Lecture 8

UNION OPERATORS, INTERSECTION, EXCEPTION, GROUPING SETS

Assistant professor: Yermaganbetova Madina

UNION OPERATOR

- The UNION operator combines result sets of two or more SELECT statements into a single result set.
- Removes all duplicate rows.
- Both queries must return same number of rows.
- The corresponding columns in the queries must have compatible data types.

THE FOLLOWING VENN DIAGRAM ILLUSTRATES HOW THE UNION WORKS:

SYNTAX:

```
SELECT column1, column2
FROM table1
UNION
SELECT column1, column2
FROM table2;
```

UNION ALL OPERATOR

- The UNION operator combines result sets of two or more SELECT statements into a single result set.
- Does not remove duplicate rows.
- Both queries must return same number of rows.
- The corresponding columns in the queries must have compatible data types.

SYNTAX:

```
SELECT
select_list_1
FROM table1
UNION ALL
SELECT
select_list_2
FROM table2
```

UNION AND UNION ALL EXAMPLES

```sql
SELECT first_name, last_name FROM student
UNION
SELECT first_name, last_name FROM instructor;
```

```sql
SELECT first_name, last_name FROM student
UNION ALL
SELECT first_name, last_name FROM instructor;
```

output :

276 rows

	first_name	last_name
1	Lorraine	Harty
2	Lorrane	Velasco
3	V.	Saliternan
4	Henry	Masser
5	Judy	Sethi
6	Joane	Buckberg
7	Salondra	Galik
8	Helga	Towle
9	Robert	Boyd
10	Eric	Da Silva

se Changes

output :

278 rows

	first_name	last_name
1	Fred	Crocitto
2	J.	Landry
3	Laetia	Enison
4	Angel	Moskowitz
5	Judith	Olvsade
6	Catherine	Mierzwa
7	Judy	Sethi
8	Larry	Walter
9	Maria	Martin
10	Peggy	Noviello

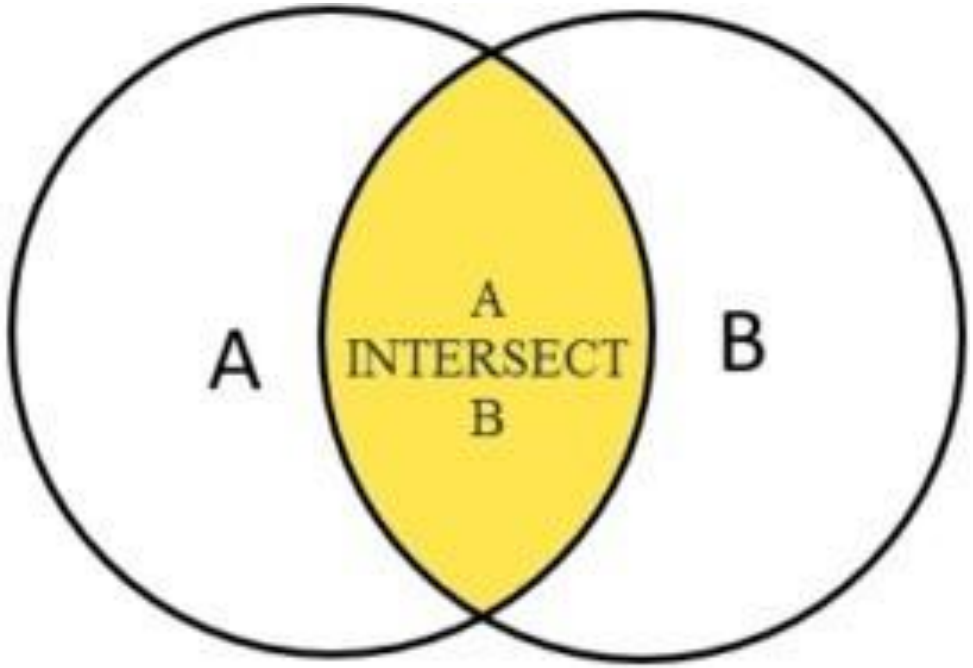
ase Changes

- UNION produces 276 rows, while UNION ALL gives 278.
- It means, we have duplications in full names of instructors and students.

INTERSECT OPERATOR

- Used to combine result set of two or more SELECT statement into a single result.
- The INTERSECT operator returns all rows in both result sets.
- The number of columns that appear in the SELECT statement must be the same.
- Data types of the columns must be compatible.

THE FOLLOWING ILLUSTRATION SHOWS THE FINAL RESULT SET PRODUCED BY THE INTERSECT OPERATOR:



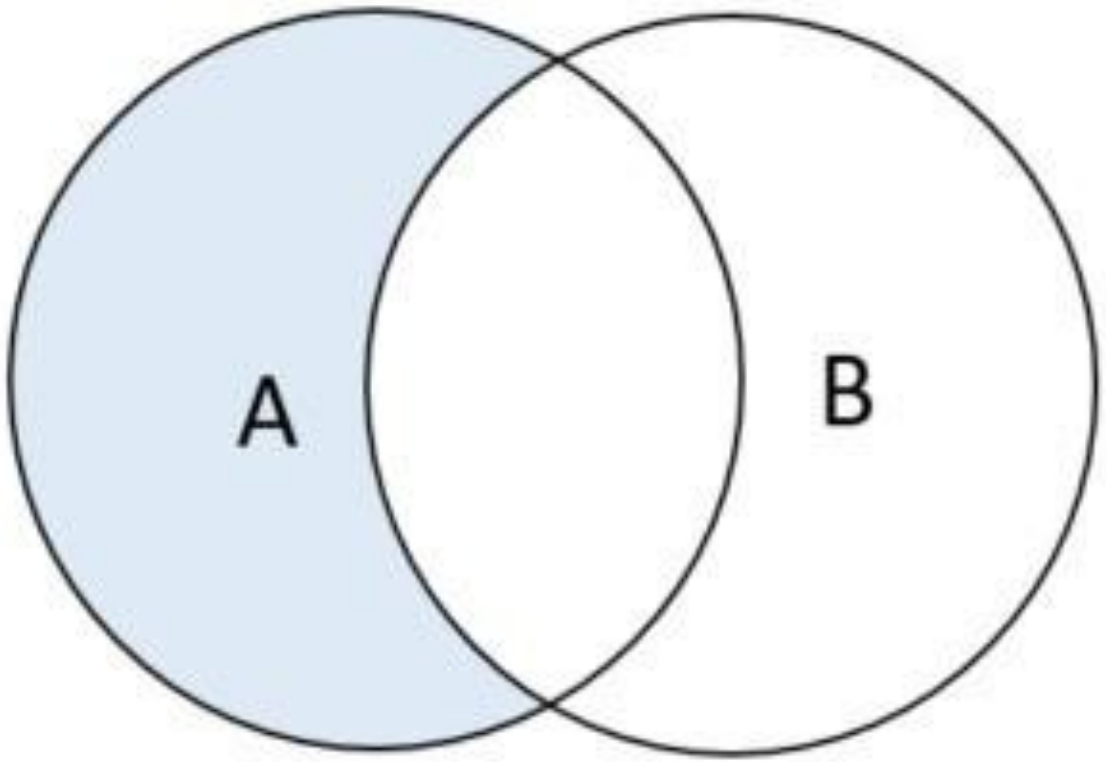
SYNTAX:

```
SELECT select_list
FROM table1
INTERSECT
SELECT select_list
FROM table2;
```

EXCEPT OPERATOR

- Returns rows by comparing the result sets of two or more queries.
- Returns rows in first query not present in output of the second query.
- Returns distinct rows from the first (left) query not in output of the second (right) query.
- The number of columns and their order must be the same in both queries.
- The data types of the respective columns must be

THE FOLLOWING VENN DIAGRAM ILLUSTRATES THE EXCEPT OPERATOR:



SYNTAX:

```
SELECT select_list
FROM table1
EXCEPT
SELECT select_list
FROM table2;
```

POSTGRESQL: GROUPING SETS

- A grouping set is a set of columns by which you group by using the GROUP BY clause.

- A grouping set is denoted by a comma-separated list of columns placed inside parentheses:

(column1, column2, ...)

GROUPING SETS

- PostgreSQL provides the GROUPING SETS clause which is the subclause of the GROUP BY clause.

- The GROUPING SETS allows you to define multiple grouping sets in the same query.

SYNTAX:

```
SELECT c1, c2, aggregate_function(c3)
FROM table_name
GROUP BY
GROUPING SETS ( (c1, c2), (c1), (c2), ()
);
```

EXAMPLE

```
select customer_id,staff_id, count(*) num
from payment
group by grouping sets (
                (customer_id, staff_id),
                (customer_id),
                (staff_id),
                ()
    );
```

output

1,800 rows

	customer_id	staff_id	num
1	<null>	<null>	14596
2	448	2	16
3	459	1	22
4	460	1	10
5	236	2	20
6	282	2	13
7	112	1	13
8	499	2	13
9	1	1	15
10	388	2	19

se Changes

- Grouping sets is equivalent to UNION ALL operator.
- They both give the same output.

```
select customer_id, staff_id, count(*) num
from payment
group by customer_id, staff_id

UNION ALL

select customer_id,null, count(*) num
from payment
group by customer_id

UNION ALL

select null,staff_id, count(*) num
from payment
group by staff_id

UNION ALL

select null,null, count(*) num
from payment;
```

GROUPING SETS: CUBE

- Grouping operations are possible with the concept of grouping sets.
- PostgreSQL CUBE is a subclause of the GROUP BY clause.
- The CUBE allows you to generate multiple grouping sets.

CUBE SYNTAX

```
SELECT c1, c2, c3, aggregate
(c4)
FROM table_name
GROUP BY CUBE (c1, c2, c3);
```

```
GROUPING SETS
(  (c1,c2,c3),
(c1,c2),
(c1,c3),
(c2,c3),
(c1),
(c2),
(c3),
() );
```

CUBE EXAMPLE

```
select customer_id, staff_id, count(*) num
from payment
group by cube (customer_id, staff_id)
order by customer_id,staff_id;
```

output
:

Output | Result 6

1,800 rows

	customer_id	staff_id	num
1	1	1	15
2	1	2	15
3	1	<null>	30
4	2	1	14
5	2	2	12
6	2	<null>	26
7	3	1	12

se Changes

Partial cube example:

```
select staff_id, customer_id, count(*) num
from payment
group by staff_id, cube (customer_id)
order by staff_id, customer_id nulls first ;
```

output
:

1,200 rows

	staff_id	customer_id	num
1	1	<null>	7292
2	1	1	15
3	1	2	14
4	1	3	12
5	1	4	12
6	1	5	14
7	1	6	15

se Changes

GROUPING SETS: ROLLUP

- PostgreSQL ROLLUP is a subclause of the GROUP BY clause.
- Different from the CUBE subclause, ROLLUP does not generate all possible grouping sets based on the specified columns. It just makes a subset of those.
- The ROLLUP assumes a hierarchy among the input columns and generates all grouping sets that make sense considering the hierarchy.

CUBE VS ROLLUP

CUBE sets:

```
(c1, c2, c3)
(c1, c2)
(c2, c3)
(c1,c3)
(c1)
(c2)
(c3)
()
```

- However, the ROLLUP(c1,c2,c3) generates only four grouping sets, assuming the hierarchy c1 > c2 > c3 as follows:

ROLLUP sets:

```
(c1, c2, c3)
(c1, c2)
(c1)
()
```

ROLLUP SYNTAX

```
SELECT c1, c2, c3,
aggregate(c4)  FROM table_name
GROUP BY ROLLUP (c1, c2, c3);
```

ROLLUP EXAMPLE

```
select staff_id, customer_id, count(*) num
from payment
group by rollup (staff_id, customer_id)
order by staff_id, customer_id nulls first;
```

output

1,201 rows

	staff_id	customer_id	num
1	1	<null>	7292
2	1	1	15
3	1	2	14
4	1	3	12
5	1	4	12
6	1	5	14
7	1	6	15

e Changes

Assignment

```
SELECT * FROM top_rated_films;
```

	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957

```
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	An American Pickle	2020
2	The Godfather	1972
3	Greyhound	2020

```
SELECT * FROM top_rated_films
UNION
SELECT * FROM most_popular_films;
```

Assignment

```
SELECT * FROM top_rated_films;
```

	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957

```
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	An American Pickle	2020
2	The Godfather	1972
3	Greyhound	2020

```
SELECT * FROM top_rated_films
UNION ALL
SELECT * FROM most_popular_films
ORDER BY title;
```

Assignment

```
SELECT * FROM top_rated_films;
```

	title character varying	release_year smallint
1	The Shawshank Redemption	1994
2	The Godfather	1972
3	12 Angry Men	1957

```
SELECT * FROM most_popular_films;
```

	title character varying	release_year smallint
1	An American Pickle	2020
2	The Godfather	1972
3	Greyhound	2020

```
SELECT *
FROM most_popular_films
INTERSECT
SELECT *
FROM top_rated_films;
```

Assignment

Employee

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Susan	Klassen	susan.klassen@mydb.com
3	3	May	Kaasman	mkaasman2@freewebs.com

Person

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Ardys	Hansberry	ardys.hansberry@myemail.com
3	3	Hayward	Demschke	[null]
4	4	May	Kaasman	may.kaasman@freemail.com

	id integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com

Assignment

Employee

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Susan	Klassen	susan.klassen@mydb.com
3	3	May	Kaasman	mkaasman2@freewebs.com

Person

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Ardys	Hansberry	ardys.hansberry@myemail.com
3	3	Hayward	Demschke	[null]
4	4	May	Kaasman	may.kaasman@freemail.com

```
SELECT first_name, last_name
FROM Employee
INTERSECT
SELECT first_name, last_name
FROM Person
ORDER BY first_name;
```

Assignment

Employee

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Susan	Klassen	susan.klassen@mydb.com
3	3	May	Kaasman	mkaasman2@freewebs.com

Person

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Ardys	Hansberry	ardys.hansberry@myemail.com
3	3	Hayward	Demschke	[null]
4	4	May	Kaasman	may.kaasman@freemail.com

SELECT id, first_name, last_name
FROM Employee
INTERSECT
SELECT first_name, last_name
FROM Person

Assignment

Employee

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Susan	Klassen	susan.klassen@mydb.com
3	3	May	Kaasman	mkaasman2@freewebs.com

Person

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Ardys	Hansberry	ardys.hansberry@myemail.com
3	3	Hayward	Demschke	[null]
4	4	May	Kaasman	may.kaasman@freemail.com

SELECT *FROM Employee
EXCEPT
SELECT * FROM Person**;**

Assignment

Employee

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Susan	Klassen	susan.klassen@mydb.com
3	3	May	Kaasman	mkaasman2@freewebs.com

Person

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	email character varying (100)
1	1	Annie	Smith	annie.smith@myemail.com
2	2	Ardys	Hansberry	ardys.hansberry@myemail.com
3	3	Hayward	Demschke	[null]
4	4	May	Kaasman	may.kaasman@freemail.com

SELECT * FROM Employee
UNION
SELECT * FROM Person**;**

Assignment

Employee

	emp_id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	email character varying (100)	salary integer	dept_id integer
1	1	Annie	Smith	F	annie.smith@myemail.com	20000	1
2	2	Susan	Klassen	F	susan.klassen@myemail.com	47000	1
3	3	May	Kaasman	M	mkaasman2@freemail.com	93000	2
4	4	Charlton	Duran	M	charlton.duran@freemail.com	56000	1
5	5	Ardys	Hansberry	F	[null]	10000	2
6	6	Hayward	Demschke	F	hayward.demschke@foody.com	75000	2
7	7	Tremaine	Wysome	M	tremaine.wysome@xyz.com	27000	2

```
SELECT dept_id, SUM(salary)
FROM employee
GROUP BY dept_id;
```

Assignment

Employee

	emp_id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	email character varying (100)	salary integer	dept_id integer
1	1	Annie	Smith	F	annie.smith@myemail.com	20000	1
2	2	Susan	Klassen	F	susan.klassen@myemail.com	47000	1
3	3	May	Kaasman	M	mkaasman2@freemail.com	93000	2
4	4	Charlton	Duran	M	charlton.duran@freemail.com	56000	1
5	5	Ardys	Hansberry	F	[null]	10000	2
6	6	Hayward	Demschke	F	hayward.demschke@foody.com	75000	2
7	7	Tremaine	Wysome	M	tremaine.wysome@xyz.com	27000	2

```
SELECT dept_id, gender, SUM(salary) FROM
employee
GROUP BY
    GROUPING SETS (
        (dept_id, gender),
        (dept_id),
        (gender),
        ()
    );
```

Assignment

Employee

	emp_id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	email character varying (100)	salary integer	dept_id integer
1	1	Annie	Smith	F	annie.smith@myemail.com	20000	1
2	2	Susan	Klassen	F	susan.klassen@myemail.com	47000	1
3	3	May	Kaasman	M	mkaasman2@freemail.com	93000	2
4	4	Charlton	Duran	M	charlton.duran@freemail.com	56000	1
5	5	Ardys	Hansberry	F	[null]	10000	2
6	6	Hayward	Demschke	F	hayward.demschke@foody.com	75000	2
7	7	Tremaine	Wysome	M	tremaine.wysome@xyz.com	27000	2

```
SELECT dept_id, gender,
SUM(salary)
FROM employee
GROUP BY
    CUBE(dept_id, gender);
```

Assignment

Employee

	emp_id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	email character varying (100)	salary integer	dept_id integer
1	1	Annie	Smith	F	annie.smith@myemail.com	20000	1
2	2	Susan	Klassen	F	susan.klassen@myemail.com	47000	1
3	3	May	Kaasman	M	mkaasman2@freemail.com	93000	2
4	4	Charlton	Duran	M	charlton.duran@freemail.com	56000	1
5	5	Ardys	Hansberry	F	[null]	10000	2
6	6	Hayward	Demschke	F	hayward.demschke@foody.com	75000	2
7	7	Tremaine	Wysome	M	tremaine.wysome@xyz.com	27000	2

```
SELECT dept_id, gender,
SUM(salary)
FROM employee
GROUP BY
    dept_id,
    CUBE(gender);
```

Assignment

Employee

	emp_id [PK] integer	first_name character varying (50)	last_name character varying (50)	gender character (1)	email character varying (100)	salary integer	dept_id integer
1	1	Annie	Smith	F	annie.smith@myemail.com	20000	1
2	2	Susan	Klassen	F	susan.klassen@myemail.com	47000	1
3	3	May	Kaasman	M	mkaasman2@freemail.com	93000	2
4	4	Charlton	Duran	M	charlton.duran@freemail.com	56000	1
5	5	Ardys	Hansberry	F	[null]	10000	2
6	6	Hayward	Demschke	F	hayward.demschke@foody.com	75000	2
7	7	Tremaine	Wysome	M	tremaine.wysome@xyz.com	27000	2

```
SELECT gender, dept_id,
SUM(salary)
FROM employee
GROUP BY
    ROLLUP(gender,dept_id)
ORDER BY gender, dept_id;
```

References

- https://www.tutorialsteacher.com/postgresql/rollup

  https://www.postgresqltutorial.com/postgresql-tutorial/postgresql-intersect/