# DATABASES

## Lecture 5. Queries

# Querying data

- SELECT retrieves rows from zero or more tables.
- You must have SELECT privilege on each column used in a SELECT command

```
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    [ * | expression [ [ AS ] output_name ] [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY grouping_element [, ...] ]
    [ HAVING condition [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
```

# SELECT List

- The SELECT list specifies expressions that form the output rows of the SELECT statement.
- The expressions can (and usually do) refer to columns computed in the FROM clause.
- Just as in a table, every output column of a SELECT has a name.
- To specify the name to use for an output column, write AS output_name after the column's expression. If you do not specify a column name, a name is chosen automatically by PostgreSQL. If the column's expression is a simple column reference, then the chosen name is the same as that column's name.
- In more complex cases a function or type name may be used, or the system may fall back on a generated name such as ?column?

```
SELECT lower(first_name) FROM actor;

SELECT first_name || ' ' || last_name FROM actor;

SELECT first_name || ' ' || last_name AS "Full Name" FROM actor;
```

# SELECT List

```
SELECT lower('HELLO'), upper('hello')
```

| | lower text | upper text |
|---|---|---|
| 1 | hello | HELLO |

```
SELECT 1+3, 3*4
```

| | ?column? integer | ?column? integer |
|---|---|---|
| 1 | 4 | 12 |

# ALL vs DISTINCT

```
SELECT ALL * FROM cd.facilities
```

- If SELECT DISTINCT is specified, all duplicate rows are removed from the result set
- One row is kept from each group of duplicates
- SELECT ALL specifies the opposite: all rows are kept; that is the default.

| | facid [PK] integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 2 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 3 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 4 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 5 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 6 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 7 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 8 | 7 | Snooker Table | 0 | 5 | 450 | 15 |
| 9 | 8 | Pool Table | 0 | 5 | 400 | 15 |

# ALL vs DISTINCT

```
SELECT DISTINCT * FROM cd.facilities
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 2 | 8 | Pool Table | 0 | 5 | 400 | 15 |
| 3 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 4 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 5 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 6 | 7 | Snooker Table | 0 | 5 | 450 | 15 |
| 7 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 8 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 9 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |

# ALL vs DISTINCT

```
SELECT DISTINCT ON(membercost) * FROM cd.facilities
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 7 | Snooker Table | 0 | 5 | 450 | 15 |
| 2 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 3 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 4 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |

```
SELECT DISTINCT ON (membercost) *
FROM cd.facilities
ORDER BY membercost, initialoutlay DESC;
```

# DISTINCT vs DISTINCT ON

| Query | What it does | Example Output |
|---|---|---|
| SELECT DISTINCT (membercost) FROM cd.facilities; | Returns **only the unique values of membercost**, no other columns. | **membercost**<br>0<br>3.5<br>5<br>35 |
| SELECT DISTINCT ON (membercost) * FROM cd.facilities; | Returns **all columns** but only one row for each unique membercost.<br>By default, the first row Postgres finds is used — you can make it deterministic with ORDER BY. | **facid – name – membercost …**<br>7 – Snooker Table – 0 – …<br>6 – Squash Court – 3.5 –…<br>0 – Tennis Court 1 – 5 – …<br>5 – Massage Room 2 – 35 – … |

# WHERE clause

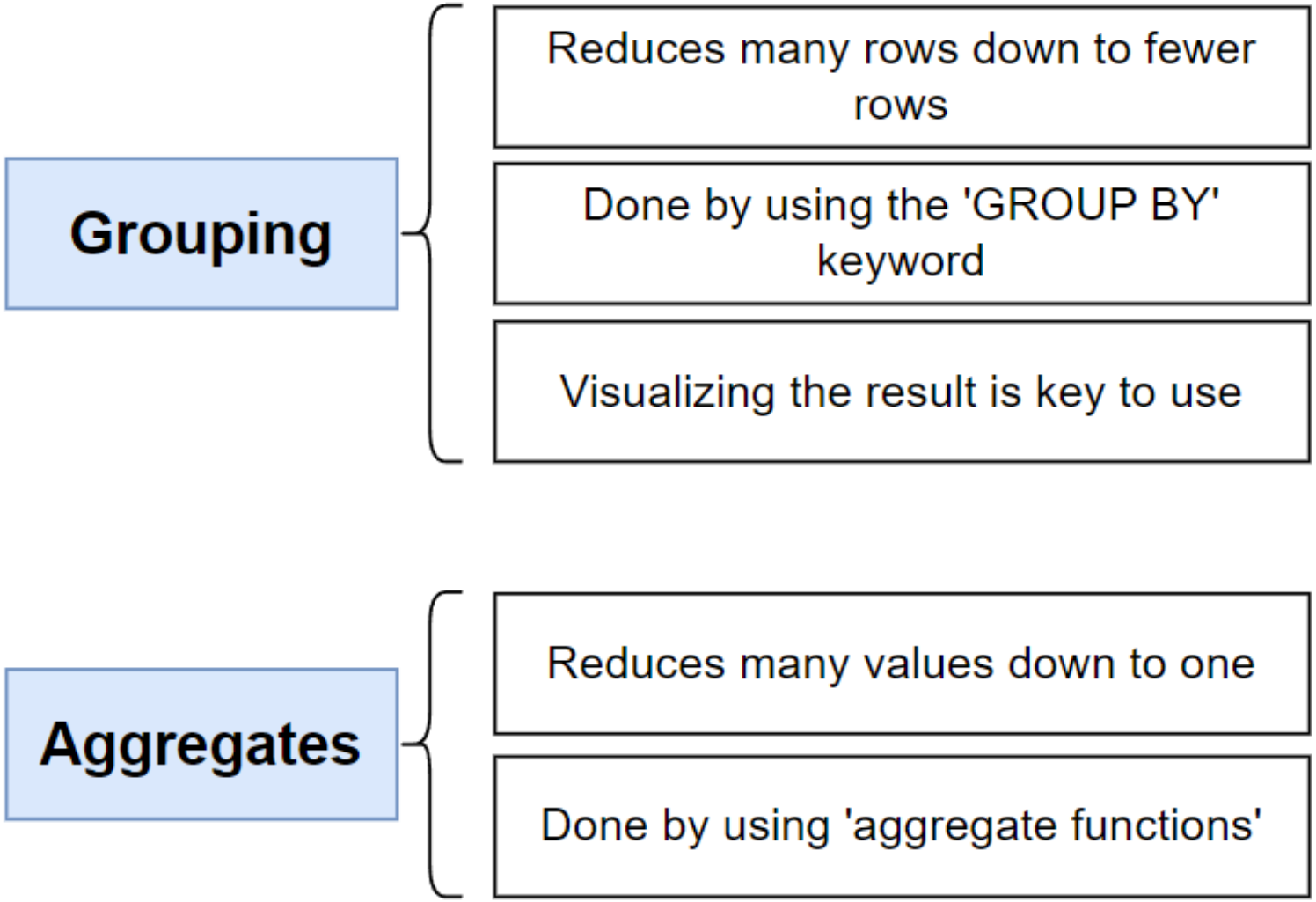
- The optional WHERE clause has the general form

```
WHERE condition
```

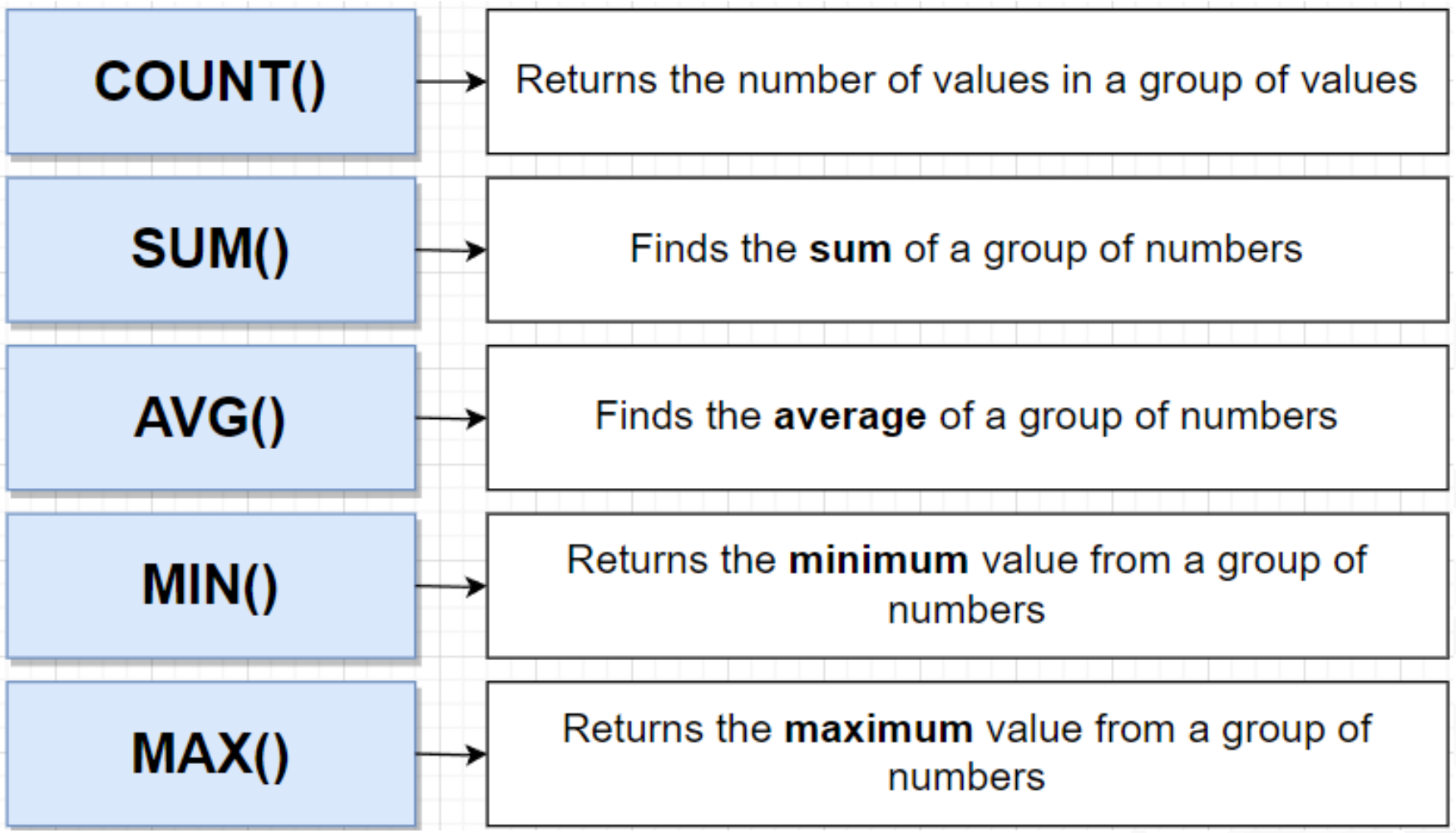
- Where condition is any expression that evaluates to a result of type boolean.
- Any row that does not satisfy this condition will be eliminated from the output.

```
SELECT * FROM cd.facilities WHERE membercost > 5
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 2 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |


Grouping
Reduces many rows down to fewer rows
Done by using the 'GROUP BY' keyword
Visualizing the result is key to use
Aggregates
Reduces many values down to one
Done by using 'aggregate functions'

# AGGREGATE FUNCTIONS



# Key concepts of GROUP BY

```
SELECT column_name1, aggregate_function(column_name2)
FROM table_name
WHERE condition
GROUP BY column_name1;
```

- GROUP BY will condense into a single row all selected rows that share the same values for the grouped expressions.
- An expression used inside a grouping_element can be an input column name, or the name or ordinal number of an output column, or an arbitrary expression formed from input-column values.

```
GROUP BY date_trunc('month', created_at)
```

# GROUP BY

Database

photos

| id | url | user_id |
|---|---|---|
| 1 | https://santina.net | 3 |
| 2 | https://alayna.net | 5 |
| 3 | https://kailyn.name | 3 |
| 20 | http://sasha.com | 3 |

users

| id | username |
|---|---|
| 1 | Reyna.Marvin |
| 2 | Micah.Cremin |
| 3 | Alfredo66 |
| 4 | Gerard_Mitchell42 |
| 5 | Frederique_Donnelly |

comments

| id | contents | user_id | photo_id |
|---|---|---|---|
| 1 | Quo velit iusto ducimus | 2 | 4 |
| 2 | Non est totam | 5 | 5 |
| 100 | Minima dolorem reiciendis | 3 | 3 |

Database

comments

| id | contents | user_id | photo_id |
|---|---|---|---|
| 1 | Non est totam | 1 | 5 |
| 2 | Sed cumque in et. | 3 | 5 |
| 3 | Et sit occaecati. | 5 | 5 |
| 4 | Enim esse magni. | 3 | 5 |
| 5 | Quo velit iusto ducimus | 2 | 4 |
| 6 | Voluptas ab eius. | 2 | 5 |
| 7 | Non est totam | 1 | 5 |

```
SELECT user_id
FROM comments
GROUP BY user_id;
```

# GROUP BY

Database

comments

| id | contents | user_id | photo_id |
|---|---|---|---|
| 1 | Non est totam | 1 | 5 |
| 2 | Sed cumque in et. | 3 | 5 |
| 3 | Et sit occaecati. | 5 | 5 |
| 4 | Enim esse magni. | 3 | 5 |
| 5 | Quo velit iusto ducimus | 2 | 4 |
| 6 | Voluptas ab eius. | 2 | 5 |
| 7 | Non est totam | 1 | 5 |

**GROUP BY user_id**

↓

Find the set of all unique user_id's

↓

Take each row and assign it to a group based on its user_id

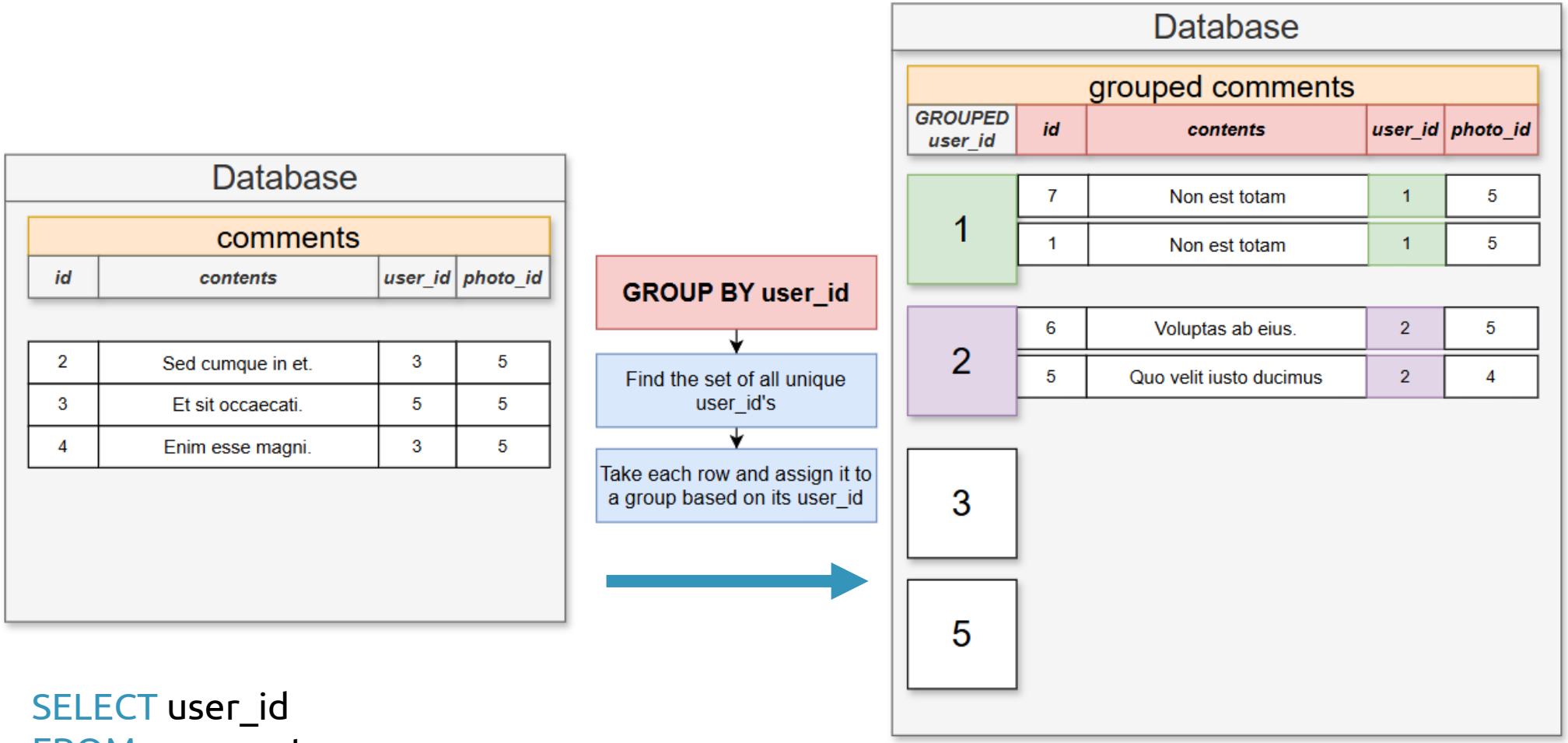
Database

grouped comments

| GROUPED user_id | id | contents | user_id | photo_id |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| 3 | | | | |
| 5 | | | | |

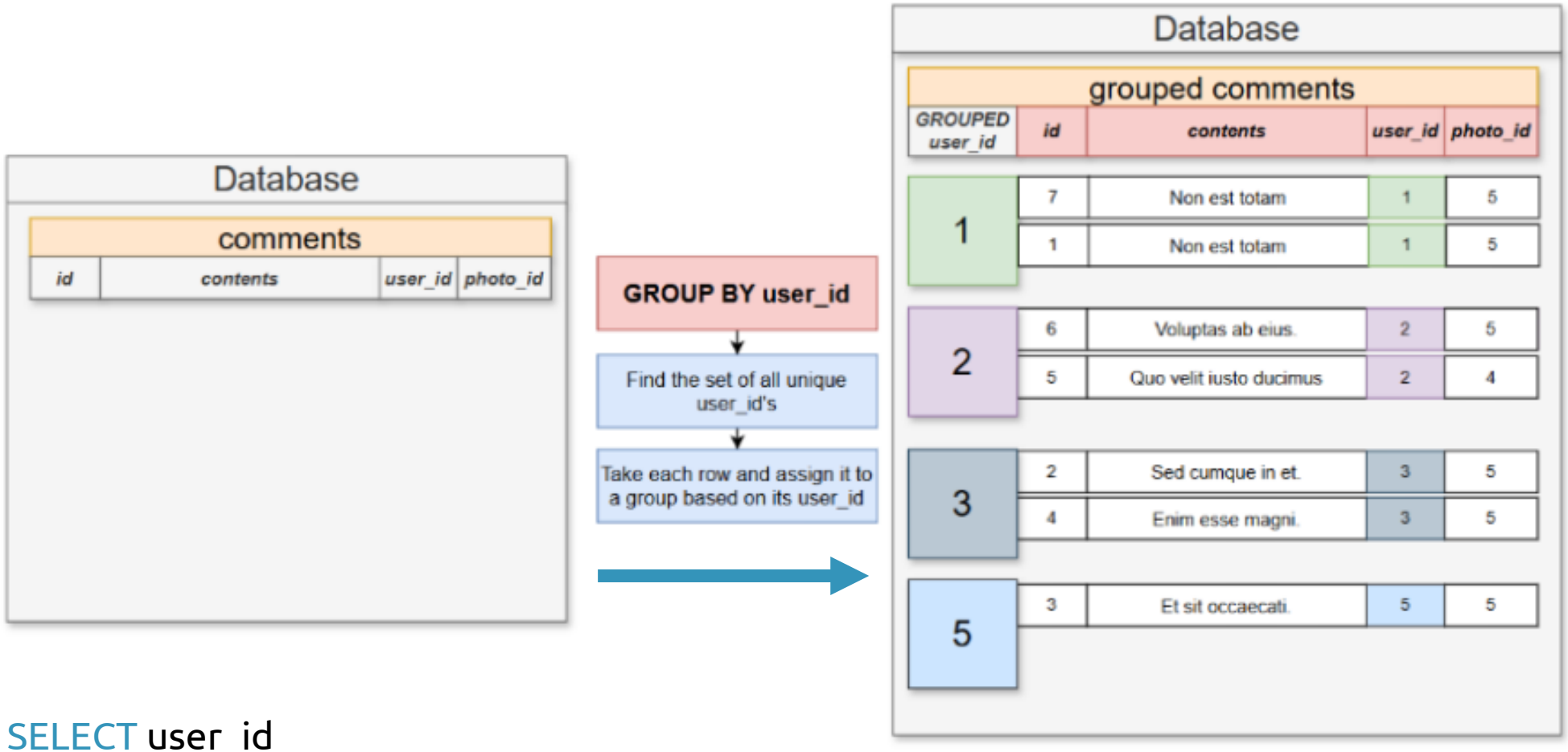
```
SELECT user_id
FROM comments
GROUP BY user_id;
```

# GROUP BY




```
SELECT user_id
FROM comments
GROUP BY user_id;
```

# GROUP BY




```
SELECT user_id
FROM comments
GROUP BY user_id;
```

# GROUP BY

Database

grouped comments

| GROUP photo_id | id | contents | user_id | photo_id | COUNT(id) |
|---|---|---|---|---|---|
| 1 | 1 | Non est totam | 1 | 5 | 2 |
| | 7 | Non est totam | 1 | 5 | |
| 2 | 5 | Quo velit iusto ducimus | 2 | 4 | 2 |
| | 6 | Voluptas ab eius. | 2 | 5 | |
| 3 | 2 | Sed cumque in et. | 3 | 5 | 2 |
| | 4 | Enim esse magni. | 3 | 5 | |
| 5 | 3 | Et sit occaecati. | 5 | 5 | 1 |

```
SELECT user_id, COUNT(*)
AS num_comments
FROM comments
GROUP BY user_id;
```

# GROUP BY

```
SELECT membercost FROM cd.facilities GROUP BY membercost
```

| | facid [PK] integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 2 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 3 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 4 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 5 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 6 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 7 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 8 | 7 | Snooker Table | 0 | 5 | 450 | 15 |
| 9 | 8 | Pool Table | 0 | 5 | 400 | 15 |
| | | | | | | |

| | membercost numeric |
|---|---|
| 1 | 3.5 |
| 2 | 35 |
| 3 | 5 |
| 4 | 0 |

# GROUP BY

```
SELECT membercost, sum(guestcost) AS guest_sum
    FROM cd.facilities
    GROUP BY membercost
```

| | facid [PK] integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 2 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 3 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 4 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 5 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 6 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 7 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 8 | 7 | Snooker Table | 0 | 5 | 450 | 15 |
| 9 | 8 | Pool Table | 0 | 5 | 400 | 15 |
| | | | | | | |

| | membercost numeric | guest_sum numeric |
|---|---|---|
| 1 | 3.5 | 17.5 |
| 2 | 35 | 160 |
| 3 | 5 | 50 |
| 4 | 0 | 30.5 |

# HAVING

- The optional HAVING clause has the general form

**HAVING** *condition*

where *condition* is the same as specified for the WHERE clause.

- HAVING eliminates group rows that do not satisfy the condition.

## WHERE VS. HAVING

- WHERE filters the query BEFORE grouping
- WHERE is used in conjunction with SELECT, INSERT, UPDATE
- Aggregation functions cannot be used in a WHERE clause, except when a subquery is used

- HAVING filters the query AFTER grouping
- HAVING is only used in conjunction with SELECT
- Aggregation functions can be used in the HAVING clause

# HAVING

```
SELECT membercost, sum(guestcost) AS guest_sum
    FROM cd.facilities
    GROUP BY membercost
    HAVING membercost > 0
```

| | membercost numeric | guest_sum numeric |
|---|---|---|
| 1 | 3.5 | 17.5 |
| 2 | 35 | 160 |
| 3 | 5 | 50 |
| 4 | 0 | 30.5 |

| | membercost numeric | guest_sum numeric |
|---|---|---|
| 1 | 3.5 | 17.5 |
| 2 | 35 | 160 |
| 3 | 5 | 50 |

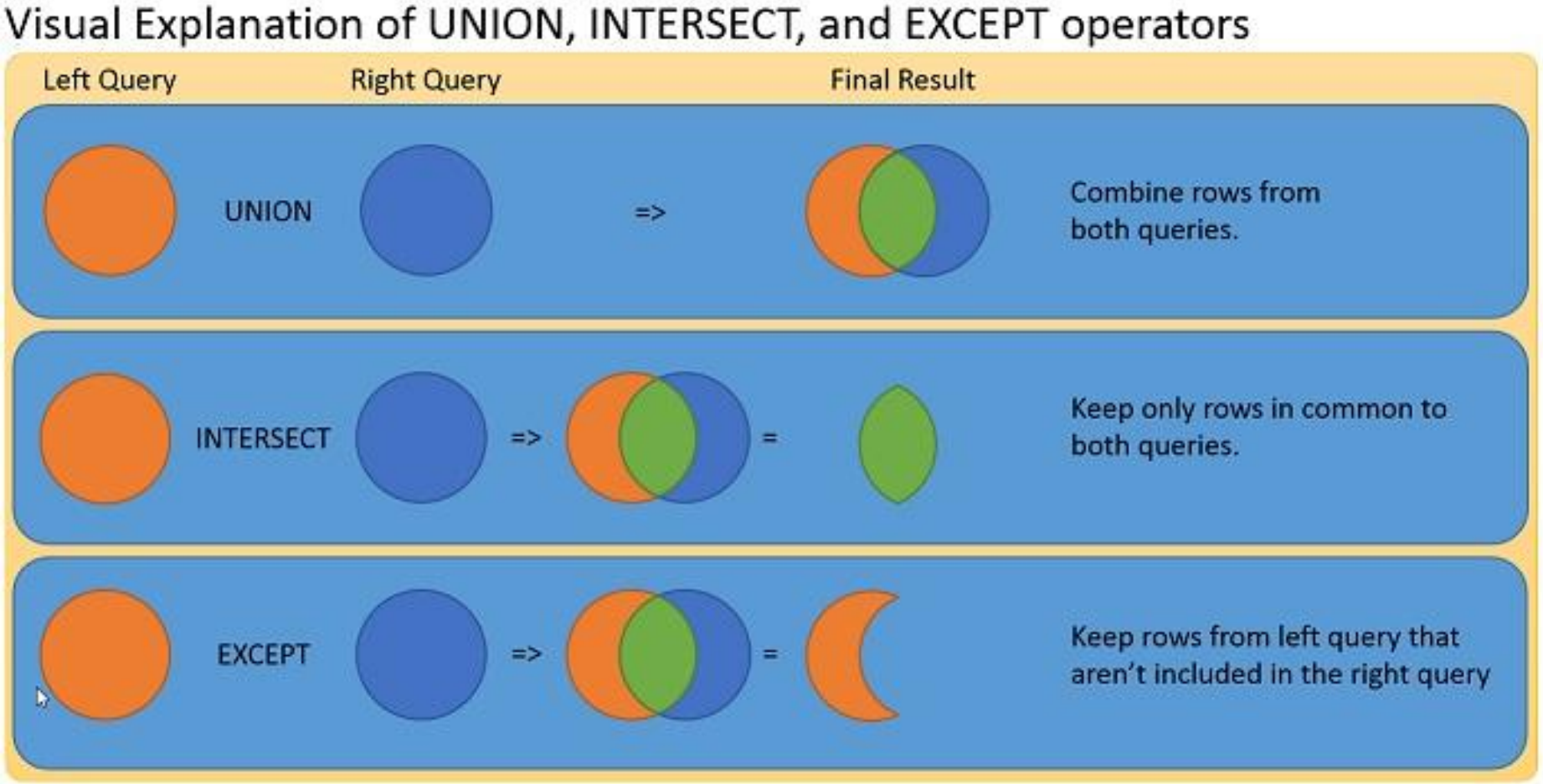
# HAVING

```
SELECT membercost, sum(guestcost)
    FROM cd.facilities
    GROUP BY membercost
    HAVING sum(initialoutlay) > 5000
```

| | membercost numeric | guest_sum numeric |
|---|---|---|
| 1 | 3.5 | 17.5 |
| 2 | 35 | 160 |
| 3 | 5 | 50 |
| 4 | 0 | 30.5 |

| | membercost numeric | guest_sum numeric |
|---|---|---|
| 1 | 35 | 160 |
| 2 | 5 | 50 |
| 3 | 0 | 30.5 |

# UNION, EXCEPT, INTERSECT



# UNION

- The UNION clause has this general form:

```
select_statement UNION [ ALL | DISTINCT ] select_statement
```

- select_statement is any SELECT statement without an ORDER BY, LIMIT clause
- The UNION operator computes the set union of the rows returned by the involved SELECT statements.
- A row is in the set union of two result sets if it appears in at least one of the result sets.
- The two SELECT statements that represent the direct operands of the UNION must produce the same number of columns
- Corresponding columns must be of compatible data types.

# UNION

```
SELECT * FROM cd.facilities WHERE guestcost >25
UNION SELECT * FROM cd.facilities WHERE membercost > 0
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 2 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 3 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 4 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 5 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |

# UNION

```
SELECT * FROM cd.facilities
        UNION ALL SELECT * FROM cd.facilities
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 2 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 3 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 4 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 5 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 6 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 7 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 8 | 7 | Snooker Table | 0 | 5 | 450 | 15 |
| 9 | 8 | Pool Table | 0 | 5 | 400 | 15 |
| 10 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 11 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 12 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 13 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 14 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 15 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 16 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 17 | 7 | Snooker Table | 0 | 5 | 450 | 15 |
| 18 | 8 | Pool Table | 0 | 5 | 400 | 15 |

# INTERSECT

- The INTERSECT clause has this general form:

```
select_statement INTERSECT [ ALL | DISTINCT ] select_statement
```

- select_statement is any SELECT statement without an ORDER BY, LIMIT clause
- The INTERSECT operator computes the set intersection of the rows returned by the involved SELECT statements.
- A row is in the intersection of two result sets if it appears in both result sets.
- The result of INTERSECT does not contain any duplicate rows unless the ALL option is specified.
- With ALL, a row that has m duplicates in the left table and n duplicates in the right table will appear min(m,n) times in the result set.
- INTERSECT binds more tightly than UNION.
- A UNION B INTERSECT C will be read as A UNION (B INTERSECT C)

# INTERSECT

```
SELECT * FROM cd.facilities WHERE guestcost >25
    INTERSECT SELECT * FROM cd.facilities WHERE membercost > 0
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 2 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |

# EXCEPT

- The EXCEPT clause has this general form:

```
select_statement EXCEPT [ ALL | DISTINCT ] select_statement
```

- select_statement is any SELECT statement without an ORDER BY, LIMIT clause
- The EXCEPT operator computes the set of rows that are in the result of the left SELECT statement but not in the result of the right one.
- The result of EXCEPT does not contain any duplicate rows unless the ALL option is specified.
- With ALL, a row that has m duplicates in the left table and n duplicates in the right table will appear max(m-n,0) times in the result set.
- Multiple EXCEPT operators in the same SELECT statement are evaluated left to right, unless parentheses dictate otherwise.
- EXCEPT binds at the same level as UNION.

# EXCEPT

```
SELECT * FROM cd.facilities WHERE guestcost > 10
    EXCEPT SELECT * FROM cd.facilities WHERE membercost > 5
```

| facid | name | membercost | guestcost | initialoutlay | monthlymaintenance |
|---|---|---|---|---|---|
| 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |

| facid | name | membercost | guestcost | initialoutlay | monthlymaintenance |
|---|---|---|---|---|---|
| 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 2 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 3 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 4 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |

# EXCEPT

```
SELECT * FROM cd.facilities
    EXCEPT SELECT * FROM cd.facilities
```

| | facid<br>integer | name<br>character varying (100) | membercost<br>numeric | guestcost<br>numeric | initialoutlay<br>numeric | monthlymaintenance<br>numeric |
|---|---|---|---|---|---|---|

# ORDER BY

- The optional ORDER BY clause has the general form

```
ORDER BY expression [ ASC | DESC | USING operator ]
    [ NULLS { FIRST | LAST } ] [, ...]
```

- The ORDER BY clause causes the result rows to be sorted according to the specified expression(s)
- If two rows are equal according to the leftmost expression, they are compared according to the next expression and so on.
- If they are equal according to all specified expressions, they are returned in an implementation-dependent order.
- Each *expression* can be the name or ordinal number of an output column (SELECT list item) or it can be an arbitrary expression formed from input column values.

# ORDER BY

- Optionally one can add the key word ASC (ascending) or DESC (descending)
- If not specified, ASC is assumed by default.
- Each expression can be the name or ordinal number of an output column (SELECT list item)
- If NULLS LAST is specified, null values sort after all non-null values
- If NULLS FIRST is specified, null values sort before all non-null values.

# ORDER BY

```
SELECT memid, recommendedby FROM cd.members WHERE memid > 20
```

| | memid integer | recommendedby integer |
|---|---|---|
| 1 | 21 | 1 |
| 2 | 22 | 16 |
| 3 | 24 | 15 |
| 4 | 26 | 11 |
| 5 | 27 | 20 |
| 6 | 28 | [null] |
| 7 | 29 | 2 |
| 8 | 30 | 2 |
| 9 | 33 | [null] |
| 10 | 35 | 30 |
| 11 | 36 | 2 |
| 12 | 37 | [null] |

# ORDER BY

```
SELECT memid, recommendedby FROM cd.members
    WHERE memid > 20
    ORDER BY recommendedby
```

| | memid integer | recommendedby integer |
|---|---|---|
| 1 | 21 | 1 |
| 2 | 29 | 2 |
| 3 | 30 | 2 |
| 4 | 36 | 2 |
| 5 | 26 | 11 |
| 6 | 24 | 15 |
| 7 | 22 | 16 |
| 8 | 27 | 20 |
| 9 | 35 | 30 |
| 10 | 33 | [null] |
| 11 | 37 | [null] |
| 12 | 28 | [null] |

```
SELECT memid, recommendedby FROM cd.members
    WHERE memid > 20
    ORDER BY recommendedby ASC
```

| | memid integer | recommendedby integer |
|---|---|---|
| 1 | 21 | 1 |
| 2 | 29 | 2 |
| 3 | 30 | 2 |
| 4 | 36 | 2 |
| 5 | 26 | 11 |
| 6 | 24 | 15 |
| 7 | 22 | 16 |
| 8 | 27 | 20 |
| 9 | 35 | 30 |
| 10 | 33 | [null] |
| 11 | 37 | [null] |
| 12 | 28 | [null] |

# ORDER BY

```
SELECT memid, recommendedby FROM cd.members
    WHERE memid > 20
    ORDER BY recommendedby DESC
```

| | memid integer | recommendedby integer |
|---|---|---|
| 1 | 33 | [null] |
| 2 | 37 | [null] |
| 3 | 28 | [null] |
| 4 | 35 | 30 |
| 5 | 27 | 20 |
| 6 | 22 | 16 |
| 7 | 24 | 15 |
| 8 | 26 | 11 |
| 9 | 29 | 2 |
| 10 | 30 | 2 |
| 11 | 36 | 2 |
| 12 | 21 | 1 |

```
SELECT memid, recommendedby FROM cd.members
    WHERE memid > 20
    ORDER BY recommendedby NULLS FIRST
```

| | memid integer | recommendedby integer |
|---|---|---|
| 1 | 28 | [null] |
| 2 | 37 | [null] |
| 3 | 33 | [null] |
| 4 | 21 | 1 |
| 5 | 36 | 2 |
| 6 | 29 | 2 |
| 7 | 30 | 2 |
| 8 | 26 | 11 |
| 9 | 24 | 15 |
| 10 | 22 | 16 |
| 11 | 27 | 20 |
| 12 | 35 | 30 |

# ORDER BY

```
SELECT memid, recommendedby FROM cd.members
    WHERE memid > 20
    ORDER BY recommendedby DESC NULLS LAST
```

| | memid integer | recommendedby integer |
|---|---|---|
| 1 | 35 | 30 |
| 2 | 27 | 20 |
| 3 | 22 | 16 |
| 4 | 24 | 15 |
| 5 | 26 | 11 |
| 6 | 36 | 2 |
| 7 | 29 | 2 |
| 8 | 30 | 2 |
| 9 | 21 | 1 |
| 10 | 33 | [null] |
| 11 | 28 | [null] |
| 12 | 37 | [null] |

```
SELECT memid, recommendedby FROM cd.members
    WHERE memid > 20
    ORDER BY recommendedby ASC, memid DESC
```

| | memid integer | recommendedby integer |
|---|---|---|
| 1 | 21 | 1 |
| 2 | 36 | 2 |
| 3 | 30 | 2 |
| 4 | 29 | 2 |
| 5 | 26 | 11 |
| 6 | 24 | 15 |
| 7 | 22 | 16 |
| 8 | 27 | 20 |
| 9 | 35 | 30 |
| 10 | 37 | [null] |
| 11 | 33 | [null] |
| 12 | 28 | [null] |

# LIMIT

- The LIMIT clause consists of two independent subclauses:

```
LIMIT { count | ALL }
OFFSET start
```

- *count* specifies the maximum number of rows to return,
- while *start* specifies the number of rows to skip before starting to return rows.
- When both are specified, start rows are skipped before starting to count the count rows to be returned.
- If the count expression evaluates to NULL, it is treated as LIMIT ALL, i.e., no limit.
- If start evaluates to NULL, it is treated the same as OFFSET 0.

# LIMIT

```
SELECT * FROM cd.facilities LIMIT 5
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 0 | Tennis Court 1 | 5 | 25 | 10000 | 200 |
| 2 | 1 | Tennis Court 2 | 5 | 25 | 8000 | 200 |
| 3 | 2 | Badminton Court | 0 | 15.5 | 4000 | 50 |
| 4 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 5 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |

# LIMIT

```
SELECT * FROM cd.facilities
    LIMIT 5 OFFSET 3
```

| | facid integer | name character varying (100) | membercost numeric | guestcost numeric | initialoutlay numeric | monthlymaintenance numeric |
|---|---|---|---|---|---|---|
| 1 | 3 | Table Tennis | 0 | 5 | 320 | 10 |
| 2 | 4 | Massage Room 1 | 35 | 80 | 4000 | 3000 |
| 3 | 5 | Massage Room 2 | 35 | 80 | 4000 | 3000 |
| 4 | 6 | Squash Court | 3.5 | 17.5 | 5000 | 80 |
| 5 | 7 | Snooker Table | 0 | 5 | 450 | 15 |