Chapter 6

Advanced SQL

In this chapter, you will learn:

- The various uses of database VIEWS
- Extracting data from multiple table through the use of OUTER JOIN for matching and unmatched data
- The use of Relational Set Operators

The HR database used in Chapter 3 is referred in this chapter

View

- A view, like a table, is a database object
- However, views are not "real" tables
- They are logical representations of existing tables or of another view
- Views contain no data of their own
- They function as a window through which data from tables can be viewed or changed

View

- The tables on which a view is based are called "base" tables
- The view is a query stored as a SELECT statement in the data dictionary

```
CREATE VIEW view_employees
AS SELECT employee_id,first_name, last_name, email
   FROM employees
   WHERE employee_id BETWEEN 100 and 124;
```

```
SELECT *
FROM view_employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL
100	Steven	King	SKING
101	Neena	Kochhar	NKOCHHAR
102	Lex	De Haan	LDEHAAN
124	Kevin	Mourgos	KMOURGOS
103	Alexander	Hunold	AHUNOLD
104	Bruce	Ernst	BERNST
107	Diana	Lorentz	DLORENTZ

ORACLE Academy

Why Use Views?

- Views restrict access to base table data because the view can display selective columns from the table
- Views can be used to reduce the complexity of executing queries based on more complicated SELECT statements
- For example, the creator of the view can construct join statements that retrieve data from multiple tables
- The user of the view neither sees the underlying code nor how to create it
- The user, through the view, interacts with the database using simple queries

Why Use Views?

- Views can be used to retrieve data from several tables, providing data independence for users
- Users can view the same data in different ways
- Views provide groups of users with access to data according to their particular permissions or criteria

Advantages of Views

- Security
  - Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see.
- Query Simplicity
  - A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.
- Structural simplicity
  - Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.
- Consistency
  - A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.
- Data Integrity
  - If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.
- Logical data independence.
  - View can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table.

Disadvantages of Views

- Performance
  - Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time to process.

- Update restrictions
  - When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying source tables. This is possible for simple views, but more complex views are often restricted to read-only.

Creating a View

- To create a view, embed a subquery within the CREATE VIEW statement
- The syntax of a view statement is as follows:

```
CREATE [OR REPLACE] [FORCE| NOFORCE] VIEW view  [(alias [,
   alias]...)] AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

Creating a View

OR REPLACE	Re-creates the view if it already exists.
FORCE	Creates the view whether or not the base tables exist.
NOFORCE	Creates the view only if the base table exists (default).
view_name	Specifies the name of the view.
alias	Specifies a name for each expression selected by the view's query.
subquery	Is a complete SELECT statement. You can use aliases for the columns in the SELECT list. The subquery can contain complex SELECT syntax.

Creating a View

WITH CHECK OPTION	Specifies that rows remain accessible to the view after insert or update operations.
CONSTRAINT	Is the name assigned to the CHECK OPTION constraint.
WITH READ ONLY	Ensures that no DML operations can be performed on this view.

Creating a View

- Example:

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, region_id, country_name, capitol
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

```
SELECT * FROM view_euro_countries
ORDER BY country_name;
```

COUNTRY_ID	REGION_ID	COUNTRY_NAME	CAPITOL
22	155	Bailiwick of Guernsey	Saint Peter Port
203	155	Bailiwick of Jersey	Saint Helier
387	39	Bosnia and Herzegovina	Sarajevo
420	151	Czech Republic	Prague
298	154	Faroe Islands	Torshavn
49	155	Federal Republic of Germany	Berlin
33	155	French Republic	Paris
...	...	...	...

Once the Database administrator creates a VIEW, the user can simply query the VIEW as if it was a table.

Guidelines for Creating a View

- The subquery that defines the view can contain complex SELECT syntax
- For performance reasons, the subquery that defines the view should not contain an ORDER BY clause. The ORDER BY clause is best specified when you retrieve data from the view
- You can use the OR REPLACE option to change the definition of the view without having to drop it or re-grant object privileges previously granted on it
- Aliases can be used for the column names in the subquery

CREATE VIEW Features

- Two classifications of views are used: simple and complex
- The table summarizes the features of each view

Feature	Simple Views	Complex Views
Number of tables used to derive data	One	One or more
Can contain functions	No	Yes
Can contain groups of data	No	Yes
Can perform DML operations (INSERT, UPDATE, DELETE) through a view	Yes	Not always

Simple View

- The view shown below is an example of a simple view
- The subquery derives data from only one table and it does not contain a join function or any group functions
- Because it is a simple view, INSERT, UPDATE, DELETE, and MERGE operations affecting the base table could possibly be performed through the view

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, country_name, capitol
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

Simple View

- Column names in the SELECT statement can have aliases as shown below
- Note that aliases can also be listed after the CREATE VIEW statement and before the SELECT subquery

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id AS "ID", country_name AS "Country",
      capitol AS "Capitol City"
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

```
CREATE OR REPLACE VIEW view_euro_countries("ID", "Country",
    "Capitol City")
AS SELECT country_id, country_name, capitol
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

Simple View

- It is possible to create a view whether or not the base tables exist
- Adding the word FORCE to the CREATE VIEW statement creates the view
- As a DBA, this option could be useful during the development of a database, especially if you are waiting for the necessary privileges to the referenced object to be granted shortly
- The FORCE option will create the view despite it being invalid
- The NOFORCE option is the default when creating a view

Complex View

- Complex views are views that can contain group functions and joins
- The following example creates a view that derives data from two tables

```
CREATE OR REPLACE VIEW view_euro_countries
   ("ID", "Country", "Capitol City", "Region")
AS SELECT c.country_id, c.country_name, c.capitol,
r.region_name
   FROM wf_countries c JOIN wf_world_regions r
   USING (region_id)
   WHERE location LIKE '%Europe';
```

```
SELECT *
FROM view_euro_countries;
```

Complex View

ID	Country	Capitol City	Region
375	Republic of Belarus	Minsk	Eastern Europe
48	Republic of Poland	Warsaw	Eastern Europe
421	Slovak Republic	Bratislava	Eastern Europe
36	Republic of Hungary	Budapest	Eastern Europe
90	Republic of Turkey	Ankara	Eastern Europe
40	Romania	Bucharest	Eastern Europe
373	Republic of Moldova	Chisinau	Eastern Europe
370	Republic of Lithuania	Vilnius	Eastern Europe
371	Republic of Latvia	Riga	Eastern Europe
372	Republic of Estonia	Tallinn	Eastern Europe
...	...	...	...

Complex View

- Group functions can also be added to complex-view statements

```
CREATE OR REPLACE VIEW view_high_pop
   ("Region ID", "Highest
population")
AS SELECT region_id, MAX(population)
   FROM wf_countries
   GROUP BY region_id;
```

```
SELECT * FROM view_high_pop;
```

Region ID	Highest population
5	188078227
9	20264082
11	131859731
13	107449525
14	74777981
15	78887007
17	62660551
18	44187637
21	298444215
...	...

Modifying a View

- To modify an existing view without having to drop then re-create it, use the OR REPLACE option in the CREATE VIEW statement
- The old view is replaced by the new version
- For example:

```
CREATE OR REPLACE VIEW view_euro_countries
AS SELECT country_id, region_id, country_name, capitol
   FROM wf_countries
   WHERE location LIKE '%Europe';
```

Views with CHECK Option

- The view is defined without the WITH CHECK OPTION

```
CREATE VIEW view_dept50
AS SELECT department_id, employee_id, first_name, last_name,
salary
   FROM copy_employees
   WHERE department_id = 50;
```

- Using the view, employee_id 124 has his department changed to dept_id 90

```
UPDATE view_dept50
SET department_id = 90
WHERE employee_id = 124;
```

```
1 row(s) updated.
```

- The update succeeds, even though this employee is now not part of the view

Views with CHECK Option

- The WITH CHECK OPTION ensures that DML operations performed on the view stay within the domain of the view
- Any attempt to change the department number for any row in the view fails because it violates the WITH CHECK OPTION constraint
- Notice in the example below that the WITH CHECK OPTION CONSTRAINT was given the name view_dept50_check

```
CREATE OR REPLACE VIEW view_dept50
AS SELECT department_id, employee_id, first_name, last_name,
salary
   FROM employees
   WHERE department_id = 50
WITH CHECK OPTION CONSTRAINT view_dept50_check;
```

Views with CHECK Option

- Now, if we attempt to modify a row in the view that would take it outside the domain of the view, an error is returned

```
UPDATE view_dept50
SET department_id = 90
WHERE employee_id = 124;
```

```
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Views with READ ONLY

- The WITH READ ONLY option ensures that no DML operations occur through the view
- Any attempt to execute an INSERT, UPDATE, or DELETE statement will result in an Oracle server error

```
CREATE OR REPLACE VIEW view_dept50
AS SELECT department_id, employee_id, first_name, last_name,
salary
 FROM employees
 WHERE department_id = 50
WITH READ ONLY;
```

DML Restrictions

- Simple views and complex views differ in their ability to allow DML operations through a view
- For simple views, DML operations can be performed through the view
- For complex views, DML operations are not always allowed
- The following three rules must be considered when performing DML operations on views

DML Restrictions

- You cannot remove a row from an underlying base table if the view contains any of the following:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM Keyword

DML Restrictions

- You cannot modify data through a view if the view contains:
  - Group functions
  - A GROUP BY clause
  - The DISTINCT keyword
  - The pseudocolumn ROWNUM keyword
  - Columns defined by expressions

DML Restrictions

- You cannot add data through a view if the view:
  - includes group functions
  - includes a GROUP BY clause
  - includes the DISTINCT keyword
  - includes the pseudocolumn ROWNUM keyword
  - includes columns defined by expressions
  - does not include NOT NULL columns in the base tables

Deleting a View

- Because a view contains no data of its own, removing it does not affect the data in the underlying tables
- If the view was used to INSERT, UPDATE, or DELETE data in the past, those changes to the base tables remain
- Deleting a view simply removes the view definition from the database

Deleting a View

- Remember, views are stored as SELECT statements in the data dictionary
- Only the creator or users with the DROP ANY VIEW privilege can remove a view
- The SQL syntax to remove a view is:

```
DROP VIEW viewname;
```

Inline Views

- Inline views are also referred to as subqueries in the FROM clause
- You insert a subquery in the FROM clause just as if the subquery was a table name
- Inline views are commonly used to simplify complex queries by removing join operations and condensing several queries into one

Inline Views

- As shown in the example below, the FROM clause contains a SELECT statement that retrieves data much like any SELECT statement
- The data returned by the subquery is given an alias (d), which is then used in conjunction with the main query to return selected columns from both query sources

```
SELECT e.last_name, e.salary, e.department_id, d.maxsal
FROM employees e,
       (SELECT department_id, max(salary) maxsal
        FROM employees
        GROUP BY department_id) d
WHERE e.department_id = d.department_id
AND e.salary = d.maxsal;
```

ORACLE Academy

This inline view finds the highest salary for each department, and the query then displays the name of the employee with that salary.

An inline view must have an alias (in this example, "d") because it functions like a table name in the FROM clause, and "SELECT department_id, max(salary) ..." is not a valid table name!

TOP-N-ANALYSIS

- Top-n-analysis is a SQL operation used to rank results
- The use of top-n-analysis is useful when you want to retrieve the top 5 records, or top-n records, of a result set returned by a query

```
SELECT ROWNUM AS "Longest employed", last_name, hire_date
FROM employees
WHERE ROWNUM <=5
ORDER BY hire_date;
```

Longest employed	LAST_NAME	HIRE_DATE
1	King	17-Jun-1987
4	Whalen	17-Sep-1987
2	Kochhar	21-Sep-1989
3	De Haan	13-Jan-1993
5	Higgins	07-Jun-1994

ORACLE Academy

The results of this query are not however what you would expect. The reason for this is that the ORDER BY clause always executes last, so the rows are ordered after they are given a number. The next slide demonstrates how to resolve this issue.

TOP-N-ANALYSIS

- The top-n-analysis query uses an inline view (a subquery) to return a result set
- You can use ROWNUM in your query to assign a row number to the result set
- The main query then uses ROWNUM to order the data and return the top five

```
SELECT ROWNUM AS "Longest employed", last_name, hire_date
FROM (SELECT last_name, hire_date
      FROM employees
      ORDER BY hire_date)
WHERE ROWNUM <=5;
```

ORACLE Academy

By using an inline view, the ORDER BY clause is performed before the ROWNUM is added.

TOP-N-ANALYSIS

Longest employed	LAST_NAME	HIRE_DATE
1	King	17-Jun-1987
2	Whalen	17-Sep-1987
3	Kochhar	21-Sep-1989
4	Hunold	03-Jan-1990
5	Ernst	21-May-1991

- In the example above, the inline view first selects the list of last_names and hire_dates of the employees:

```
(SELECT last_name, hire_date FROM employees...
```

- Then the inline view orders the years from oldest to newest

```
...ORDER BY hire_date)
```

TOP-N-ANALYSIS

- The outer query WHERE clause is used to restrict the number of rows returned and must use a < or <= operator

```
SELECT ROWNUM AS "Longest employed", last_name, hire_date
FROM (SELECT last_name, hire_date
      FROM employees
      ORDER BY hire_date)
WHERE ROWNUM <=5;
```

Longest employed	LAST_NAME	HIRE_DATE
1	King	17-Jun-1987
2	Whalen	17-Sep-1987
3	Kochhar	21-Sep-1989
4	Hunold	03-Jan-1990
5	Ernst	21-May-1991

Joining tables with matching and unmatched data

INNER And OUTER Joins

- In ANSI-99 SQL, a join of two or more tables that returns only the matched rows is called an inner join
- When a join returns the unmatched rows as well as the matched rows, it is called an outer join
- Outer join syntax uses the terms "left, full, and right"
- These names are associated with the order of the table names in the FROM clause of the SELECT statement

NATURAL JOIN, JOIN ON and JOIN USING are types of inner joins.

LEFT and RIGHT OUTER Joins

- In the example shown of a left outer join, note that the table name listed to the left of the words "left outer join" is referred to as the "left table."

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

LAST_NAME	DEPT_ID	DEPT_NAME
Whalen	10	Administration
Fay	20	Marketing
...		
Zlotkey	80	Sales
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant	-	-

Column names in sample output have been abbreviated in order to fit on the slide.

LEFT and RIGHT OUTER Joins

- This query will return all employee last names, both those that are assigned to a department and those that are not

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

LAST_NAME	DEPT_ID	DEPT_NAME
Whalen	10	Administration
Fay	20	Marketing
...		
Zlotkey	80	Sales
De Haan	90	Executive
Kochhar	90	Executive
King	90	Executive
Gietz	110	Accounting
Higgins	110	Accounting
Grant	-	-

ORACLE Academy

Column names in sample output have been abbreviated in order to fit on the slide.

LEFT and RIGHT OUTER Joins

- This right outer join would return all department IDs and department names, both those that have employees assigned to them and those that do not

```
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	DEPT_ID	DEPT_NAME
Whalen	10	Administration
Hartstein	20	Marketing
…		
King	90	Executive
Kochhar	90	Executive
De Haan	90	Executive
Higgins	110	Accounting
Gietz	110	Accounting
-	190	Contracting

ORACLE Academy

Column names in sample output have been abbreviated in order to fit on the slide.

FULL OUTER Join

- It is possible to create a join condition to retrieve all matching rows and all unmatched rows from both tables
- Using a full outer join solves this problem
- The result set of a full outer join includes all rows from a left outer join and all rows from a right outer join combined together without duplication

FULL OUTER Join Example

- The example shown is a 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	DEPT_ID	DEPT_NAME
King	90	Executive
Kochhar	90	Executive
...		
Taylor	80	Sales
Grant	-	-
Mourgos	50	Shipping
...		
Fay	20	Marketing
-	190	Contracting

ORACLE
Academy

Column names in sample output have been abbreviated in order to fit on the slide.

Join Scenario

- Construct a join to display a list of employees, their current job_id and any previous jobs they may have held
- The job_history table contains details of an employee's previous jobs

```
SELECT last_name, e.job_id AS "Job", jh.job_id AS "Old job",
end_date
FROM employees e LEFT OUTER JOIN job_history jh
  ON(e.employee_id = jh.employee_id);
```

LAST_NAME	Job	Old job	END_DATE
King	AD_PRES	-	-
Kochhar	AD_VP	AC_MGR	15-Mar-1997
Kochhar	AD_VP	AC_ACCOUNT	27-Oct-1993
De Haan	AD_VP	IT_PROG	24-Jul-1998
Whalen	AD_ASST	AD_ASST	17-Jun-1993
Whalen	AD_ASST	AC_ACCOUNT	31-Dec-1998
Higgins	AC_MGR	-	-

Using the Set Operators

Purpose

- Set operators are used to combine the results from different SELECT statements into one single result output
- Sometimes you want a single output from more than one table
- If you join the tables, the rows that meet the join criteria are returned, but what if a join will return a result set that doesn't meet your needs?
- This is where SET operators come in
- They can return the rows found in multiple SELECT statements, the rows that are in one table and not the other, or the rows common to both statements

Setting the Stage

- In order to explain the SET operators, the following two lists will be referred to throughout this lesson:

  A = {1, 2, 3, 4, 5}

  B = {4, 5, 6, 7, 8}

- Or in reality: two tables, one called A and one called B

A

A_ID
1
2
3
4
5

B

B_ID
4
5
6
7
8

ORACLE Academy

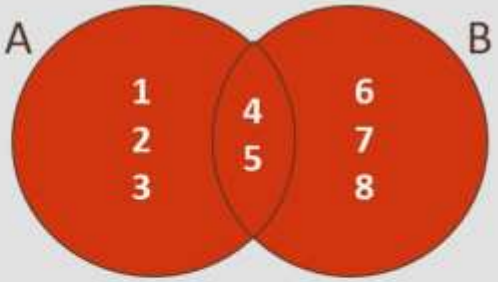
Rules to Remember

- There are a few rules to remember when using SET operators:
  - The number of columns and the data types of the columns must be identical in all of the SELECT statements used in the query
  - The names of the columns need not be identical
  - Column names in the output are taken from the column names in the first SELECT statement
- So, any column aliases should be entered in the first statement as you would want to see them in the finished report

UNION

- The UNION operator returns all rows from both tables, after eliminating duplicates

```
SELECT a_id
FROM a
 UNION
SELECT b_id
FROM b;
```

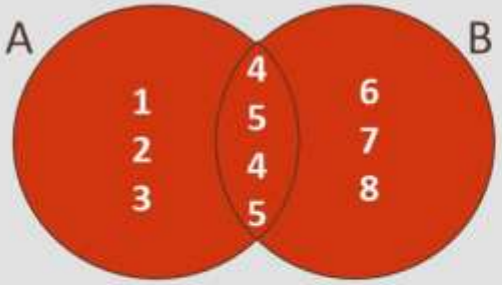


- The result of listing all elements in A and B eliminating duplicates is {1, 2, 3, 4, 5, 6, 7, 8}
- If you joined A and B you would get only {4, 5}. You would have to perform a full outer join to get the same list as above

UNION ALL

- The UNION ALL operator returns all rows from both tables, without eliminating duplicates

```
SELECT a_id
FROM a
 UNION  ALL
SELECT b_id
FROM b;
```

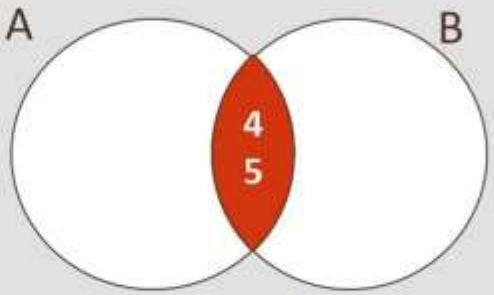


- The result of listing all elements in A and B without eliminating duplicates is {1, 2, 3, 4, 5, 4, 5, 6, 7, 8}

INTERSECT

- The INTERSECT operator returns all rows common to both tables

```
SELECT a_id
FROM a
 INTERSECT
SELECT b_id
FROM b;
```

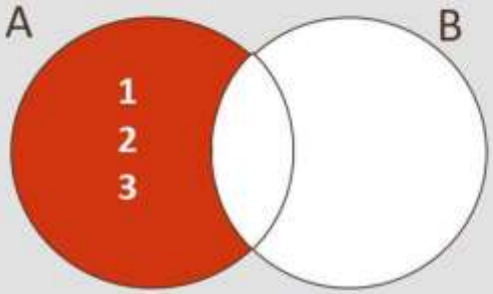


- The result of listing all elements found in both A and B is {4, 5}

MINUS

- The MINUS operator returns all rows found in one table but not the other

```
SELECT a_id
FROM a
 MINUS
SELECT b_id
FROM b;
```



- The result of listing all elements found in A but not B is {1, 2, 3}
- The result of B MINUS A would give {6, 7, 8}

Set Operator Examples

- Sometimes if you are selecting rows from tables that do not have columns in common, you may have to create your own columns in order to match the number of columns in the queries
- The easiest way to do this is to include one or more NULL values in the select list
- Remember to give each one a suitable alias and matching data type

Set Operator Examples

- For example:
  - The employees table contains a hire date, employee id and a job id
  - The job history table contains employee id and job id, but does not have a hire date column
  - The two tables have the employee id and job id in common, but job history does not have a start date
- You can use the TO_CHAR(NULL) function to create matching columns as in the next slide

Set Operator Examples

```
SELECT hire_date, employee_id, job_id
FROM employees
 UNION
SELECT TO_DATE(NULL),employee_id,
job_id
FROM job_history;
```

HIRE_DATE	EMPLOYEE_ID	JOB_ID
17-Jun-1987	100	AD_PRES
17-Sep-1987	200	AD_ASST
21-Sep-1989	101	AD_VP
03-Jan-1990	103	IT_PROG
21-May-1991	104	IT_PROG
13-Jan-1993	102	AD_VP
07-Jun-1994	205	AC_MGR
07-Jun-1994	206	AC_ACCOUNT
17-Oct-1995	141	ST_CLERK
17-Feb-1996	201	MK_MAN
11-May-1996	174	SA_REP
29-Jan-1997	142	ST_CLERK
17-Aug-1997	202	MK_REP
15-Mar-1998	143	ST_CLERK
24-Mar-1998	176	SA_REP
09-Jul-1998	144	ST_CLERK
07-Feb-1999	107	IT_PROG
24-May-1999	178	SA_REP
16-Nov-1999	124	ST_MAN
29-Jan-2000	149	SA_MAN
-	101	AC_ACCOUNT
-	101	AC_MGR
-	102	IT_PROG
-	114	ST_CLERK
-	122	ST_CLERK
-	176	SA_MAN
-	176	SA_REP
-	200	AC_ACCOUNT
-	200	AD_ASST
-	201	MK_REP

Set Operator Examples

- The keyword NULL can be used to match columns in a SELECT list
- One NULL is included for each missing column
- Furthermore, NULL is formatted to match the data type of the column it is standing in for, so TO_CHAR, TO_DATE, or TO_NUMBER functions are used to achieve identical SELECT lists

SET Operations ORDER BY

- If you want to control the order of the returned rows when using SET operators in your query, the ORDER BY statement must only be used once, in the last SELECT statement in the query
- Using the previous query example, we could ORDER BY employee_id to see the jobs each employee has held

```
SELECT hire_date, employee_id, job_id
FROM employees
 UNION
SELECT TO_DATE(NULL),employee_id, job_id
FROM job_history
ORDER BY employee_id;
```

SET Operations ORDER BY

```
SELECT hire_date, employee_id, job_id
FROM employees
 UNION
SELECT TO_DATE(NULL),employee_id, job_id
FROM job_history
ORDER BY employee_id;
```

HIRE_DATE	EMPLOYEE_ID	JOB_ID
17-Jun-1987	100	AD_PRES
21-Sep-1989	101	AD_VP
-	101	AC_ACCOUNT
-	101	AC_MGR
13-Jan-1993	102	AD_VP
-	102	IT_PROG
03-Jan-1990	103	IT_PROG
21-May-1991	104	IT_PROG
07-Feb-1999	107	IT_PROG
-	114	ST_CLERK
...	...	...

ORACLE Academy

Note: for ease of reading, only a partial query result is shown.

SET Operations ORDER BY

- We could improve the readability of the output, by including the start date and end date columns from the job history table, to do this, we would need to match the columns in both queries by adding two more TO_DATE(NULL) columns to the first query

```
SELECT hire_date, employee_id, TO_DATE(null) start_date,
   TO_DATE(null) end_date, job_id, department_id
FROM   employees
 UNION
SELECT TO_DATE(null), employee_id, start_date, end_date,
job_id, department_id
FROM job_history
ORDER BY employee_id;
```

ORACLE
Academy

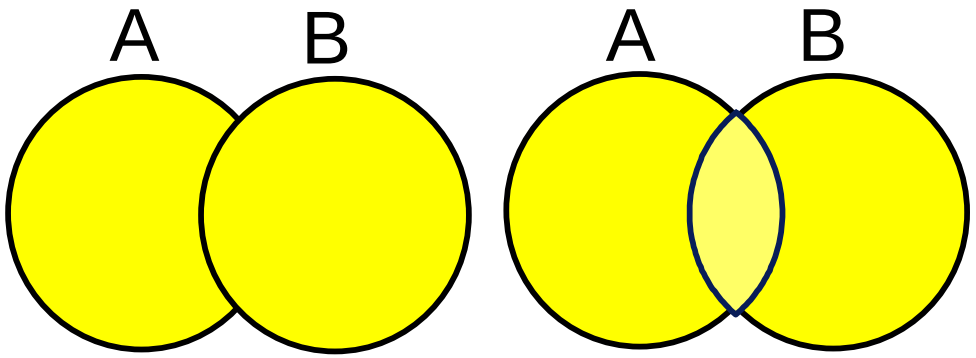
As the column headings for the query output are taken from the first query, they have been given aliases of the same name as the matching columns in the second query.

SET Operations ORDER BY

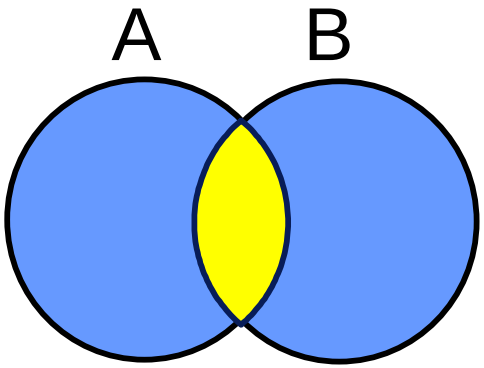
HIRE_DATE	EMPLOYEE_ID	START_DATE	END_DATE	JOB_ID	DEPARTMENT_ID
17-Jun-1987	100	-	-	AD_PRES	90
21-Sep-1989	101	-	-	AD_VP	90
-	101	21-Sep-1989	27-Oct-1993	AC_ACCOUNT	110
-	101	28-Oct-1993	15-Mar-1997	AC_MGR	110
13-Jan-1993	102	-	-	AD_VP	90
-	102	13-Jan-1993	24-Jul-1998	IT_PROG	60
03-Jan-1990	103	-	-	IT_PROG	60
21-May-1991	104	-	-	IT_PROG	60
07-Feb-1999	107	-	-	IT_PROG	60
-	114	24-Mar-1998	31-Dec-1999	ST_CLERK	50
-	122	01-Jan-1999	31-Dec-1999	ST_CLERK	50
16-Nov-1999	124	-	-	ST_MAN	50
17-Oct-1995	141	-	-	ST_CLERK	50
29-Jan-1997	142	-	-	ST_CLERK	50
15-Mar-1998	143	-	-	ST_CLERK	50
...	...	...	...	...	...

Note: for ease of reading, only a partial query result is shown.

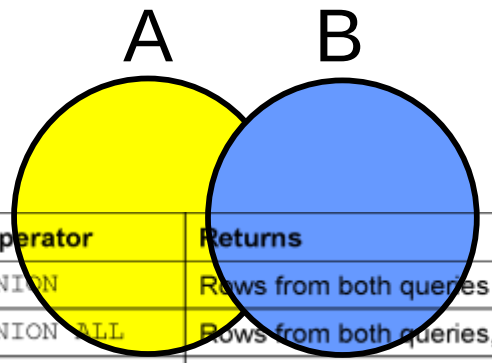
Set Operators



UNION/UNION ALL



INTERSECT



MINUS

Operator	Returns
UNION	Rows from both queries after eliminating duplications
UNION ALL	Rows from both queries, including all duplications
INTERSECT	Rows that are common to both queries
MINUS	Rows in the first query that are not present in the second query