General Functions

The following functions work with any data type and pertain to using nulls:

- `NVL (expr1, expr2)`
- `NVL2 (expr1, expr2, expr3)`
- `NULLIF (expr1, expr2)`
- `COALESCE (expr1, expr2, ..., exprn)`

NVL Function

NVL (expr1, expr2)

Converts a null value to an actual value

Expr1: is the source value or expression that may contain null

Expr2: is the target value for converting the null

- Data types that can be used are date, character, and number.
- Data types must match:
  - `NVL(commission_pct,0)`
  - `NVL(hire_date,'01-JAN-97')`
  - `NVL(job_id,'No Job Yet')`

Using the NVL Function

List the employee names, annual salaries and their commission_pct, if they have commission_pct include annual commissions to annual salary too.

```
SELECT last name, salary, NVL(commission pct, 0),
   (salary*12) + (salary*12*NVL(commission_pct, 0)) AN_SAL
FROM employees;
```

1

2

	LAST_NAME	SALARY	NVL(COMMISSION_PCT,0)	AN_SAL
1	Whalen	4400	0	52800
2	Hartstein	13000	0	156000
3	Fay	6000	0	72000
4	Higgins	12000	0	144000
5	Gietz	8300	0	99600
6	King	24000	0	288000
7	Kochhar	17000	0	204000
8	De Haan	17000	0	204000
9	Hunold	9000	0	108000
10	Ernst	6000	0	72000

...

1

2

Using the NVL Function

Q: List all the departments and their managers, If the department do not have any manager list it like 'NO MANGER‘.

Using the NVL Function

List all the departments and their manager_id’s, If the department do not have any manager list it like 'NO MANGER‘.

```
select department_name , NVL(to_char(Manager_id), 'NO MANAGER')
from departments;
```

```
select department_name , Manager_id
from departments
Where manager_id is not NULL;
```

Using the `NVL2` Function

NVL2 (expr1, expr2, expr3)

Expr1: is the source value or expression that may contain a null

Expr2: is the value that is returned if expr1 is *not null*.

Expr3: is the value that is returned if expr1 is *null*.

Q: List the employees names and salaries only from department 80 and 50, if they have commission_pct "income" column will show that they have salary + Comm, otherwise this column will display the salary .

List the employees names and salaries only from department 80 and 50, if they have commission_pct "income" column will show that they have salary + Comm, otherwise this column display the salary .

```
SELECT last name,  salary, commission pct,
       NVL2(commission_pct,
            'SAL+COMM', 'SAL') income
FROM   employees WHERE department_id IN (50, 80);
```

1

2

	LAST_NAME	SALARY	COMMISSION_PCT	INCOME
1	Mourgos	5800	(null)	SAL
2	Rajs	3500	(null)	SAL
3	Davies	3100	(null)	SAL
4	Matos	2600	(null)	SAL
5	Vargas	2500	(null)	SAL
6	Zlotkey	10500	0.2	SAL+COMM
7	Abel	11000	0.3	SAL+COMM
8	Taylor	8600	0.2	SAL+COMM

1

2

Conditional Expressions

- Provide the use of the `IF-THEN-ELSE` logic within a SQL statement.
- Use two methods:
  - `CASE` expression
  - `DECODE` function

CASE Expression

Facilitates conditional inquiries by doing the work of an IF-THEN-ELSE statement:

```
CASE expr WHEN comparison_expr1 THEN return_expr1
          [WHEN comparison_expr2 THEN return_expr2
           WHEN comparison_exprn THEN return_exprn
           ELSE else_expr]
END
```

If expr is equal to the comparison_expr, returns the return_expr.

If none of them meet the condition, if there is an else stmt, it is returned.

Otherwise returns NULL.

=All the expressions data type should be same=

Using the CASE Expression

Facilitates conditional inquiries by doing the work of an `IF-THEN-ELSE` statement:

Increment the salaries according to the job_id's.

```
SELECT last_name, job_id, salary,
       CASE job_id WHEN 'IT_PROG'  THEN  1.10*salary
                   WHEN 'ST_CLERK' THEN  1.15*salary
                   WHEN 'SA_REP'   THEN  1.20*salary
       ELSE       salary END     "REVISED_SALARY"
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
1	Whalen	AD_ASST	4400	4400
...				
9	Hunold	IT_PROG	9000	9900
10	Ernst	IT_PROG	6000	6600
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
14	Davies	ST_CLERK	3100	3565
...				
19	Taylor	SA_REP	8600	10320
20	Grant	SA_REP	7000	8400

EXAMPLE

Q: Define the category of the salary for all employees:

salary <5000 : 'Low'
salary <10000 : 'Medium'
salary < 20000 : 'Good'
salary > 20000 : 'Excellent'

EXAMPLE

Define the category of the salary for all employees:
salary <5000 : 'Low'
salary <10000 : 'Medium'
salary < 20000 : 'Good'
salary > 20000 : 'Excellent'

```
SELECT last_name, salary,
     (CASE  WHEN salary <5000 THEN 'Low'
                WHEN salary <10000 THEN 'Medium'
                WHEN salary < 20000 THEN 'Good'
                ELSE 'Excellent'
      END) "qualified salary"
FROM employees;
```

DECODE Function

Facilitates conditional inquiries by doing the work of a CASE expression or an IF-THEN-ELSE statement:

```
DECODE(col|expression, search1, result1
                       [, search2, result2,...,]
                       [, default])
```

Using the DECODE Function

```
SELECT last_name, job_id, salary,
       DECODE(job_id, 'IT_PROG',  1.10*salary,
                      'ST_CLERK', 1.15*salary,
                      'SA_REP',   1.20*salary,
              salary)
       REVISED_SALARY
FROM   employees;
```

	LAST_NAME	JOB_ID	SALARY	REVISED_SALARY
...				
10	Ernst	IT_PROG	6000	6600
11	Lorentz	IT_PROG	4200	4620
12	Mourgos	ST_MAN	5800	5800
13	Rajs	ST_CLERK	3500	4025
...				
19	Taylor	SA_REP	8600	10320
20	Grant	SA_REP	7000	8400

Using the DECODE Function

Display the applicable tax rate for each employee:

MONTHLY RANGE	TAX RATE
0.0-1999.99	00%
2000.00-3999.99	09%
4000.00-5999.99	20%
6000.00-7999.99	30% …….40%, 42%, 44%, 45%

HINT: Use TRUNC(salary/2000, 0)

Using the DECODE Function

Display the applicable tax rate for each employee:

```
SELECT last_name, salary,
       DECODE (TRUNC(salary/2000, 0),
                        0, 0.00,
                        1, 0.09,
                        2, 0.20,
                        3, 0.30,
                        4, 0.40,
                        5, 0.42,
                        6, 0.44,
                           0.45) TAX_RATE
FROM   employees
WHERE  department_id = 80;
```

MONTHLY RANGE	TAX RATE
0.0-1999.99	00%
2000.00-3999.99	09%
4000.00-5999.99	20%
6000.00-7999.99	30% .....

Displaying Data from Multiple Tables Using Joins

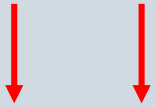
Obtaining Data from Multiple Tables

EMPLOYEES

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
...			
18	174	Abel	80
19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700



	EMPLOYEE_ID	DEPARTMENT_ID	DEPARTMENT_NAME
1	200	10	Administration
2	201	20	Marketing
3	202	20	Marketing
4	124	50	Shipping
...			
18	205	110	Accounting
19	206	110	Accounting

Types of Joins

Joins that are compliant with the SQL:1999 standard include the following:

- Natural join with the `NATURAL JOIN` clause
- Join with the `USING` Clause
- Join with the `ON` Clause
- `OUTER` joins:
  - `LEFT OUTER JOIN`
  - `RIGHT OUTER JOIN`
  - `FULL OUTER JOIN`
- Cross joins

Joining Tables Using SQL:1999 Syntax

Use a join to query data from more than one table:

```
SELECT   table1.column, table2.column
FROM     table1
[NATURAL JOIN table2] |
[JOIN table2 USING (column_name)] |
[JOIN table2
  ON (table1.column_name = table2.column_name)]|
[LEFT|RIGHT|FULL OUTER JOIN table2
  ON (table1.column_name = table2.column_name)]|
[CROSS JOIN table2];
```

Qualifying Ambiguous Column Names

- Use table prefixes to qualify column names that are in multiple tables.
- Use table prefixes to improve performance.
- Instead of full table name prefixes, use table aliases.
- Table alias gives a table a shorter name:
  - Keeps SQL code smaller, uses less memory
- Use column aliases to distinguish columns that have identical names, but reside in different tables.

Creating Natural Joins

- The `NATURAL JOIN` clause is based on all the columns in the two tables that have the same name.
- It selects rows from the two tables that have equal values in all matched columns.
- If the columns having the same names have different data types, an error is returned.

Retrieving Records with Natural Joins

List the departments and their locations.

```
SELECT department_id, department_name,
       location_id, city
FROM   departments
NATURAL JOIN locations ;
```

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID	CITY
1	60	IT	1400	Southlake
2	50	Shipping	1500	South San Francisco
3	10	Administration	1700	Seattle
4	90	Executive	1700	Seattle
5	110	Accounting	1700	Seattle
6	190	Contracting	1700	Seattle
7	20	Marketing	1800	Toronto
8	80	Sales	2500	Oxford

Retrieving Records with Natural Joins

List the department names and their manager names .

This join operation is not correct!! WHY?

```
SELECT department_name,first_name
FROM   departments
NATURAL JOIN employees ;
```

Creating Joins with the `USING` Clause

- If several columns have the same names but the data types do not match, use the `USING` clause to specify the columns for the equijoin.
- Use the `USING` clause to match only one column when more than one column matches.
- The `NATURAL JOIN` and `USING` clauses are mutually exclusive.

Joining Column Names

EMPLOYEES

	EMPLOYEE_ID	DEPARTMENT_ID
1	200	10
2	201	20
3	202	20
4	205	110
5	206	110
6	100	90
7	101	90
8	102	90
9	103	60
10	104	60

...

Foreign key

DEPARTMENTS

	DEPARTMENT_ID	DEPARTMENT_NAME
1	10	Administration
2	20	Marketing
3	50	Shipping
4	60	IT
5	80	Sales
6	90	Executive
7	110	Accounting
8	190	Contracting

Primary key

Retrieving Records with the `USING` Clause

List the employees, their departments and the location of these departments.

```
SELECT employee_id, last_name,
       location_id, department_id
FROM   employees JOIN departments
USING (department_id) ;
```

	EMPLOYEE_ID	LAST_NAME	LOCATION_ID	DEPARTMENT_ID
1	200	Whalen	1700	10
2	201	Hartstein	1800	20
3	202	Fay	1800	20
4	144	Vargas	1500	50
5	143	Matos	1500	50
6	142	Davies	1500	50
7	141	Rajs	1500	50
8	124	Mourgos	1500	50

...

18	206	Gietz	1700	110
19	205	Higgins	1700	110

Using Table Aliases with the `USING` Clause

- Do not qualify a column that is used in the `USING` clause.
- If the same column is used elsewhere in the SQL statement, do not alias it.

List the departments and their cities for only location 1400.

```
SELECT l.city, d.department_name
FROM   locations l JOIN departments d
USING (location_id)
WHERE d.location_id = 1400;
```

```
ORA-25154: column part of USING clause cannot have qualifier
25154. 00000 - "column part of USING clause cannot have qualifier"
*Cause:    Columns that are used for a named-join (either a NATURAL join
           or a join with a USING clause) cannot have an explicit qualifier.
*Action:   Remove the qualifier.
Error at Line: 4 Column: 6
```

Creating Joins with the ON Clause

- The join condition for the natural join is basically an equijoin of all columns with the same name.
- Use the ON clause to specify arbitrary conditions or specify columns to join.
- The join condition is separated from other search conditions.
- The ON clause makes code easy to understand.

Retrieving Records with the ON Clause

List the employees and their departments and locations.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id);
```

	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_ID_1	LOCATION_ID
1	200	Whalen	10	10	1700
2	201	Hartstein	20	20	1800
3	202	Fay	20	20	1800
4	144	Vargas	50	50	1500
5	143	Matos	50	50	1500
6	142	Davies	50	50	1500
7	141	Rajs	50	50	1500
8	124	Mourgos	50	50	1500
9	103	Hunold	60	60	1400
10	104	Ernst	60	60	1400
11	107	Lorentz	60	60	1400

...

List the managers of the departments.

```
select first_name, last_name, department_name
from departments d join employees
on(d.manager_id= employee_id);
```

```
select first_name, last_name, department_name
from departments d , employees
Where d.manager_id= employee_id;
```

Q: List the employees, their departments and their cities

Creating Three-Way Joins with the ON Clause

List the employees, their departments and their cities

```
SELECT employee_id, city, department_name
FROM   employees e
JOIN   departments d
ON     d.department_id = e.department_id
JOIN   locations l
ON     d.location_id = l.location_id;
```

	EMPLOYEE_ID	CITY	DEPARTMENT_NAME
1	100	Seattle	Executive
2	101	Seattle	Executive
3	102	Seattle	Executive
4	103	Southlake	IT
5	104	Southlake	IT
6	107	Southlake	IT
7	124	South San Francisco	Shipping
8	141	South San Francisco	Shipping
9	142	South San Francisco	Shipping

Q: List the employees and the department information for only the employees who has manager_id 149.

Applying Additional Conditions to a Join

Use the AND clause or the WHERE clause to apply additional conditions:

List the employees and the department information for only the employees who has manager_id 149.

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
AND    e.manager_id = 149 ;
```

Or

```
SELECT e.employee_id, e.last_name, e.department_id,
       d.department_id, d.location_id
FROM   employees e JOIN departments d
ON     (e.department_id = d.department_id)
WHERE   e.manager_id = 149 ;
```

Joining a Table to Itself (SELF JOIN)

EMPLOYEES (WORKER)

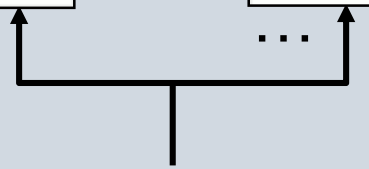
EMPLOYEE_ID	LAST_NAME	MANAGER_ID
200	Whalen	101
201	Hartstein	100
202	Fay	201
205	Higgins	101
206	Gietz	205
100	King	(null)
101	Kochhar	100
102	De Haan	100
103	Hunold	102
104	Ernst	103

...

EMPLOYEES (MANAGER)

EMPLOYEE_ID	LAST_NAME
200	Whalen
201	Hartstein
202	Fay
205	Higgins
206	Gietz
100	King
101	Kochhar
102	De Haan
103	Hunold
104	Ernst

...



MANAGER_ID in the WORKER table is equal to EMPLOYEE_ID in the MANAGER table.

Self-Joins Using the ON Clause

Q: List the worker names and their manager names.

Self-Joins Using the ON Clause

List the worker names and their manager names.

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker JOIN employees manager
ON    (worker.manager_id = manager.employee_id);
```

	EMP	MGR
1	Hunold	De Haan
2	Fay	Hartstein
3	Gietz	Higgins
4	Lorentz	Hunold
5	Ernst	Hunold
6	Zlotkey	King
7	Mourgos	King

...

Nonequijoins

List the employees and their job grade levels.

EMPLOYEES

	LAST_NAME	SALARY
1	Whalen	4400
2	Hartstein	13000
3	Fay	6000
4	Higgins	12000
5	Gietz	8300
6	King	24000
7	Kochhar	17000
8	De Haan	17000
9	Hunold	9000
10	Ernst	6000
...		
19	Taylor	8600
20	Grant	7000

JOB_GRADES

	GRADE_LEVEL	LOWEST_SAL	HIGHEST_SAL
1	A	1000	2999
2	B	3000	5999
3	C	6000	9999
4	D	10000	14999
5	E	15000	24999
6	F	25000	40000

The JOB_GRADES table defines the LOWEST_SAL and HIGHEST_SAL range of values for each GRADE_LEVEL. Therefore, the GRADE_LEVEL column can be used to assign grades to each employee.

Retrieving Records with Nonequijoins

List the employees and their job grade levels.

```
SELECT e.last_name, e.salary, j.grade_level
FROM   employees e JOIN job_grades j
ON     e.salary
       BETWEEN j.lowest_sal AND j.highest_sal;
```

	LAST_NAME	SALARY	GRADE_LEVEL
1	Vargas	2500	A
2	Matos	2600	A
3	Davies	3100	B
4	Rajs	3500	B
5	Lorentz	4200	B
6	Whalen	4400	B
7	Mourgos	5800	B
8	Ernst	6000	C
9	Fay	6000	C
10	Grant	7000	C

...

Returning Records with No Direct Match Using OUTER Joins

DEPARTMENTS

	DEPARTMENT_NAME	DEPARTMENT_ID
1	Administration	10
2	Marketing	20
3	Shipping	50
4	IT	60
5	Sales	80
6	Executive	90
7	Accounting	110
8	Contracting	190

There are no employees in department 190.

Equijoin with EMPLOYEES

	DEPARTMENT_ID	LAST_NAME
1	10	Whalen
2	20	Hartstein
3	20	Fay
4	110	Higgins
5	110	Gietz
6	90	King
7	90	Kochhar
8	90	De Haan
9	60	Hunold
10	60	Ernst
...		
18	80	Abel
19	80	Taylor

Employee “Grant” has not been assigned a department ID.

`INNER` Versus `OUTER` Joins

- In SQL:1999, the join of two tables returning only matched rows is called an `INNER` join.
- A join between two tables that returns the results of the `INNER` join as well as the unmatched rows from the left (or right) table is called a left (or right) `OUTER` join.
- A join between two tables that returns the results of an `INNER` join as well as the results of a left and right join is a full `OUTER` join.

LEFT OUTER JOIN

```
SELECT e.last_name, e.department_id, d.department_name
FROM   employees e LEFT OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Fay	20	Marketing
3	Hartstein	20	Marketing
4	Vargas	50	Shipping
5	Matos	50	Shipping

...

16	Kochhar	90	Executive
17	King	90	Executive
18	Gietz	110	Accounting
19	Higgins	110	Accounting
20	Grant	(null)	(null)

This query retrieves all the rows in the EMPLOYEES table , which is the left table, even if there is no match in the DEPARTMENTS table.

RIGHT OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e RIGHT OUTER JOIN departments d
ON     (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Davies	50	Shipping
5	Vargas	50	Shipping
6	Rajs	50	Shipping
7	Mourgos	50	Shipping
8	Matos	50	Shipping

...

18	Higgins	110	Accounting
19	Gietz	110	Accounting
20	(null)	190	Contracting

This query retrieves all the rows in the DEPARTMENTS table , which is the table at right, even if there is no match in the EMPLOYEES table.

FULL OUTER JOIN

```
SELECT e.last_name, d.department_id, d.department_name
FROM   employees e FULL OUTER JOIN departments d
ON    (e.department_id = d.department_id) ;
```

	LAST_NAME	DEPARTMENT_ID	DEPARTMENT_NAME
1	Whalen	10	Administration
2	Hartstein	20	Marketing
3	Fay	20	Marketing
4	Higgins	110	Accounting

...

17	Zlotkey	80	Sales
18	Abel	80	Sales
19	Taylor	80	Sales
20	Grant	(null)	(null)
21	(null)	190	Contracting

EXAMPLES:

Q1: List all the employees and their manager names, even they do not have any manager.

Q2: List all the managers and their workers even they do not have any worker.

EXAMPLES :

List all the employees and their manager names, even they do not have any manager.

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker left outer JOIN employees manager
ON    (worker.manager_id = manager.employee_id);
```

List all the managers and their workers even they do not have any worker.

```
SELECT worker.last_name emp, manager.last_name mgr
FROM   employees worker right outer JOIN employees manager
ON    (worker.manager_id = manager.employee_id)
order by mgr;
```

Cartesian Products

- A Cartesian product is formed when:
  - A join condition is omitted
  - A join condition is invalid
  - All rows in the first table are joined to all rows in the second table
- Always include a valid join condition if you want to avoid a Cartesian product.

Generating a Cartesian Product

EMPLOYEES (20 rows)

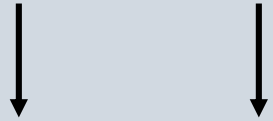
	EMPLOYEE_ID	LAST_NAME	DEPARTMENT_ID
1	200	Whalen	10
2	201	Hartstein	20
3	202	Fay	20
4	205	Higgins	110

...

19	176	Taylor	80
20	178	Grant	(null)

DEPARTMENTS (8 rows)

	DEPARTMENT_ID	DEPARTMENT_NAME	LOCATION_ID
1	10	Administration	1700
2	20	Marketing	1800
3	50	Shipping	1500
4	60	IT	1400
5	80	Sales	2500
6	90	Executive	1700
7	110	Accounting	1700
8	190	Contracting	1700



Cartesian product:
20 x 8 = 160 rows

	EMPLOYEE_ID	DEPARTMENT_ID	LOCATION_ID
1	200	10	1700
2	201	20	1700

...

21	200	10	1800
22	201	20	1800

...

159	176	80	1700
160	178	(null)	1700

Creating Cross Joins

- The `CROSS JOIN` clause produces the cross-product of two tables. (a join condition is omitted)
- This is also called a Cartesian product between the two tables.

List all the employees and their possible departments.

```
SELECT last_name, department_name
FROM   employees
CROSS JOIN departments ;
```

	LAST_NAME	DEPARTMENT_NAME
1	Abel	Administration
2	Davies	Administration
3	De Haan	Administration
4	Ernst	Administration
5	Fay	Administration

...

158	Vargas	Contracting
159	Whalen	Contracting
160	Zlotkey	Contracting

RELATIONAL ALGEBRA

Join Operation: JOINING TABLES –(*from stmt. and join conditions in where in SQL*)

Used to combine related tuples from two relations into single tuples.

(<relation1 name>) ∞ $_{<CONDITION>}$ (<relation 2 name>)

Join can also be defined as: $R \infty_{<CONDITION>} S \equiv \delta_{<CONDITION>}(R \times S)$

(USE THE COMPANY DATABASE)

- Retrieve the name of the manager of each department

DEP-MGR $\leftarrow$ DEPARTMENT $\infty_{MGRSSN=SSN}$ EMPLOYEE

RESULT $\leftarrow \pi_{DNAME,LNAME,FNAME}$ (DEP-MGR)

SQL SOLUTION:

```
SELECT DNAME, LNAME, FNAME
FROM DEPARTMENT, EMPLOYEE
WHERE MGRSSN = SSN;
```

another way of join operation

RELATIONAL ALGEBRA

(USE THE COMPANY DATABASE)

- Find the name and address of all employees who work for the "Research" department.

$$RESEARCH\_DEPT \leftarrow \delta_{DNAME='RESEARCH'}(DEPARTMENT)$$

$$RESEARCH\_EMPS \leftarrow RESEARCH\text{-}DEPT \infty_{DNUMBER=DNO} EMPLOYEE$$

$$RESULT \leftarrow \pi_{FNAME,LNAME,ADDRESS}(RESEARCH\_EMPS)$$

SQL SOLUTION:

```
SELECT LNAME, FNAME, ADDRESS
FROM DEPARTMENT, EMPLOYEE
WHERE DNO=DNUMBER AND DNAME ='RESEARCH';
```

SOLVE THE FOLLOWING QUERIES WITH SQL AND RELATIOANAL ALGEBRA BY USING COMPANY DATABASE:

1. Retrieve the birth date and address of the employee whose name is "ALICE SMITH".
2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth-date.
3. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.
4. Retrieve each female employee and a list of names of her dependents.
5. Retrieve all employees in department 1 whose salary is between $15000 and $20000.

SOLVE THE FOLLOWING QUERIES WITH SQL

6. Show the resulting salaries of every employee working on the 'Product X' project as they are given a 10% raise.
7. Select all employees whose address is Houston.
8. Retrieve all employees who were born during the 1950s.
9. Retrieve a list of employees and the projects each works in , ordered by the employees department and within each department ordered alphabetically by name.