Using Subqueries to Solve Queries

Objectives

After completing this lesson, you should be able to do the following:

- **Define subqueries**
- **Describe the types of problems that subqueries can solve**
- **List the types of subqueries**
- **Write single-row and multiple-row subqueries**

Using a Subquery to Solve a Problem

Who has a salary greater than Abel's?

Main query:

Which employees have salaries greater than Abel's salary?

Subquery:

What is Abel's salary?

Subquery Syntax

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

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

Using a Subquery

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

Guidelines for Using Subqueries

- Enclose subqueries in parentheses.
- Place subqueries on the right side of the comparison condition.
- The ORDER BY clause in the subquery is not needed unless you are performing Top-N analysis.
- Use single-row operators with single-row subqueries, and use multiple-row operators with multiple-row subqueries.

Types of Subqueries

Single-row subquery

Main query

Subquery $\xrightarrow{\text{returns}}$ **ST_CLERK**

Multiple-row subquery

Main query

Subquery $\xrightarrow{\text{returns}}$ **ST_CLERK**

SA_MAN

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

```
SELECT last_name, job_id, salary
FROM employees
WHERE job_id =
                                                returns
              (SELECT job_id          ──────────────────→  ST_CLERK
              FROM employees
              WHERE employee_id = 141)
AND salary >
                                                returns
              (SELECT salary              ───────────→  2600
              FROM employees
              WHERE employee_id = 143);
```

Using Group Functions in a Subquery

```
SELECT last_name, job_id, salary
FROM employees
WHERE salary =
                    (SELECT MIN(salary)  ──────────→  2500
                    FROM employees);
```

The HAVING Clause with Subqueries

- The Oracle server executes subqueries first.
- The Oracle server returns results into the HAVING clause of the main query.

```
SELECT department_id, MIN(salary)
FROM employees
GROUP BY department_id
HAVING MIN(salary) >        ← 2500
                   (SELECT MIN(salary)
                   FROM employees
                   WHERE department_id = 50);
```

What Is Wrong with This Statement?

```
SELECT employee_id, last_name
FROM employees
WHERE salary =
                (SELECT MIN(salary)
                FROM employees
                GROUP BY department_id);
```

Single-row operator with multiple-row subquery

Will This Statement Return Rows?

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

Subquery returns no values.

Multiple-Row Subqueries

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

Operator	Meaning
IN	Equal to any member in the list
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

Using the ANY Operator in Multiple-Row Subqueries

```
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

Using the ALL Operator in Multiple-Row Subqueries

```
SELECT employee_id, last_name, job_id, salary
FROM employees
WHERE salary < ALL  ←――――――――――  9000, 6000, 4200
                    (SELECT salary
                    FROM employees
                    WHERE job_id = 'IT_PROG')
AND job_id <> 'IT_PROG';
```

Null Values in a Subquery

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

Using the Set Operators

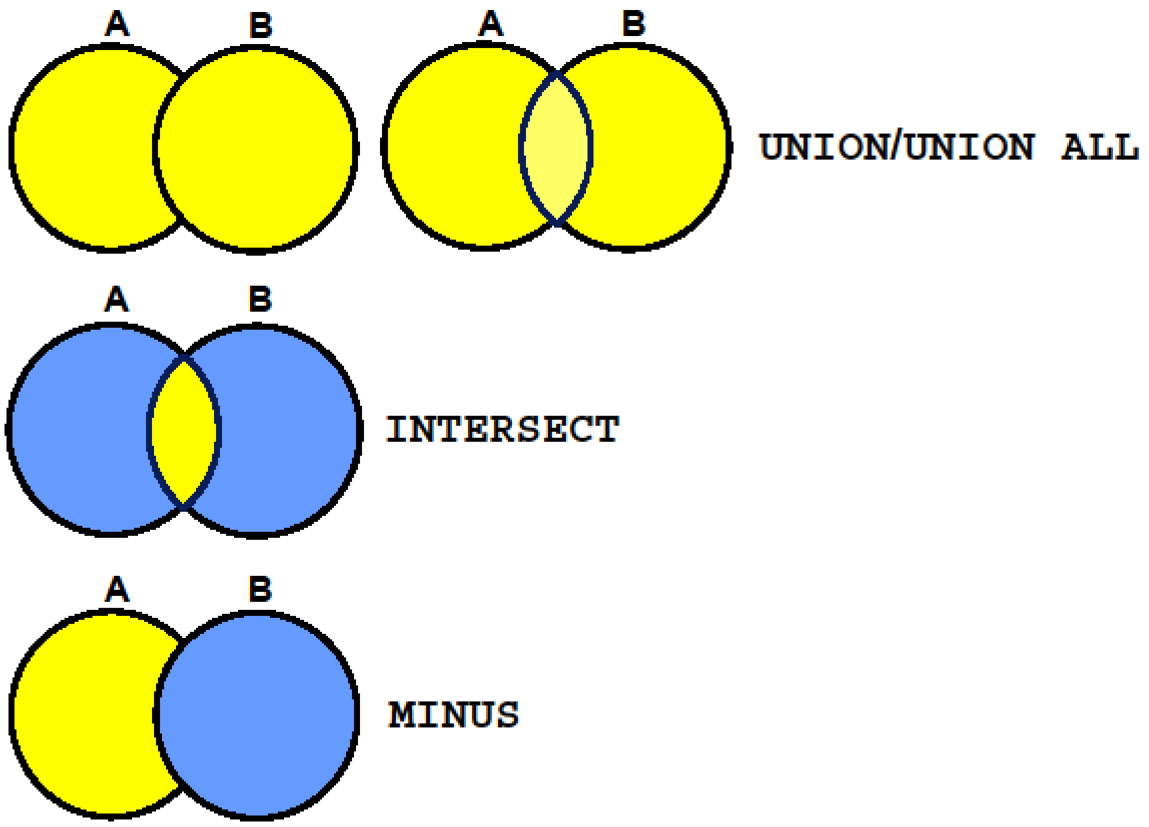
Objectives

After completing this lesson, you should be able to do the following:

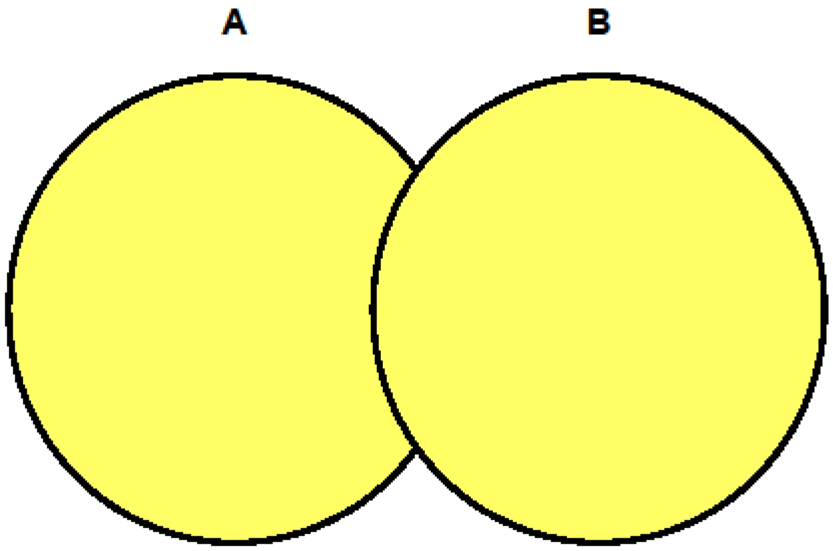
- **Describe set operators**
- **Use a set operator to combine multiple queries into a single query**
- **Control the order of rows returned**

Set Operators

Set Operators



`UNION` Operator



The `UNION` operator returns results from both queries after eliminating duplications.

Using the UNION Operator

Create Job_history table.

```
SQL> desc job_history
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 EMPLOYEE_ID                               NOT NULL NUMBER(6)
 START_DATE                                NOT NULL DATE
 END_DATE                                  NOT NULL DATE
 JOB_ID                                    NOT NULL VARCHAR2(10)
 DEPARTMENT_ID                                      NUMBER(4)
```

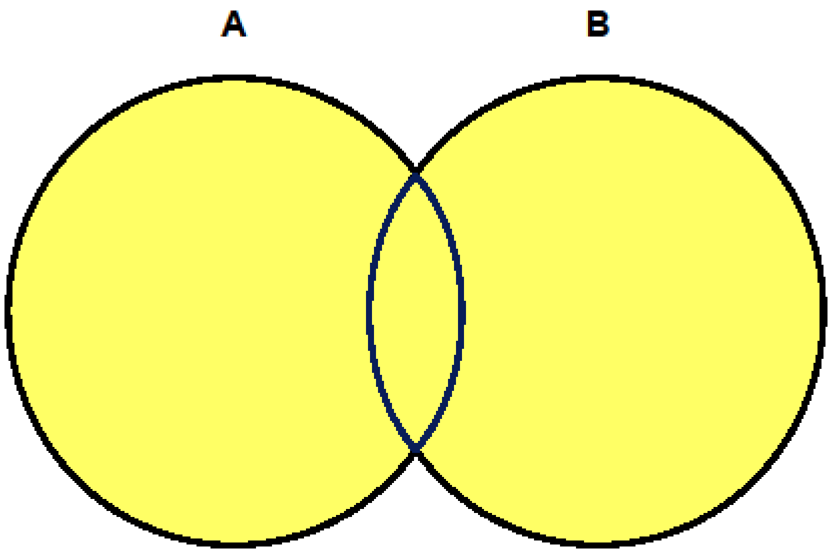
```
SQL> select * from job_history;

EMPLOYEE_ID   START_DATE   END_DATE          JOB_ID     DEPARTMENT_ID
----------- -------- -------- ---------- -----------------------------------------------------
      102         93-01-13       98-07-24             IT_PROG               60
      101         89-09-21       93-10-27             AC_ACCOUNT          110
      101         93-10-28       97-03-15             AC_MGR              110
      201         96-02-17       99-12-19             MK_REP                20
      114         98-03-24       99-12-31             ST_CLERK              50
      122         99-01-01       99-12-31             ST_CLERK              50
      200         87-09-17       93-06-17             AD_ASST               90
      176         98-03-24       98-12-31             SA_REP                80
      176         99-01-01       99-12-31             SA_MAN                80
      200         94-07-01       98-12-31             AC_ACCOUNT            90

10 rows selected.
```

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;
```

`UNION ALL` Operator



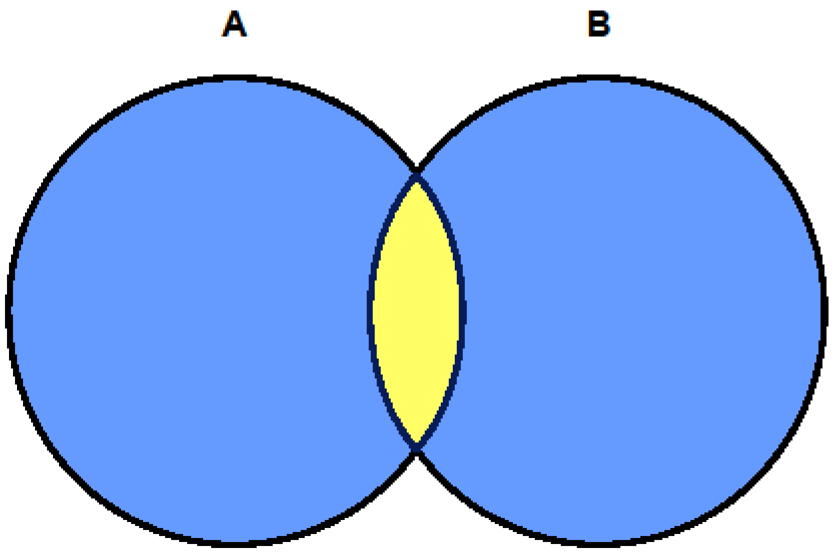
The `UNION ALL` operator returns results from both queries, including all duplications.

Using the UNION ALL Operator

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;
```

INTERSECT Operator



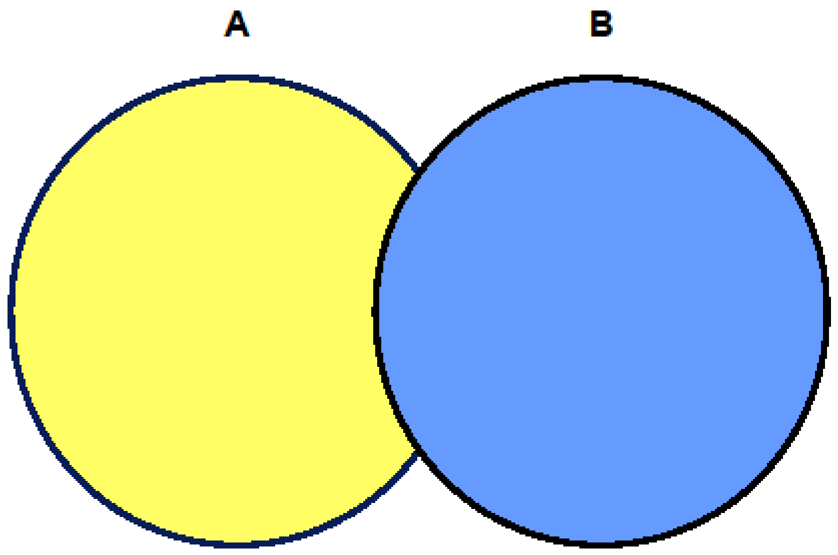
The INTERSECT operator returns rows that are common to both queries.

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 job title when they were initially hired (that is, they changed jobs but have now gone back to doing their original job).

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

`MINUS` Operator



The `MINUS` operator returns rows in the first query that are not present in the second query.

MINUS Operator

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

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

Set Operator Guidelines

- The expressions in the SELECT lists must match in number and data type.
- Parentheses can be used to alter the sequence of execution.
- The ORDER BY clause:

– Can appear only at the very end of the statement
– Will accept the column name, aliases from the first
SELECT statement, or the positional notation

The 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.

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;
```