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 syntax

```
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.

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.

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.

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.

SELECT List

- If you do not specify a column name, a name is chosen automatically by PostgreSQL.

SELECT List

- 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.

SELECT List

- 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 List

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

	lower text	upper text
1	hello	HELLO

SELECT List

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

	?column? integer	?column? integer
1	4	12

ALL vs DISTINCT

- If `SELECT DISTINCT` is specified, all duplicate rows are removed from the result set

ALL vs DISTINCT

- If `SELECT DISTINCT` is specified, all duplicate rows are removed from the result set

- One row is kept from each group of duplicates

ALL vs DISTINCT

- 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.

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

	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

```
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

```
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

WHERE clause

- The optional `WHERE` clause has the general form

```
WHERE condition
```

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.

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

GROUP BY

- The optional `GROUP BY` clause has the general form

```
GROUP BY grouping_element [, ...]
```

GROUP BY

- The optional `GROUP BY` clause has the general form

```
GROUP BY grouping_element [, ...]
```

- GROUP BY will condense into a single row all selected rows that share the same values for the grouped expressions.

GROUP BY

- The optional `GROUP BY` clause has the general form

```
GROUP BY grouping_element [, ...]
```

- 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.

```
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

```
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
```

HAVING

- The optional HAVING clause has the general form

```
HAVING condition
```

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

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.

HAVING

- `HAVING` is different from `WHERE`:

- `WHERE` filters individual rows before the application of `GROUP BY`

HAVING

- `HAVING` is different from `WHERE`:

- `WHERE` filters individual rows before the application of `GROUP BY`

- `HAVING` filters group rows created by `GROUP BY`

HAVING

- `HAVING` is different from `WHERE`:

- `WHERE` filters individual rows before the application of `GROUP BY`

- `HAVING` filters group rows created by `GROUP BY`

- Each column referenced in *`condition`* must unambiguously reference a grouping column, unless the reference appears within an aggregate function

```
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

```
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

- The `UNION` clause has this general form:

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

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

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.

UNION

- A row is in the set union of two result sets if it appears in at least one of the result sets.

UNION

- 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

UNION

- 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.

```
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

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

```
SELECT * FROM cd.facilities WHERE guestcost >25
    UNION ALL 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	5	Massage Room 2	35	80	4000	3000
3	0	Tennis Court 1	5	25	10000	200
4	1	Tennis Court 2	5	25	8000	200
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

```
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
```

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

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.

INTERSECT

- A row is in the intersection of two result sets if it appears in both result sets.

INTERSECT

- 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.

INTERSECT

- 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

- `INTERSECT` binds more tightly than `UNION`.
- A `UNION B INTERSECT C` will be read as `A UNION (B INTERSECT C)`

```
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

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

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

```
SELECT * FROM cd.facilities
    INTERSECT SELECT * 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

EXCEPT

- The `EXCEPT` clause has this general form:

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

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

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.

EXCEPT

- The result of `EXCEPT` does not contain any duplicate rows unless the `ALL` option is specified.

EXCEPT

- 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.

EXCEPT

- 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

- 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.

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

	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

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

	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

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

	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

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

	facid integer	name character varying (100)	membercost numeric	guestcost numeric	initialoutlay numeric	monthlymaintenance numeric

```sql
CREATE TABLE employees (
 employee_id serial PRIMARY KEY,
 employee_name VARCHAR (255) NOT NULL
);

CREATE TABLE keys (
 employee_id INT PRIMARY KEY,
 effective_date DATE NOT NULL,
 FOREIGN KEY (employee_id) REFERENCES employees (employee_id)
);

CREATE TABLE hipos (
 employee_id INT PRIMARY KEY,
 effective_date DATE NOT NULL,
 FOREIGN KEY (employee_id) REFERENCES employees (employee_id)
);
```

```sql
INSERT INTO employees (employee_name)
VALUES
 ('Joyce Edwards'),
 ('Diane Collins'),
 ('Alice Stewart'),
 ('Julie Sanchez'),
 ('Heather Morris'),
 ('Teresa Rogers'),
 ('Doris Reed'),
 ('Gloria Cook'),
 ('Evelyn Morgan'),
 ('Jean Bell');
```

```sql
INSERT INTO keys
VALUES
 (1, '2000-02-01'),
 (2, '2001-06-01'),
 (5, '2002-01-01'),
 (7, '2005-06-01');

INSERT INTO hipos
VALUES
 (9, '2000-01-01'),
 (2, '2002-06-01'),
 (5, '2006-06-01'),
 (10, '2005-06-01');
```

```sql
SELECT employee_id
    FROM keys;
```

	employee_id integer
1	1
2	2
3	5
4	7

```sql
SELECT employee_id
    FROM hipos;
```

	employee_id integer
1	9
2	2
3	5
4	10

```sql
SELECT employee_id FROM keys
    UNION SELECT employee_id FROM hipos;
```

	employee_id integer
1	2
2	10
3	9
4	1
5	7
6	5

```
SELECT employee_id FROM keys
    INTERSECT SELECT employee_id FROM hipos;
```

	employee_id integer
1	2
2	5

```
SELECT employee_id FROM keys
    EXCEPT SELECT employee_id FROM hipos;
```

	employee_id integer
1	7
2	1

ORDER BY

- The optional `ORDER BY` clause has the general form

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

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)

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.

ORDER BY

- If they are equal according to all specified expressions, they are returned in an implementation-dependent order.

ORDER BY

- 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)

ORDER BY

- 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

- 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.

- Optionally one can add the key word `ASC` (ascending) or `DESC` (descending)

ORDER BY

- If not specified, `ASC` is assumed by default.

ORDER BY

- If not specified, `ASC` is assumed by default.
- Alternatively, a specific ordering operator name can be specified in the `USING` clause.Each *`expression`* can be the name or ordinal number of an output column (`SELECT` list item)

ORDER BY

- If not specified, `ASC` is assumed by default.
- Alternatively, a specific ordering operator name can be specified in the `USING` clause.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

ORDER BY

- If not specified, `ASC` is assumed by default.

- Alternatively, a specific ordering operator name can be specified in the `USING` clause.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.

```
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]

```
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]

```
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

```
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]

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

	memid integer	recommendedby integer	sum integer
1	21	1	22
2	29	2	31
3	30	2	32
4	26	11	37
5	22	16	38
6	36	2	38
7	24	15	39
8	27	20	47
9	35	30	65
10	28	[null]	[null]
11	33	[null]	[null]
12	37	[null]	[null]

LIMIT

- The `LIMIT` clause consists of two independent sub-clauses:

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

LIMIT

- The `LIMIT` clause consists of two independent sub-clauses:

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

- *`count`* specifies the maximum number of rows to return,

LIMIT

- The `LIMIT` clause consists of two independent sub-clauses:

```
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.

LIMIT

- When both are specified, *`start`* rows are skipped before starting to count the *`count`* rows to be returned.

LIMIT

- 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.

LIMIT

- 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`.

```
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

```
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

Questions?