SUBTYPE INTEGER IS NUMBER(38,0);

DECLARE

c\_id customers.id%type := 1; c\_name customers.name%type; c\_addr customers.address%type;

c\_sal customers.salary%type;

BEGIN

SELECT name, address, salary INTO c\_name, c\_addr, c\_sal FROM customers WHERE id = c\_id; dbms\_output.put\_line ('Customer ' ||c\_name || ' from ' || c\_addr || ' earns ' || c\_sal); END;

pi constant number := 3.141592654;

1 ASCII(x); Returns the ASCII value of the character x.

2 CHR(x); Returns the character with the ASCII value of x.

3 CONCAT(x, y); Concatenates the strings x and y and return the appended string.

4 INITCAP(x); Converts the initial letter of each word in x to uppercase and returns that string.

INSTR(x, find\_string [, start] [, occurrence]); Searches for find\_string in x and returns the

5

position at which it occurs.

6 INSTRB(x); Returns the location of a string within another string, but returns the value in bytes.

7 LENGTH(x); Returns the number of characters in x.

8 LENGTHB(x); Returns the length of a character string in bytes for single byte character set.

9 LOWER(x); Converts the letters in x to lowercase and returns that string.

LPAD(x, width [, pad\_string]) ; Pads x with spaces to left, to bring the total length of the string

10

up to width characters.

11 LTRIM(x [, trim\_string]); Trims characters from the left of x.

NANVL(x, value); Returns value if x matches the NaN special value (not a number), otherwise x is

12

returned.

NLS\_INITCAP(x); Same as the INITCAP function except that it can use a different sort method as

13 specified by NLSSORT.

NLS\_LOWER(x) ; Same as the LOWER function except that it can use a different sort method as

14 specified by NLSSORT.

NLS\_UPPER(x); Same as the UPPER function except that it can use a different sort method as

15 specified by NLSSORT.

NLSSORT(x); Changes the method of sorting the characters. Must be specified before any NLS

16

function; otherwise, the default sort will be used.

17 NVL(x, value); Returns value if x is null; otherwise, x is returned.

18 NVL2(x, value1, value2); Returns value1 if x is not null; if x is null, value2 is returned.

REPLACE(x, search\_string, replace\_string); Searches x for search\_string and replaces it with

19

replace\_string.

20 RPAD(x, width [, pad\_string]); Pads x to the right.

21 RTRIM(x [, trim\_string]); Trims x from the right.

22 SOUNDEX(x) ; Returns a string containing the phonetic representation of x.

SUBSTR(x, start [, length]); Returns a substring of x that begins at the position specified by start. An

23 optional length for the substring may be supplied.

SUBSTRB(x); Same as SUBSTR except the parameters are expressed in bytes instead of characters for

24

the single-byte character systems

25 TRIM([trim\_char FROM) x); Trims characters from the left and right of x.

26 UPPER(x); Converts the letters in x to uppercase and returns that string.

DECLARE

type namesarray IS VARRAY(5) OF VARCHAR2(10); type grades IS VARRAY(5) OF INTEGER; names namesarray; marks grades; total integer;

BEGIN

names := namesarray('Kavita', 'Pritam', 'Ayan', 'Rishav', 'Aziz'); marks:= grades(98, 97, 78, 87, 92); total := names.count;

dbms\_output.put\_line('Total '|| total || ' Students');

FOR i in 1 .. total LOOP dbms\_output.put\_line('Student: ' || names(i) || ' Marks: ' || marks(i)); END LOOP;

DECLARE

a number; b number; c number;

PROCEDURE findMin(x IN number, y IN number, z OUT number) IS

BEGIN

IF x < y THEN z:= x;

ELSE z:= y;

END IF;

END;

BEGIN

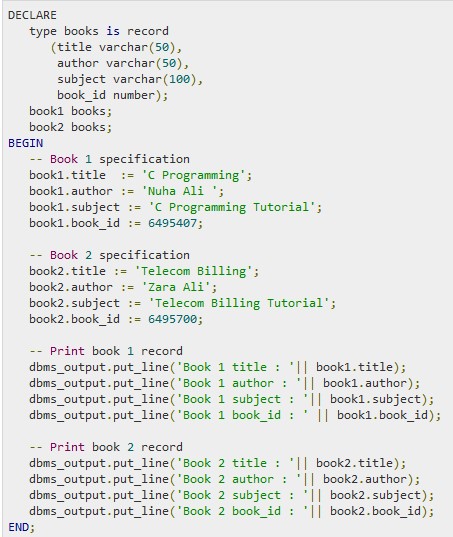
a:= 23; b:= 45; findMin(a, b, c);

dbms\_output.put\_line(' Minimum of (23, 45) : ' || c); END;

CREATE OR REPLACE FUNCTION totalCustomers RETURN number IS total number(2) := 0;

BEGIN

SELECT count(\*) into total FROM customers; RETURN total; END;

RECORDS:

Exceptions:

DECLARE

c\_id customers.id%type := 8;

c\_name customers.name%type;

c\_addr customers.address%type;

BEGIN

SELECT name, address INTO c\_name, c\_addr

FROM customers

WHERE id = c\_id;

DBMS\_OUTPUT.PUT\_LINE ('Name: '|| c\_name);

DBMS\_OUTPUT.PUT\_LINE ('Address: ' || c\_addr);

EXCEPTION

WHEN no\_data\_found THEN

dbms\_output.put\_line('No such customer!');

WHEN others THEN

dbms\_output.put\_line('Error!');

END;

/

Trigger:

CREATE OR REPLACE TRIGGER display\_salary\_changes

BEFORE DELETE OR INSERT OR UPDATE ON emp1170

FOR EACH ROW

WHEN (NEW.ID > 0)

DECLARE

sal\_diff number;

BEGIN

sal\_diff := :NEW.salary - :OLD.salary;

dbms\_output.put\_line('Old salary: ' || :OLD.salary);

dbms\_output.put\_line('New salary: ' || :NEW.salary);

dbms\_output.put\_line('Salary difference: ' || sal\_diff);

END;

/

goto:

DECLARE

a number(2) := 10;

BEGIN

<<loopstart>>

-- while loop execution

WHILE a < 20 LOOP

dbms\_output.put\_line ('value of a: ' || a);

a := a + 1;

IF a = 15 THEN

a := a + 1;

GOTO loopstart;

END IF;

END LOOP;

END;

/

Continue:

DECLARE

a number(2) := 10;

BEGIN

-- while loop execution

WHILE a < 20 LOOP

dbms\_output.put\_line ('value of a: ' || a);

a := a + 1;

IF a = 15 THEN

-- skip the loop using the CONTINUE statement

a := a + 1;

CONTINUE;

END IF;

END LOOP;

END;

/

**DATE AND TIME:**

PL/SQL provides two classes of date and time related data types:

Datetime data types

Interval data types

The Datetime data types are:

DATE

TIMESTAMP

TIMESTAMP WITH TIME ZONE

TIMESTAMP WITH LOCAL TIME ZONE

The Interval data types are:

INTERVAL YEAR TO MONTH

INTERVAL DAY TO SECOND

Field Values for Datetime and Interval Data Types

Both datetime and interval data types consist of fields. The values of these fields determine the value of the datatype. The following table lists the fields and their possible values for datetimes and intervals.

Field Name Valid Datetime Values Valid Interval Values

YEAR -4712 to 9999 (excluding year 0) Any nonzero integer

MONTH 01 to 12 0 to 11

DAY 01 to 31 (limited by the values of MONTH and YEAR, according to the rules of the calendar for the locale) Any nonzero integer

HOUR 00 to 23 0 to 23

MINUTE 00 to 59 0 to 59

SECOND 00 to 59.9(n), where 9(n) is the precision of time fractional seconds

The 9(n) portion is not applicable for DATE.

0 to 59.9(n), where 9(n) is the precision of interval fractional seconds

TIMEZONE\_HOUR -12 to 14 (range accommodates daylight savings time changes)

Not applicable for DATE or TIMESTAMP.

Not applicable

TIMEZONE\_MINUTE 00 to 59

Not applicable for DATE or TIMESTAMP.

Not applicable

TIMEZONE\_REGION Not applicable for DATE or TIMESTAMP. Not applicable

TIMEZONE\_ABBR Not applicable for DATE or TIMESTAMP. Not applicable

The Datetime Data Types and Functions

Following are the Datetime data types:

DATE - it stores date and time information in both character and number datatypes. It is made of information on century, year, month, date, hour, minute, and second. It is specified as:

TIMESTAMP - it is an extension of the DATE datatype. It stores the year, month, and day of the DATE datatype, along with hour, minute, and second values. It is useful for storing precise time values.

TIMESTAMP WITH TIME ZONE - it is a variant of TIMESTAMP that includes a time zone region name or a time zone offset in its value. The time zone offset is the difference (in hours and minutes) between local time and UTC. This datatype is useful for collecting and evaluating date information across geographic regions.

TIMESTAMP WITH LOCAL TIME ZONE - it is another variant of TIMESTAMP that includes a time zone offset in its value.

Following table provides the Datetime functions (where, x has datetime value):

S.N Function Name & Description

1 ADD\_MONTHS(x, y);

Adds y months to x.

2 LAST\_DAY(x);

Returns the last day of the month.

3 MONTHS\_BETWEEN(x, y);

Returns the number of months between x and y.

4 NEXT\_DAY(x, day);

Returns the datetime of the next day after x.

5 NEW\_TIME;

Returns the time/day value from a time zone specified by the user.

6 ROUND(x [, unit]);

Rounds x;

7 SYSDATE();

Returns the current datetime.

8 TRUNC(x [, unit]);

Truncates x.

Timestamp functions (where, x has a timestamp value):

S.N Function Name & Description

1 CURRENT\_TIMESTAMP();

Returns a TIMESTAMP WITH TIME ZONE containing the current session time along with the session time zone.

2 EXTRACT({ YEAR | MONTH | DAY | HOUR | MINUTE | SECOND } | { TIMEZONE\_HOUR | TIMEZONE\_MINUTE } | { TIMEZONE\_REGION | } TIMEZONE\_ABBR ) FROM x)

Extracts and returns a year, month, day, hour, minute, second, or time zone from x;

3 FROM\_TZ(x, time\_zone);

Converts the TIMESTAMP x and time zone specified by time\_zone to a TIMESTAMP WITH TIMEZONE.

4 LOCALTIMESTAMP();

Returns a TIMESTAMP containing the local time in the session time zone.

5 SYSTIMESTAMP();

Returns a TIMESTAMP WITH TIME ZONE containing the current database time along with the database time zone.

6 SYS\_EXTRACT\_UTC(x);

Converts the TIMESTAMP WITH TIMEZONE x to a TIMESTAMP containing the date and time in UTC.

7 TO\_TIMESTAMP(x, [format]);

Converts the string x to a TIMESTAMP.

8 TO\_TIMESTAMP\_TZ(x, [format]);

Converts the string x to a TIMESTAMP WITH TIMEZONE.

Examples:

The following code snippets illustrate the use of the above functions:

SELECT SYSDATE FROM DUAL;

Output:

08/31/2012 5:25:34 PM

SELECT TO\_CHAR(CURRENT\_DATE, 'DD-MM-YYYY HH:MI:SS') FROM DUAL;

Output:

31-08-2012 05:26:14

SELECT ADD\_MONTHS(SYSDATE, 5) FROM DUAL;

Output:

01/31/2013 5:26:31 PM

SELECT LOCALTIMESTAMP FROM DUAL;

Output:

8/31/2012 5:26:55.347000 PM

The Interval Data Types and Functions

Following are the Interval data types:

INTERVAL YEAR TO MONTH - it stores a period of time using the YEAR and MONTH datetime fields.

INTERVAL DAY TO SECOND - it stores a period of time in terms of days, hours, minutes, and seconds.

Interval functions:

S.N Function Name & Description

1 NUMTODSINTERVAL(x, interval\_unit);

Converts the number x to an INTERVAL DAY TO SECOND.

2 NUMTOYMINTERVAL(x, interval\_unit);

Converts the number x to an INTERVAL YEAR TO MONTH.

3 TO\_DSINTERVAL(x);

Converts the string x to an INTERVAL DAY TO SECOND.

4 TO\_YMINTERVAL(x);

Converts the string x to an INTERVAL YEAR T**O** MONTH.

**CURSOR:**

DECLARE

c\_id customers.id%type;

c\_name customers.name%type;

c\_addr customers.address%type;

CURSOR c\_customers is

SELECT id, name, address FROM customers;

BEGIN

OPEN c\_customers;

LOOP

FETCH c\_customers into c\_id, c\_name, c\_addr;

EXIT WHEN c\_customers%notfound;

dbms\_output.put\_line(c\_id || ' ' || c\_name || ' ' || c\_addr);

END LOOP;

CLOSE c\_customers;

END;

/

DECLARE

total\_rows number(2);

BEGIN

UPDATE customers

SET salary = salary + 500;

IF sql%notfound THEN

dbms\_output.put\_line('no customers selected');

ELSIF sql%found THEN

total\_rows := sql%rowcount;

dbms\_output.put\_line( total\_rows || ' customers selected ');

END IF;

END;

/

DATE AGHAIN:

SQL> DECLARE

2 l\_date DATE;

3 BEGIN

4 l\_date := TO\_DATE (&l\_date);

5 dbms\_output.put\_line(l\_date);

6 end;

7 /

Enter value for l\_date: '12-JAN-2011'

old 4: l\_date := TO\_DATE (&l\_date);

new 4: l\_date := TO\_DATE ('12-JAN-2011');

12-JAN-11

PL/SQL procedure successfully completed.

SQLPLUS:  
ARITHMETHIC:  
SELECT ABS(-15) "Absolute" FROM DUAL;

**SELECT ROUND(SALARY) FROM EMP;**

**SELECT CEIL(SALARY) FROM EMP;(NOT:ceil always goes to higher number)**

select max(hours) from emp;

select count(salary) from emp;

select avg(salary) from emp;

select sum(salary) from emp;

**group by:**

**:select sum(salary) from emp group by category;**

SQL> select category, sum(salary) from emp group by category;

CATEGORY SUM(SALARY)

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

manager 17000.75

supervisor 30000

account manager 46501.32

ALTER TABLE table\_name ADD (column\_name datatype[,...]);

ALTER TABLE table\_name MODIFY (column\_name datatype[,...]);

ALTER TABLE table\_name DROP (column\_name datatype[,...]);

ALTER TABLE table\_name RENAME COLUMN old\_name to new\_name;

ALTER TABLE table\_name RENAME to new\_name;

* CREATE TABLE *NewTableName* AS SELECT \* FROM *ExistingTable*;
* CREATE TABLE Employees (

EmployeeNumber varchar2(6),

FirstName nvarchar2(20) DEFAULT 'John',

LastName varchar2(20) DEFAULT 'Doe', HourlySalary number(6, 2) DEFAULT 12.50 7 );

CONSTRAINTS:

**CREATE TABLE table\_name**

**( column1 datatype null/not null,**

**column2 datatype null/not null, ...**

**CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, ... column\_n) );**

* **ALTER TABLE table\_name ADD CONSTRAINT constraint\_name PRIMARY KEY (column1, column2, ... column\_n);**
* **ALTER TABLE table\_name DROP CONSTRAINT constraint\_name;**
* **ALTER TABLE table\_name DISABLE CONSTRAINT constraint\_name;**
* **Alter student disable constraint reg\_pk;**
* **ALTER TABLE table\_name ENABLE CONSTRAINT constraint\_name;  
  Alter student enable constraint reg\_pk;**

**CHECK CONSTRAINT:  
CONSTRAINT empno\_ck CHECK(employee\_number>100));**

**CONSTRAINT car\_uk UNIQUE(car\_no));**

**SELECT \* FROM Persons**

**WHERE City='Sandnes'**

**LIKE:**

**SELECT \* FROM Persons**

**WHERE FirstName LIKE '%a'**

**UPDATE table\_name**

**SET column\_name = new\_value**

**WHERE column\_name = some\_value**

**DELETE FROM table\_name**

**WHERE column\_name = some\_value**

**Create table table\_name (columnname datatype constraint constraint\_name references table\_name (columnname));**

* **SOL>create table adm(stuid number(6) constraint stuid\_pk primary key,sname varchar2(15),per number(5));**
* **SQL> create table course(stuid number(6) constraint sid\_fk references adm(stuid),branch varchar2(5),sec varchar2(10));**
* **Defer specific constraints during session:**

**SET CONSTRAINTS DOG\_FK, SHOW\_NAME\_FK DEFERRED;**

* **Defer all deferrable constraints during session:**

**SET CONSTRAINTS ALL DEFERRED;**

* **Reset all deferrable constraints during session:**

**SET CONSTRAINTS ALL IMMEDIATE;**

**select distinct *customer\_name* from *borrower, loan* where *borrower loan\_number = loan.loan\_number* and  
 *branch\_name =* 'Perryridge'   
 order by *customer\_name***

* **Find all customers who have a loan, an account, or both:**
* **(select *customer\_name* from *depositor*)  
  union  
  (select *customer\_name* from *borrower)***
* **Find all customers who have both a loan and an account.**
* **(select *customer\_name* from *depositor*)  
  intersect  
  (select *customer\_name* from *borrower)***

**Find all customers who have an account but no loan.**

**select *customer\_name* from *depositor*)  
except  
(select *customer\_name* from *borrower)***

**select count (distinct *customer\_name)* from *depositor***

**select *branch\_name,* avg (*balance*)from *account* group by *branch\_name* having avg(*balance*) *>* 1200**

**Note: predicates in the having clause are applied after the   
 formation of groups whereas predicates in the where   
 clause are applied before forming groups**

**IN CONSTRUCT**

* **Find all customers who have both an account and a loan at the bank.**
* **select distinct *customer\_name* from *borrower* where *customer\_name* in (select *customer\_name* from *depositor* )**
* **Find all customers who have a loan at the bank but do not have   
   an account at the bank**
* **select distinct *customer\_name* from *borrower* where *customer\_name* not in (select *customer\_name* from *depositor* )**
* **“Some” Construct:**
* **select *branch\_name* from *branch* where *assets >* some  
   (select *assets* from *branch* where *branch\_city =* 'Brooklyn')**

**ALL construct**

**select *branch\_name* from *branch* where *assets >* all  
 (select *assets* from *branch* where *branch\_city =* 'Brooklyn')**

* **Find all customers who have an account at all branches located in Brooklyn.**
* **select distinct *S.customer\_name* from *depositor* as *S* where not exists (  
   (select *branch\_name* from *branch* where *branch\_city =* 'Brooklyn')   
   except  
   (select *R.branch\_name* from *depositor* as *T, account* as *R* where *T.account\_number = R.account\_number* and  
   *S.customer\_name = T.customer\_name* ))**

**AGGREGATE FUNTIONS:**

**SQL RAND Function - This is used to generate a random number using**

**SQL command.**

**● SQL CONCAT Function - This is used to concatenate any string inside any**

**SQL command.**

**● SQL Numeric Functions - Complete list of SQL functions required to**

**manipulate numbers in SQL.**

**● SQL String Functions - Complete list of SQL functions required to**

**manipulate strings in SQL.**

**● SQL STDDEV Functions – Used to calculate the standard deviation of N (T)**

**● SQL Variance Functions – Used to find the variance of N(tuples)**

**SQL COUNT Function - The SQL COUNT aggregate function is used to**

**count the number of rows in a database table.**

**● SQL MAX Function - The SQL MAX aggregate function allows us to select**

**the highest (maximum) value for a certain column.**

**● SQL MIN Function - The SQL MIN aggregate function allows us to select**

**the lowest (minimum) value for a certain column.**

**● SQL AVG Function - The SQL AVG aggregate function selects the average**

**value for certain table column.**

**● SQL SUM Function - The SQL SUM aggregate function allows selecting the**

**total for a numeric column.**

**● SQL SQRT Functions - This is used to generate a square root of a given**

**number.**

**BETWEEN:**

**SELECT \* FROM Products  
 WHERE Price BETWEEN 10 AND 20;**

**SELECT \* FROM Orders  
 WHERE OrderDate**

**BETWEEN #07/04/2014# AND #07/09/2014#;**

**SQL CREATE INDEX Statement:**

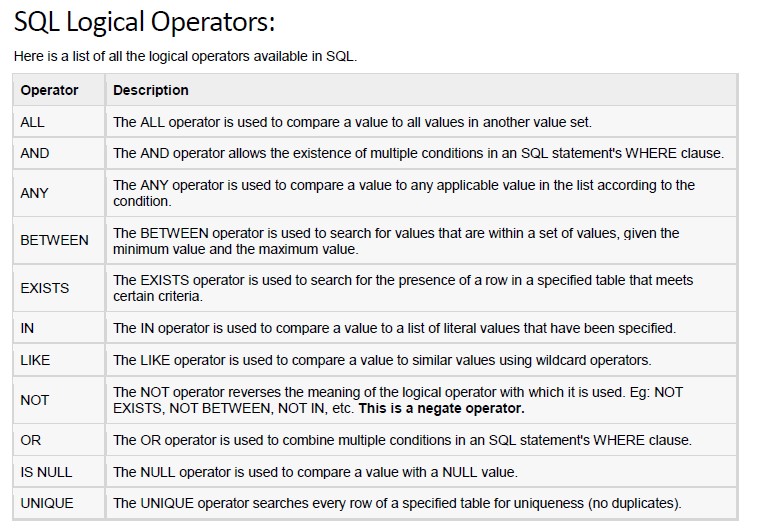
**CREATE UNIQUE INDEX index\_name**

**ON table\_name ( column1, column2,...columnN);**

**SQL DROP INDEX Statement:**

**ALTER TABLE table\_name**

**DROP INDEX index\_name;**



**JOINS:  
INNER JOIN:**

**SELECT *column\_name(s)*  
FROM *table1* INNER JOIN *table2*  
ON *table1.column\_name*=*table2.column\_name*;**

**Natural join is same as Inner join but only the difference is it will restrict to display redundant values**

CROSS JOIN:

select \*from a cross join b ;

FULL OUTER JOIN (has rows of both tables with null alongl=side the)

SELECT *column\_name(s)*  
FROM ***table1* FULL OUTER JOIN *table2***ON *table1.column\_name*=*table2.column\_name*;

RIGHT OUTER JOIN:

* SELECT gid, first\_name, last\_name, pid, gardener\_id, plant\_name FROM Gardners **RIGHT OUTER JOIN** Plantings ON gid = gardener\_id

LEFT OUTER JOIN:

* SELECT gid, first\_name, last\_name, pid, gardener\_id, plant\_name FROM Gardners **left OUTER JOIN** Plantings ON gid = gardener\_id

SELF JOIN:

* SELECT G1.gid, G1.first\_name, G1.last\_name, G2.gid, G2.first\_name, G2.last\_name FROM Gardners G1 INNER JOIN Gardners G2 ON G1.first\_name = G2.first\_name

MULTIPLE TABLES SELECT:

**SELECT e.ename, d.dname**

**FROM emp e, dept d;**

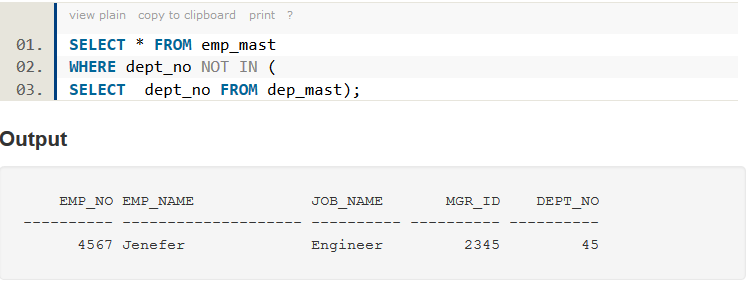
NATURAL JOIN (removes redundant values)

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

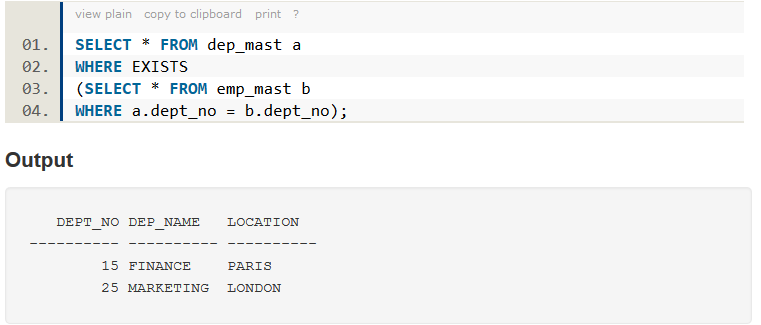
**NATURAL JOIN locations ;**

ANTI JOIN:

* An [antijoin](http://www.w3resource.com/oracle/joins/oracle-antijoins.php) between two tables returns rows from the first table where no matches are found in the second table. Anti-Joins are only available when performing a NOT IN sub-query



SEMI JOIN:



VIEWS:

* Create a view of all loan data in the *loan* relation, hiding the *amount* attribute

**create view** *loan\_branch* **as  
 select** *loan\_number, branch\_name* **from** *loan*

* Add a new tuple to *loan\_branch*

**insert into** *loan\_branch*  
 **values** ('L-37‘, 'Perryridge‘)

This insertion must be represented by the insertion of the tuple

('L-37', 'Perryridge', *null* )

into the *loan* relation