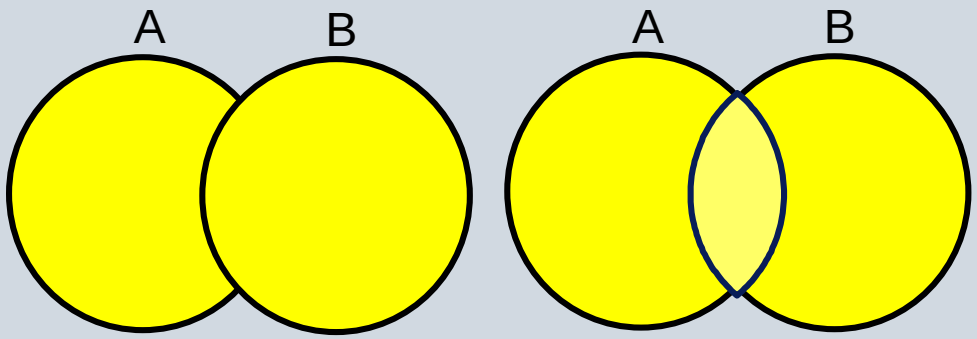
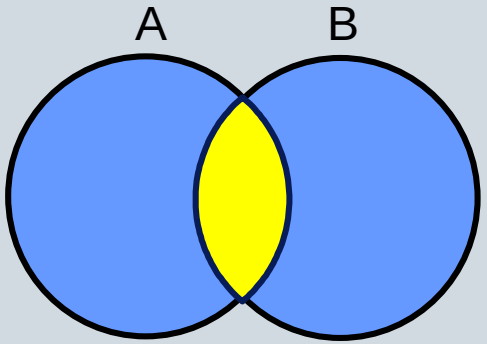
Set Operations With SQL

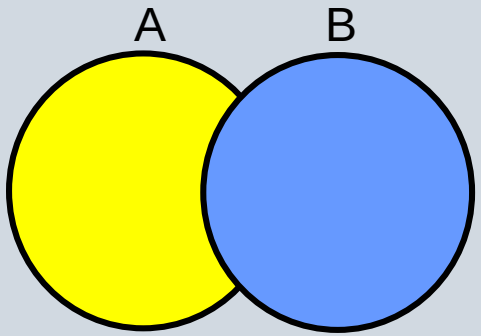
Set Operators



UNION/UNION ALL



INTERSECT



MINUS

Set Operator Guidelines

- The expressions in the `SELECT` lists must match in number.
- The data type of each column in the second query must match the data type of its corresponding column in the first query.
- Parentheses can be used to alter the sequence of execution.
- `ORDER BY` clause can appear only at the very end of the statement.

Oracle Server and Set Operators

- Duplicate rows are automatically eliminated except in `UNION ALL`.
- Column names from the first query appear in the result.
- The output is sorted in ascending order by default except in `UNION ALL`.

SET OPERATIONS			
OPERATION	**SQL FORMAT**	**OUTPUT**	
set 1 ∪ set2	set1 **UNION** set2	set	
set 1 ∩ set2	set1 **INTERSECTION** set2	set	
set 1 **-** set2	set1 **EXCEPT** set2	set	**MINUS in Oracle**
set1 ⊃ set 2	set1 **[NOT] CONTAINS** set2	T/F	**Not supported in Oracle**
member **ε** set	Attribute **[NOT] IN** set	T/F	
set is ∅	**[NOT] EXISTS** set	T/F	

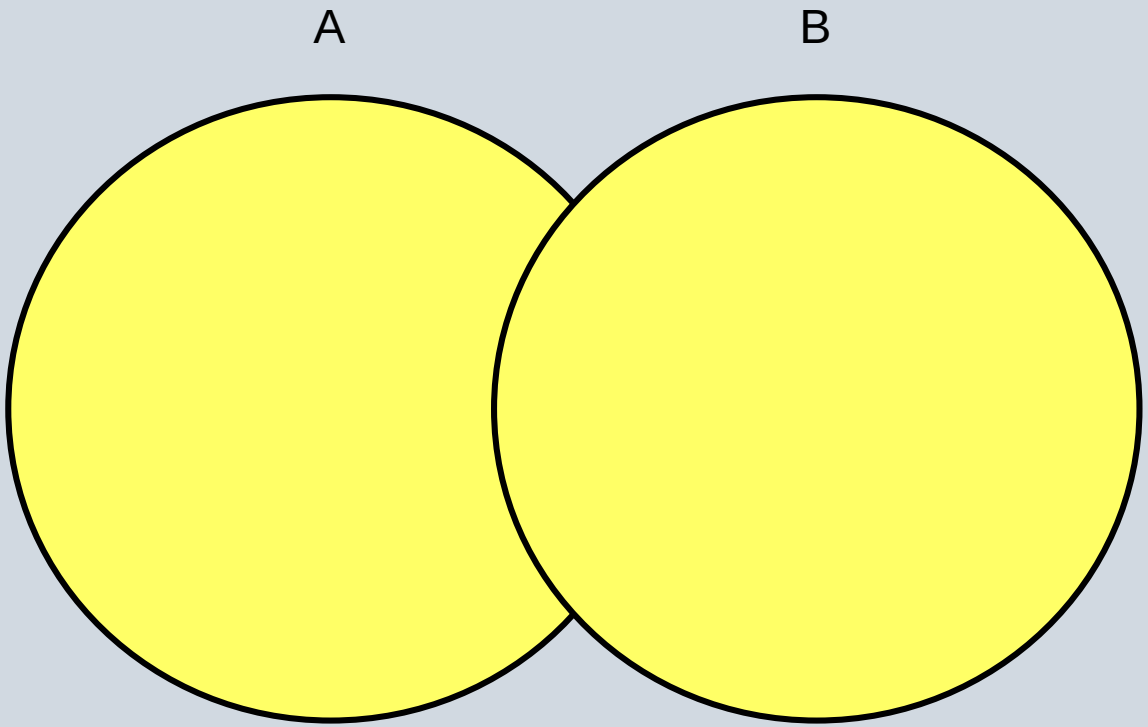
Using the UNION Operator

Display the current and previous job details of all employees. Display each employee only once.

```
SELECT employee_id, job_id
FROM   employees
UNION
SELECT employee_id, job_id
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID
1	100	AD_PRES
2	101	AC_ACCOUNT
...		
22	200	AC_ACCOUNT
23	200	AD_ASST
...		
27	205	AC_MGR
28	206	AC_ACCOUNT

`UNION ALL` Operator



The `UNION ALL` operator returns rows from both queries, including all duplications.

Using the UNION ALL Operator

The UNION ALL operator returns rows from both queries, including all duplications.

Display the current and previous departments of all employees.

```
SELECT employee_id, job_id, department_id
FROM   employees
UNION ALL
SELECT employee_id, job_id, department_id
FROM   job_history
ORDER BY  employee_id;
```

	EMPLOYEE_ID	JOB_ID	DEPARTMENT_ID
1	100	AD_PRES	90
...			
17	149	SA_MAN	80
18	174	SA_REP	80
19	176	SA_REP	80
20	176	SA_MAN	80
21	176	SA_REP	80
22	178	SA_REP	(null)
23	200	AD_ASST	10
...			
30	206	AC_ACCOUNT	110

Using the INTERSECT Operator

Display the employee IDs and job IDs of those employees who currently have a job title that is the same as their previous one (that is, they changed jobs but have now gone back to doing the same job they did previously).

```
SELECT employee_id, job_id
FROM   employees
INTERSECT
SELECT employee_id, job_id
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID
1	176	SA_REP
2	200	AD_ASST

Using the MINUS Operator

Display the employee IDs of those employees who have not changed their jobs even once.

```
SELECT employee_id
FROM   employees
MINUS
SELECT employee_id
FROM   job_history;
```

	EMPLOYEE_ID
1	100
2	103
3	104

...

13	202
14	205
15	206

Matching the SELECT Statements

Using the UNION operator, display the location ID, department name, and the state where it is located.

You must match the data type (using the TO_CHAR function or any other conversion functions) when columns do not exist in one or the other table.

```
SELECT location_id, department_name "Department",
   TO_CHAR(NULL) "Warehouse location"
FROM departments
UNION
SELECT location_id, TO_CHAR(NULL) "Department",
   state_province
FROM locations;
```

Matching the SELECT Statement: Example

Using the UNION operator, display the employee ID, job ID, and salary of all employees.

```
SELECT employee_id, job_id, salary
FROM   employees
UNION
SELECT employee_id, job_id,0
FROM   job_history;
```

	EMPLOYEE_ID	JOB_ID	SALARY
1	100	AD_PRES	24000
2	101	AC_ACCOUNT	0
3	101	AC_MGR	0
4	101	AD_VP	17000
5	102	AD_VP	17000

...

29	205	AC_MGR	12000
30	206	AC_ACCOUNT	8300

Using the `ORDER BY` Clause in Set Operations

- The `ORDER BY` clause can appear only once at the end of the compound query.
- Component queries cannot have individual `ORDER BY` clauses.
- The `ORDER BY` clause recognizes only the columns of the first `SELECT` query.
- By default, the first column of the first `SELECT` query is used to sort the output in an ascending order.

SET OPERATIONS IN RELATIONAL ALGEBRA

UNION : **R ∪ S:** Includes all tuples of R&S. Duplicate tuples eliminated.

INTERSECTION: **R ∩ S:** Includes all tuples which are in R & S at the same time. (common tuples)

DIFFERENCE: **R-S :** Include tuples who are in R but not in S.

The relations must be ***union compatible***.

(Use COMPANY DATABASE)

- Make a list of project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

$$SMITHS(ESSN) \leftarrow \pi_{SSN}(\delta_{LNAME='Smith'}(EMPLOYEE))$$

$$SMITH\text{-}WORK\text{-}PRJ \leftarrow \pi_{PNO}(WORKS\_ON \infty SMITHS)$$

$$MGRS \leftarrow \pi_{LNAME,DNUMBER}(EMPLOYEE \infty_{SSN=MGRSSN} DEPARTMENT)$$

$$SMITH\text{-}MGRS \leftarrow \delta_{LNAME='Smith'}(MGRS)$$

$$SMITH\text{-}MNGD\text{-}DEPTS(DNUM) \leftarrow \pi_{DNUMBER}(SMITH\text{-}MGRS)$$

$$SMITH\text{-}MGR\text{-}P(PNO) \leftarrow \pi_{PNUMBER}(SMITH\text{-}MNGD\text{-}DEPTS \infty PROJECT)$$

$$RESULT \leftarrow (SMITH\text{-}WORK\text{-}PRJ \cup SMITH\text{-}MGR\text{-}P)$$

```
(SELECT     PNUMBER
FROM        PROJECT, DEPARTMENT, EMPLOYEE
WHERE       DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='SMITH')
UNION
(SELECT     PNO
FROM        PROJECT, WORKS_ON, EMPLOYEE
WHERE       ESSN=SSN AND LNAME= 'SMITH');
```

- List the names of employees who have no dependents.

ALL-EMPS ← π_{SSN} (EMPLOYEE)

EMPS-WITH-DEPS(SSN) ← π_{ESSN} (DEPENDENT)

EMP-WITHOUT-DEPS ← (ALL-EMPS - EMPS-WITH-DEPS)

RESULT← $\pi_{LNAME,FNAME}$ (EMPS-WITHOUT-DEPS ∞ EMPLOYEE)

SQL:

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SSN NOT IN (SELECT ESSN
                  FROM DEPENDENT);
```

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE NOT EXIXTS (SELECT *
                  FROM DEPENDENT
                  WHERE ESSN=SSN);
```

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SSN IN ((SELECT SSN
               FROM EMPLOYEE)
        MINUS (SELECT ESSN
               FROM DEPENDENT));
```

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

1. List the names of managers who have at least one dependent

2. Find the departments in which there is no employees working on.

3. Retrieve the names of each employee who has a dependent with the same first name and same sex as the employee.

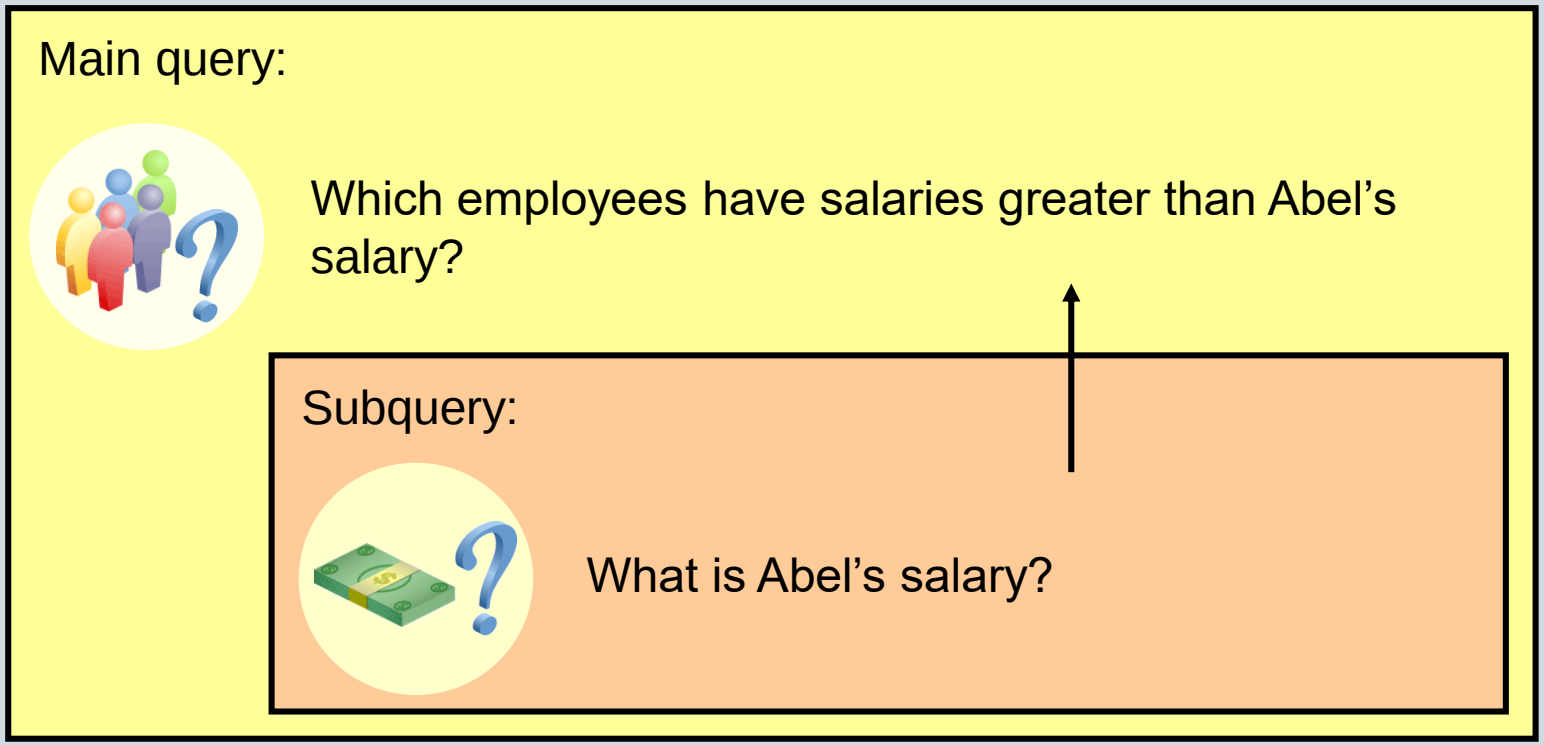
4. Retrieve the names of all employees who do not have supervisors.

5. Select the social security numbers of all employees who work on the same (project, hours) combination on some project that employee ‘ARDEN ABRAM ' (whose SSN = '113') works on.

Using Subqueries to Solve Queries

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?



Subquery Syntax

```
SELECT    select_list
FROM      table
WHERE     expr operator
                    (SELECT       select_list
                    FROM          table);
```

- The subquery (inner query) executes *before* the main query (outer query).
- The result of the subquery is used by the main query.

Q: Who has a salary greater than Abel's?

Using a Subquery

Who has a salary greater than Abel's?

```
SELECT last_name, salary
FROM   employees
WHERE  salary >   11000
                  (SELECT salary
                   FROM   employees
                   WHERE  last_name = 'Abel');
```

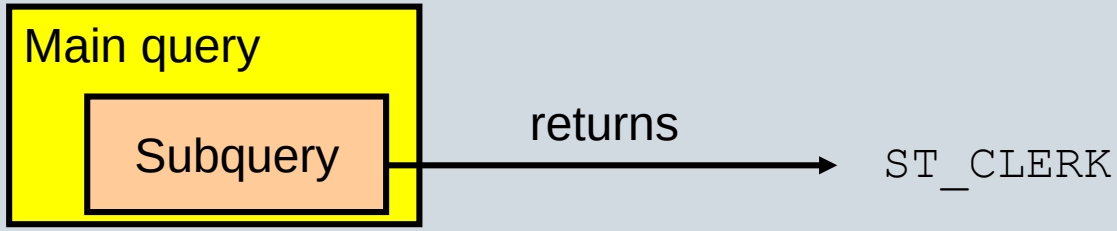
	LAST_NAME	SALARY
1	Hartstein	13000
2	Higgins	12000
3	King	24000
4	Kochhar	17000
5	De Haan	17000

Rules for Using Subqueries

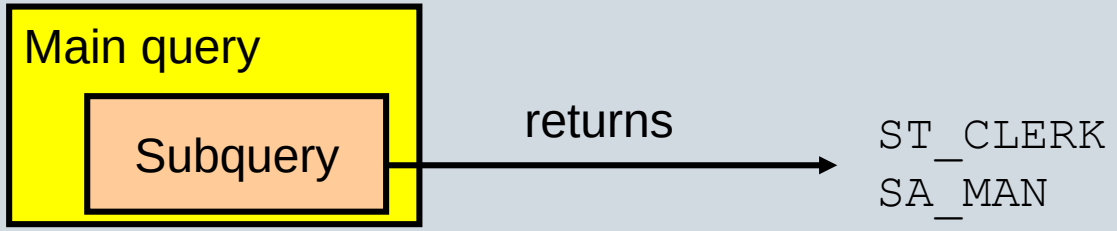
- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition for readability. (However, the subquery can appear on either side of the comparison operator.)
- Use single-row operators with single-row subqueries and multiple-row operators with multiple-row subqueries.

Types of Subqueries

- Single-row subquery



- Multiple-row subquery



Single-Row Subqueries

- Return only one row
- Use single-row comparison operators

Operator	Meaning
=	Equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

Executing Single-Row Subqueries

Q: Find the employees who have same job_id but have greater salary than ‘Taylor’

Executing Single-Row Subqueries

Find the employees who have same job_id but have greater salary than ‘Taylor’.

```
SELECT last_name, job_id, salary
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Taylor')
AND    salary >
                (SELECT salary
                 FROM   employees
                 WHERE  last_name = 'Taylor');
```

SA_REP

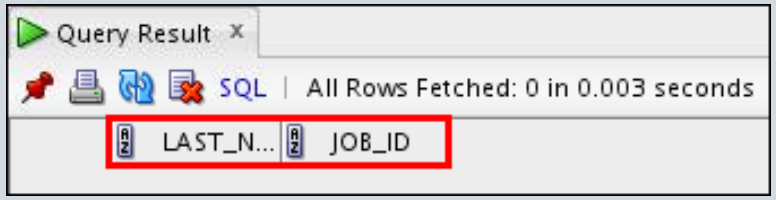
8600

	LAST_NAME	JOB_ID	SALARY
1	Abel	SA_REP	11000

No Rows Returned by the Inner Query

Find the employees who have same job_id with ‘Haas’.

```
SELECT last_name, job_id
FROM   employees
WHERE  job_id =
                (SELECT job_id
                 FROM   employees
                 WHERE  last_name = 'Haas');
```



Subquery returns no rows because there is no employee named “Haas.”

Multiple-Row Subqueries

- Return more than one row
- Use multiple-row comparison operators

Operator	Meaning
IN	Equal to any member in the list
ANY	Must be preceded by =, !=, >, <, <=, >=. Returns TRUE if at least one element exists in the result-set of the Subquery for which the relation is TRUE.
ALL	Must be preceded by =, !=, >, <, <=, >=. Returns TRUE if the relation is TRUE for all elements in the result set of the Subquery.

ANY & ALL OPERATIONS

OPERATION	MEANING	POSSIBLE SCOPE OF THE ATTRIBUTE
Attribute1 **> ANY** set1	Attribute1 **> MIN** element of set	Within the set or outside of the set
Attribute1 **< ANY** set1	Attribute1 **< MAX** element of set	Within the set or outside of the set
Attribute1 **> ALL** set1	Attribute1 **> MAX** element of set	Outside of the set
Attribute1 **< ALL** set1	Attribute1 **< MIN** element of set	Outside of the set
Attribute1 **= ANY** set1	Attribute1 **IN** set (member of the set)	Inside the set
Attribute1 **= ALL** set1	Never correct	

Using the ANY Operator in Multiple-Row Subqueries

Find the names, job IDs and salary of those employees whose salary is smaller than any one of the salaries of IT_Programmers.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ANY
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	144	Vargas	ST_CLERK	2500
2	143	Matos	ST_CLERK	2600
3	142	Davies	ST_CLERK	3100
4	141	Rajs	ST_CLERK	3500
5	200	Whalen	AD_ASST	4400

...

9	206	Gietz	AC_ACCOUNT	8300
10	176	Taylor	SA_REP	8600

Using the ALL Operator in Multiple-Row Subqueries

Find the names, job IDs and salary of those employees whose salary is smaller than ALL one of the salaries of IT_Programmers.

```
SELECT employee_id, last_name, job_id, salary
FROM   employees
WHERE  salary < ALL
                (SELECT salary
                 FROM   employees
                 WHERE  job_id = 'IT_PROG')
AND    job_id <> 'IT_PROG';
```

9000, 6000, 4200

	EMPLOYEE_ID	LAST_NAME	JOB_ID	SALARY
1	141	Rajs	ST_CLERK	3500
2	142	Davies	ST_CLERK	3100
3	143	Matos	ST_CLERK	2600
4	144	Vargas	ST_CLERK	2500

Q: Find the managers whose salary is less than any one of his/her employees.

Find the managers whose salary is less than any one of his/her employees.

```
SELECT employee_id, salary, last_name, job_id, salary
FROM   employees M
WHERE  M. Salary < ANY (SELECT salary
                        FROM employees W
                        WHERE W.manager_id =  M. Employee_id) ;
```

```
SELECT m.first_name, m.last_name, m.salary, w.last_name, w.salary
FROM employees m JOIN employees w ON (w.manager_id = m.employee_id)
WHERE m.salary < w.salary;
```

Q: Find the names of managers who has the workers earning more then 10.000

Using the `EXISTS` Operator

Find the names of managers who has the workers earning more then 10.000

```
SELECT employee_id, salary, last_name
FROM employees M
WHERE EXISTS
   (SELECT employee_id
    FROM employees W
    WHERE (W.manager_id = M.employee_id) AND W.salary > 10000);
```

	EMPLOYEE_ID	SALARY	LAST_NAME
1	100	24000	King
2	149	10500	Zlotkey
3	101	17000	Kochhar

Q: Find the names of those departments where there is no employees working for them.

Using the `EXISTS` Operator

Find the names of those departments where there is no employees working for them.

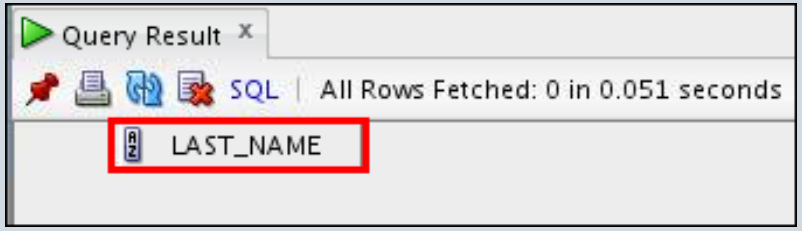
```
SELECT * FROM departments
WHERE NOT EXISTS
  (SELECT *
   FROM employees
   WHERE employees.department_id = departments.department_id);
```

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	190	Contracting	(null)	1700

Null Values in a Subquery

Find the names of employees who are not manager.
NOTE: What happens if sub query has at least one null value?)

```
SELECT emp.last_name
FROM   employees emp
WHERE  emp.employee_id NOT IN
                          (SELECT mgr.manager_id
                           FROM   employees mgr);
```



Subquery returns no rows because one of the values returned by a subquery is Null.

SOLVE THE FOLLOWING QUERIES WITH SQL BY USING COMPANY DATABASE:

1. Find the names of employees whose salary is better than any one of the salary of the employees working in "TOURISM" department;

2. Find the names of employees whose salary greater than all the employees' salaries in TOURISM department.

3. Find the names of employees whose salary is worst than all of the salary of the employees working in "TOURISM" department;