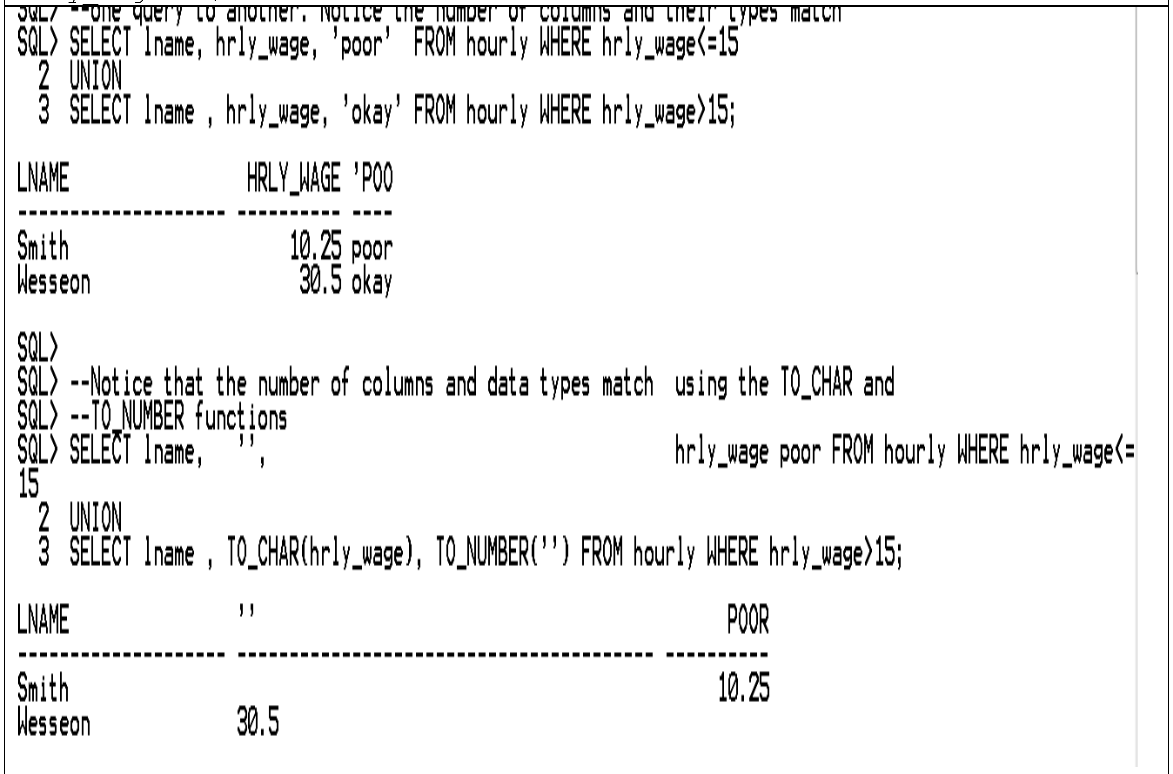


```
SQL>
SQL> --Invalid: column types do not match
SQL> SELECT lname, hrly_wage * 40 FROM hourly
  2  UNION
  3  SELECT salary, lname FROM salaried;
SELECT lname, hrly_wage * 40 FROM hourly
              *
ERROR at line 1:
ORA-01790: expression must have same datatype as corresponding expression
```

Example 10.5b (UNION)

```
--Instead of doing DECODE or a CASE statement we can just append the results of
```

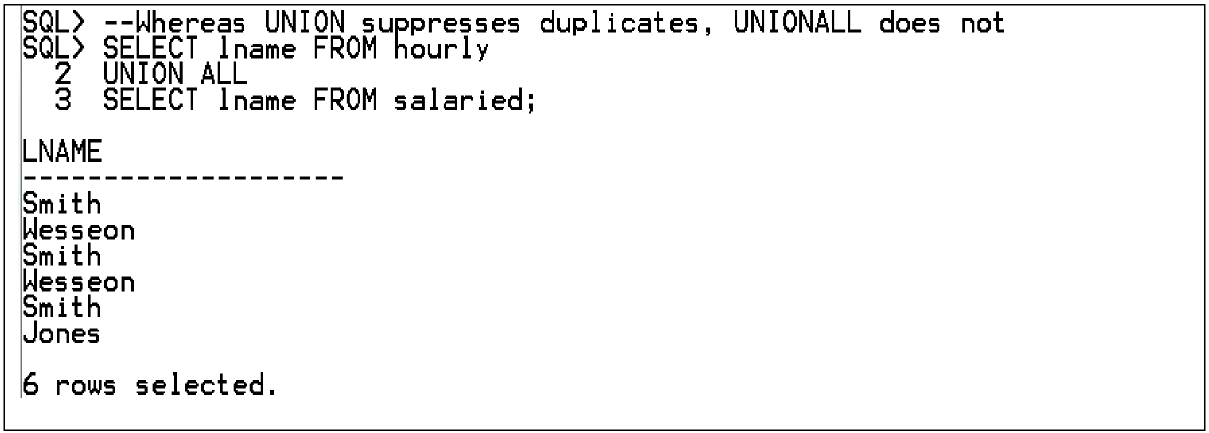
```
--one query to another. Notice the number of columns and their types match.
SELECT lname, hrly_wage, 'poor'  FROM hourly WHERE hrly_wage<=15
UNION
SELECT lname , hrly_wage, 'okay' FROM hourly WHERE hrly_wage>15;


--Notice that the number of columns and data types match using the TO_CHAR and
--TO_NUMBER  functions.
SELECT lname,'', hrly_wage poor FROM hourly WHERE hrly_wage<=15
UNION
SELECT lname , TO_CHAR(hrly_wage), TO_NUMBER('') FROM hourly WHERE
hrly_wage>15;
```



```
SQL> --one query to another. Notice the number of columns and their types match
SQL> SELECT lname, hrly_wage, 'poor'  FROM hourly WHERE hrly_wage<=15
  2  UNION
  3  SELECT lname , hrly_wage, 'okay' FROM hourly WHERE hrly_wage>15;

LNAME                HRLY_WAGE 'POO
-------------------- ---------- ----
Smith                    10.25 poor
Wesseon                   30.5 okay

SQL>
SQL> --Notice that the number of columns and data types match  using the TO_CHAR and
SQL> --TO_NUMBER functions
SQL> SELECT lname,   '',                              hrly_wage poor FROM hourly WHERE hrly_wage<=
15
  2  UNION
  3  SELECT lname , TO_CHAR(hrly_wage), TO_NUMBER('') FROM hourly WHERE hrly_wage>15;

LNAME                ''                                             POOR
-------------------- ---------------------------------------- ----------
Smith                                                              10.25
Wesseon              30.5
```

Example 10.5c (UNION ALL)

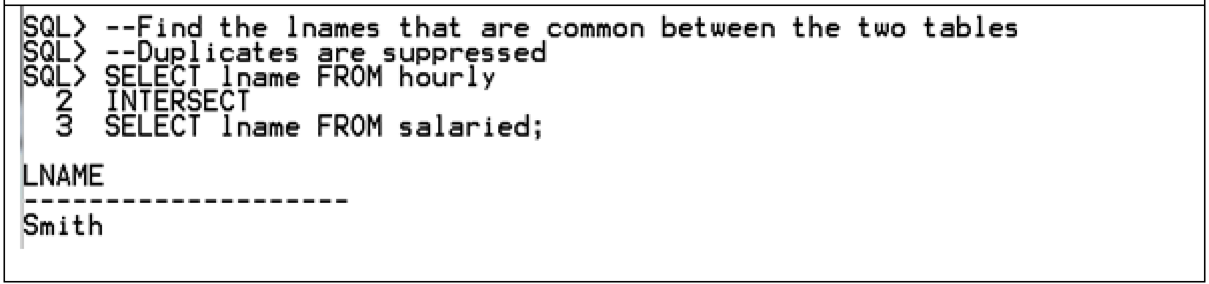
```
--Whereas UNION suppresses duplicates, UNIONALL does not.
SELECT lname FROM hourly

UNION ALL

SELECT lname FROM salaried;
```



```
SQL> --Whereas UNION suppresses duplicates, UNIONALL does not
SQL> SELECT lname FROM hourly
  2  UNION ALL
  3  SELECT lname FROM salaried;

LNAME
--------------------
Smith
Wesseon
Smith
Wesseon
Smith
Jones

6 rows selected.
```

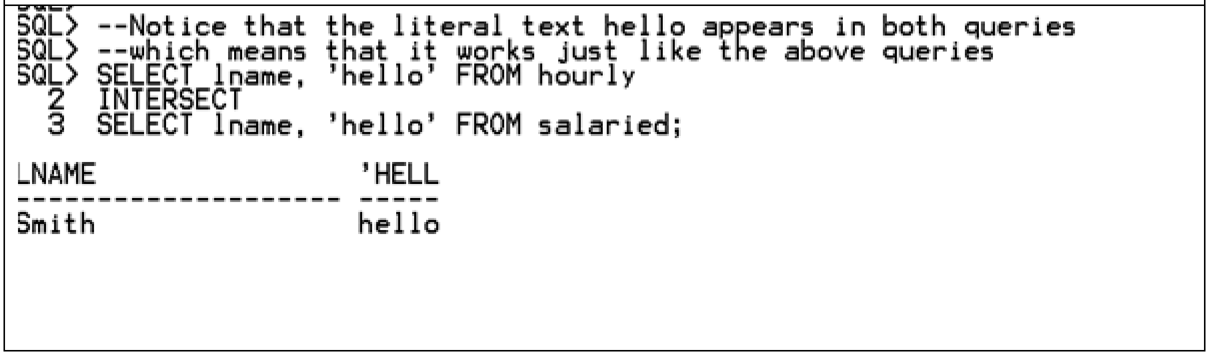
Example 10.5d (INTERSECT)

```
--Find the lnames that are common between the two tables.
--Duplicates are suppressed.
SELECT lname FROM hourly
INTERSECT
SELECT lname FROM salaried;
```



```
SQL> --Find the lnames that are common between the two tables
SQL> --Duplicates are suppressed
SQL> SELECT lname FROM hourly
  2  INTERSECT
  3  SELECT lname FROM salaried;

LNAME
--------------------
Smith
```

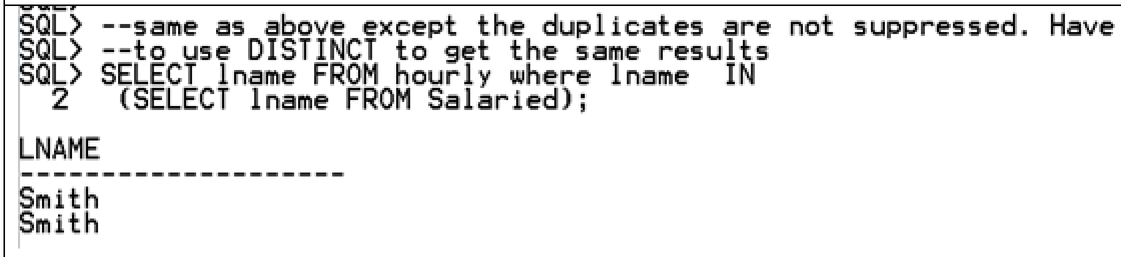
```
--Notice that the literal text hello appears in both queries, which means that it works just like the
--above queries.
SELECT lname, 'hello' FROM hourly
INTERSECT
SELECT lname, 'hello' FROM salaried;
```



```
SQL> --Notice that the literal text hello appears in both queries
SQL> --which means that it works just like the above queries
SQL> SELECT lname, 'hello' FROM hourly
  2  INTERSECT
  3  SELECT lname, 'hello' FROM salaried;

LNAME                'HELL
-------------------- -----
Smith                hello
```

--Same as above except the duplicates are not suppressed. Have to use DISTINCT to get the same --results.

```
SELECT lname FROM hourly where lname  IN
 (SELECT lname FROM Salaried);
```



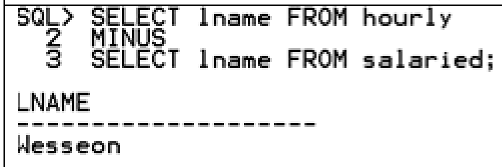
```
SQL> --same as above except the duplicates are not suppressed. Have
SQL> --to use DISTINCT to get the same results
SQL> SELECT lname FROM hourly where lname  IN
  2   (SELECT lname FROM Salaried);

LNAME
--------------------
Smith
Smith
```

Example 10.5e (MINUS)

--All the records that are in hourly which are not in salaried are displayed.

--Must have the same number of columns and type.

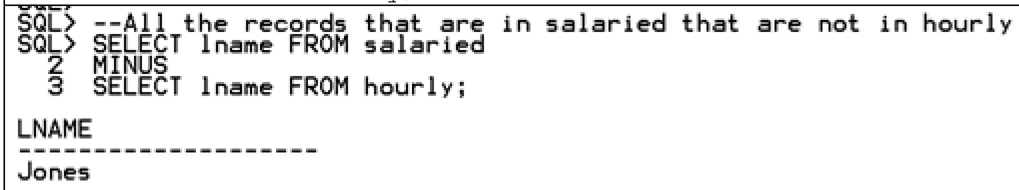
```
SELECT lname FROM hourly
MINUS
SELECT lname FROM salaried;
```



```
SQL> SELECT lname FROM hourly
  2  MINUS
  3  SELECT lname FROM salaried;

LNAME
--------------------
Wesseon
```

--All the records that are in salaried that are not in hourly are displayed.

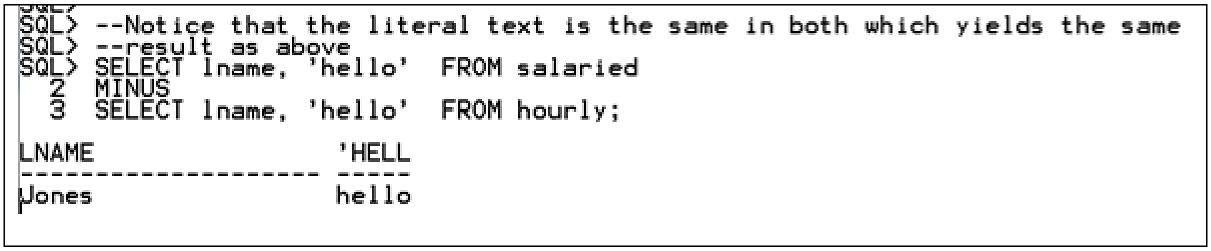
```
SELECT lname FROM salaried
MINUS
SELECT lname FROM hourly;
```



```
SQL> --All the records that are in salaried that are not in hourly
SQL> SELECT lname FROM salaried
  2  MINUS
  3  SELECT lname FROM hourly;

LNAME
--------------------
Jones
```

--Notice that the literal text is the same in both which yields the same result as above.

```
SELECT lname, 'hello'  FROM salaried
MINUS
SELECT lname, 'hello'  FROM hourly;
```



```
SQL> --Notice that the literal text is the same in both which yields the same
SQL> --result as above
SQL> SELECT lname, 'hello'  FROM salaried
  2  MINUS
  3  SELECT lname, 'hello'  FROM hourly;

LNAME                'HELL
-------------------- -----
Jones                hello
```

Example 10.5f (EXISTS   Uncorrelated)

The EXISTS function searches for the presence of a single row meeting the stated criteria as opposed to the IN statement which looks for all occurrences.

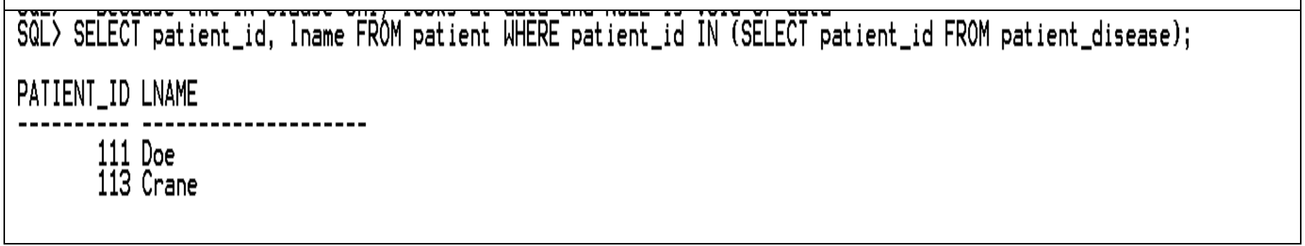
Rule of thumb:

- If the majority of the filtering criteria are in the subquery then the IN variation may be more **efficient**.
- If the majority of the filtering criteria are in the top query then the EXISTS variation may be more **efficient.**

.

EXISTS is usually more efficient that IN Because EXISTS use indexes of the table and hence scans the table faster as well as it returns the boolean value (T or F) If T is received for EXISTS clause than the rows will be returned otherwise not. Whereas IN works as simple query where it will scan all possible values in the table and then compares the condition given by you and then the result.

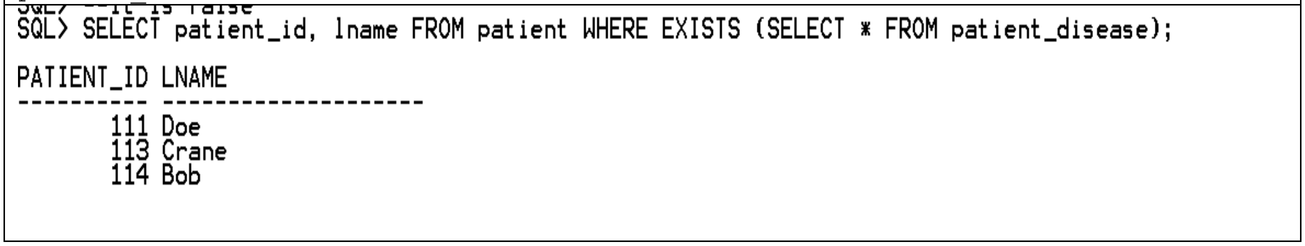
```
--The following displays all the patients who have diseases.
--The inner query is done first. The results are then passed on to the outer query.
--Notice that the patient_id in the outer query connects to the patient_id in the inner query.
--Note that patient_ids that are NULL in the outer query will not be considered because the IN clause only
--looks at data. NULL is void of data.
SELECT patient_id, lname FROM patient WHERE patient_id IN (SELECT patient_id FROM patient_disease);
```



```
SQL> SELECT patient_id, lname FROM patient WHERE patient_id IN (SELECT patient_id FROM patient_disease);

PATIENT_ID LNAME
---------- --------------------
       111 Doe
       113 Crane
```

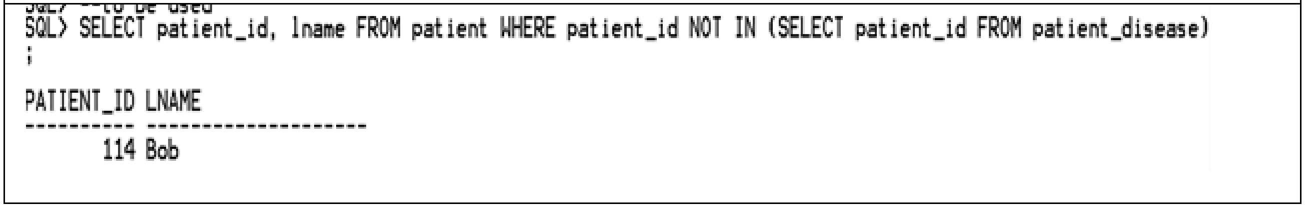
```
--All the patients, even the ones that don't have a disease are displayed.
--For every row that is being processed in the outer query, the inner query is executed. Notice
--that unlike the IN clause, there is nothing to connect the outer to the inner query. There is no
--column in the where clause. If EXISTS (SELECT * FROM patient_disease) which appears after
--the WHERE clause comes back with a TRUE result, then the row that is being processed by
--the outer query is accepted, otherwise it is rejected. When we get rows back from
```

```
--(SELECT * FROM patient_disease) then we have a true condition. If we get no rows back then it is false.
SELECT patient_id, lname FROM patient WHERE EXISTS (SELECT * FROM
patient_disease);
```



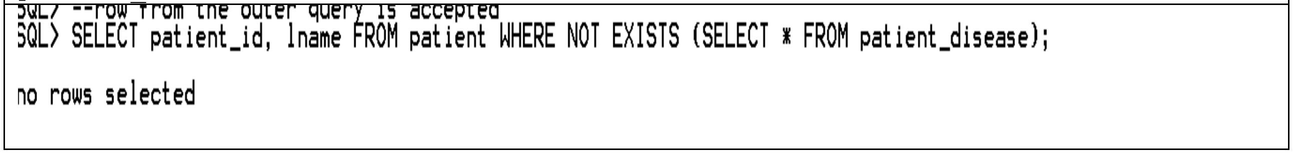
```
SQL> --it is false
SQL> SELECT patient_id, lname FROM patient WHERE EXISTS (SELECT * FROM patient_disease);

PATIENT_ID LNAME
---------- --------------------
       111 Doe
       113 Crane
       114 Bob
```

```
--All the patients that don't have a disease are displayed.
--The NOT IN will not come back with any results if the inner query has any nulls in it.
--If there is a NULL, then it cannot check.
--The NOT IN will only check against data. NULL is void of data. The IS operator will have to be used instead.
SELECT patient_id, lname FROM patient WHERE patient_id NOT IN (SELECT
patient_id FROM patient_disease);
```



```
SQL> --to be used
SQL> SELECT patient_id, lname FROM patient WHERE patient_id NOT IN (SELECT patient_id FROM patient_disease)
;

PATIENT_ID LNAME
---------- --------------------
       114 Bob
```

```
/*The query below does not display any of the patients because there is always something in the
patient_disease table. For every row that is being processed in the outer query, the inner query is executed.
Notice that unlike the IN clause, there is nothing to connect the outer to the inner query. There is no column
in the where clause. If NOT EXISTS (SELECT * FROM patient_disease) which appears after the WHERE clause
comes back with a TRUE result then the row that is being processed by the outer query is accepted, otherwise
it is rejected. When we get rows back from (SELECT * FROM patient_disease), we have a true condition. we
negate it using the NOT operator  and it becomes false. Therefore the row from the outer query is rejected.

SELECT patient_id, lname FROM patient WHERE NOT EXISTS (SELECT * FROM
patient_disease);
```

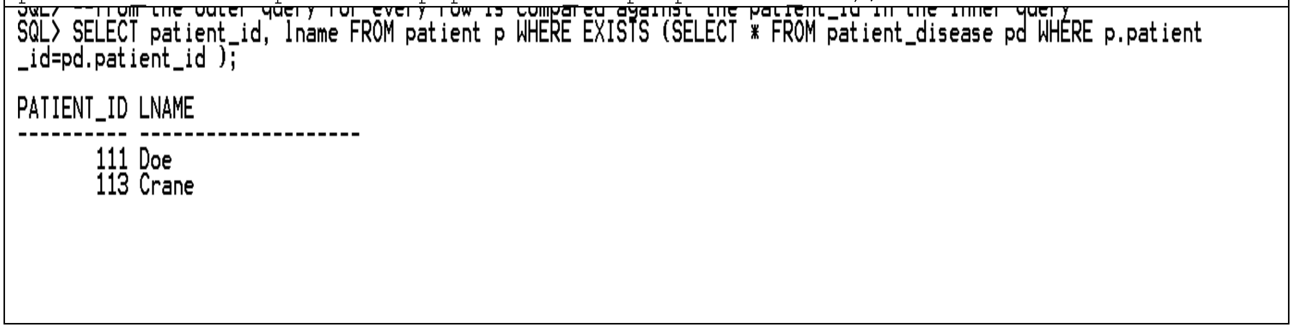


```
SQL> --row from the outer query is accepted
SQL> SELECT patient_id, lname FROM patient WHERE NOT EXISTS (SELECT * FROM patient_disease);

no rows selected
```

Example 10.5g (EXISTS   correlated)

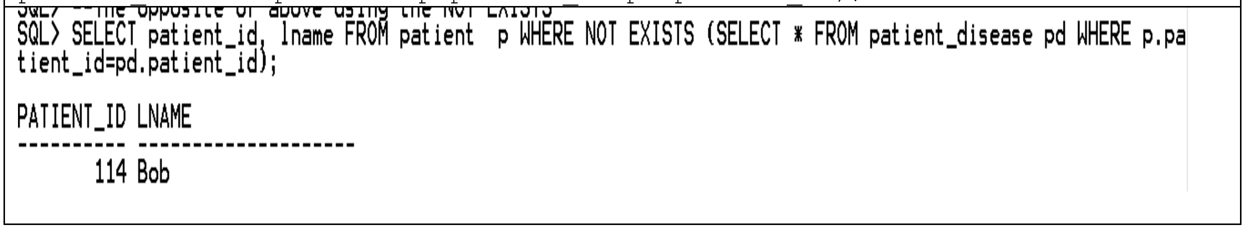
So far you have studied mostly uncorrelated subqueries: The subquery is executed first, its results are passed to the outer query, and then the outer query is executed. In a correlated subquery, Oracle uses a different procedure to execute a query. A correlated subquery references one or more columns in the outer query, and the EXISTS operator is used to test whether the relationship or link is present.

```
--All the patients who have a disease are displayed.
--Unlike the previous example, this one connects the outer query to the inner query. For every
--row that is processed in the outer query, the inner query also gets processed.  For every row in the outer
--query, the patient-id is compared against the patient_id in the inner query.
SELECT patient_id, lname FROM patient p WHERE EXISTS (SELECT * FROM
patient_disease pd WHERE p.patient_id=pd.patient_id );
```



```
SQL> --from the outer query for every row is compared against the patient_id in the inner query
SQL> SELECT patient_id, lname FROM patient p WHERE EXISTS (SELECT * FROM patient_disease pd WHERE p.patient
_id=pd.patient_id );

PATIENT_ID LNAME
---------- --------------------
       111 Doe
       113 Crane
```

```
--All the patients that don't have a disease are displayed.
--The opposite of above is done using the NOT EXISTS clause.
SELECT patient_id, lname FROM patient  p WHERE NOT EXISTS (SELECT * FROM
patient_disease pd WHERE p.patient_id=pd.patient_id);
```



```
SQL> --The opposite of above using the NOT EXISTS
SQL> SELECT patient_id, lname FROM patient  p WHERE NOT EXISTS (SELECT * FROM patient_disease pd WHERE p.pa
tient_id=pd.patient_id);

PATIENT_ID LNAME
---------- --------------------
       114 Bob
```

✓ *CHECK 10C*

1. Display the fname and personality description of all with salaries greater than 10000. Union the results with all those who are making less than 5000.
2. Display the name of all those who don't have a personality. Display name and "No personality"
    a. Use not exists
    b. Use minus
    c. Use not in

"The only people with whom you should try to get even are those who have helped you"

Summary Examples

```
--Cartesian product (All combinations)
--No connection is made between the tables.
SELECT * FROM patient, patient_disease;
SELECT * FROM patient CROSS JOIN patient_disease;


--Inner join (Only commonalities)

--All tables are connected.
SELECT p.patient_id, disease_desc FROM patient p, patient_disease pd, disease d
WHERE p.patient_id=pd.patient_id AND pd.disease_id=d.disease_id;


--No alias is needed with natural join.
SELECT patient_id, disease_desc FROM patient NATURAL JOIN patient_disease
NATURAL JOIN disease;


--Connect on the common column.
SELECT patient_id, disease_id FROM patient JOIN patient_disease USING
(patient_id) ;


--Replace the keyword WHERE with ON.
SELECT p.patient_id, disease_id FROM patient p JOIN patient_disease pd ON
p.patient_id=pd.patient_id;


--Example:  For each name and description category, display the sum for the salary for only those
-- categories whose sum is greater than 10000.
SELECT lname, description, sum(salary)  FROM patient p, patient_disease pd,
disease d WHERE p.patient_id=pd.patient_id AND pd.disease_id=d.disease_id GROUP
BY lname, description HAVING sum(salary)>10000;


--Outer join (Commonalities plus orphan records from one side)

--(+) means that if you cannot find a match then create an implicit NULL row which means we don't care
--if we find a match or not. (+) is associated with the table that does not have the matching record.
--(+) should be associated with the table only once and cannot appear on both sides of (=).
SELECT p.patient_id, disease_desc FROM patient p, patient_disease pd, disease d
WHERE p.patient_id=pd.patient_id(+) AND pd.disease_id=d.disease_id(+);

--Given the syntax LEFT OUTER JOIN, patient is on the left hand side. This means include all
--records, even the ones that don't match up. Take that result set and do another left outer join, which
--means include non-matching records.
SELECT p.patient_id, disease_desc FROM patient p LEFT OUTER JOIN patient_disease
```

```
pd ON p.patient_id=pd.patient_id  LEFT OUTER JOIN  disease d ON
pd.disease_id=d.disease_id;
```

--Can use RIGHT OUTER JOIN to accomplish the same thing.

```
SELECT p.patient_id, disease_desc FROM patient_disease pd  RIGHT OUTER JOIN
patient p ON p.patient_id=pd.patient_id  LEFT OUTER JOIN  disease d ON
pd.disease_id=d.disease_id;
```

--Full outer join (Commonalities and orphan records from both sides)

--FULL OUTER JOIN includes the results that are non-matching from both the left and the right hand side.

```
SELECT p.patient_id, disease_desc FROM  patient_disease pd RIGHT OUTER JOIN
patient p ON p.patient_id=pd.patient_id  FULL OUTER JOIN disease d ON
pd.disease_id=d.disease_id ;
```

--Same as above

```
SELECT p.patient_id, disease_desc FROM patient p LEFT OUTER JOIN patient_disease
pd ON p.patient_id=pd.patient_id  FULL OUTER JOIN  disease d ON
pd.disease_id=d.disease_id;
```

--With unions, unionall, intersect, and minus, the number of columns and the type of columns must
--match. Also, the column headings come from the first query. Order by must be associated with the
--last query.
--UNION : Suppresses duplicates whereas UNIONALL does not suppress. They both appends the results of one
--result set to another.

```
SELECT patient_id FROM patient
UNION
SELECT patient_id FROM sick
```

--INTERSECT: Finds the common patient_ids between the two tables.

```
SELECT patient_id FROM patient
INTERSECT
SELECT patient_id FROM sick
```

--MINUS: The patient_ids that are in patient but not in sick table are displayed.

```
SELECT patient_id FROM patient
MINUS
SELECT patient_id FROM sick
```

--Notice the parantheses. We want to do the minus first. Then the result is going to be intersected.

```
SELECT patient_id FROM patient
INTERSECT
(SELECT patient_id FROM sick
MINUS
SELECT patient_id FROM another)


--IN/EXISTS

--The inner query is done first and the results are passed back to the outer query.
--Connection is made with the patient_id.
SELECT * FROM patient WHERE patient_id IN (SELECT patient_id FROM
patient_disease)


--For every row being processed in the outer query, the inner query is processed. Notice the
--(*) and also, there is no connecting column in the outer WHERE clause. If rows  come back from
--the inner query then it is true otherwise it is false. When false, the row from the outer query is rejected
--otherwise it is accepted.
SELECT * FROM patient  p WHERE EXISTS (SELECT * FROM patient_disease  pd WHERE
p.patient_id=pd.patient_id)
```