

```

SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (14.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# create database test;
CREATE DATABASE
postgres=# \l
      List of databases
  Name   |  Owner   | Encoding | Collate       | Ctype        | Access privileges
-----+-----+-----+-----+-----+-----+
postgres | postgres | UTF8    | English_India.1252 | English_India.1252 |
template0 | postgres | UTF8    | English_India.1252 | English_India.1252 | =c/postgres      +
          |          |          |          |          | postgres=CTc/postgres
template1 | postgres | UTF8    | English_India.1252 | English_India.1252 | =c/postgres      +
          |          |          |          |          | postgres=CTc/postgres
test     | postgres | UTF8    | English_India.1252 | English_India.1252 | =c/postgres      +
          |          |          |          |          | postgres=CTc/postgres
(4 rows)

postgres=# \c test
You are now connected to database "test" as user "postgres".
test=#

```

Using sql shell pressing enter till we reach password and type it.

Using commands in pictures we continue.

\l used to list all databases present.

\c test to select test database out of all available databases.

```

test=# \c postgres
You are now connected to database "postgres" as user "postgres".

postgres=# drop database test;
DROP DATABASE
postgres=# \l
      List of databases
  Name   |  Owner   | Encoding | Collate       | Ctype        | Access privileges
-----+-----+-----+-----+-----+-----+
postgres | postgres | UTF8    | English_India.1252 | English_India.1252 |
template0 | postgres | UTF8    | English_India.1252 | English_India.1252 | =c/postgres      +
          |          |          |          |          | postgres=CTc/postgres
template1 | postgres | UTF8    | English_India.1252 | English_India.1252 | =c/postgres      +
          |          |          |          |          | postgres=CTc/postgres
c(3 rows)

postgres=#

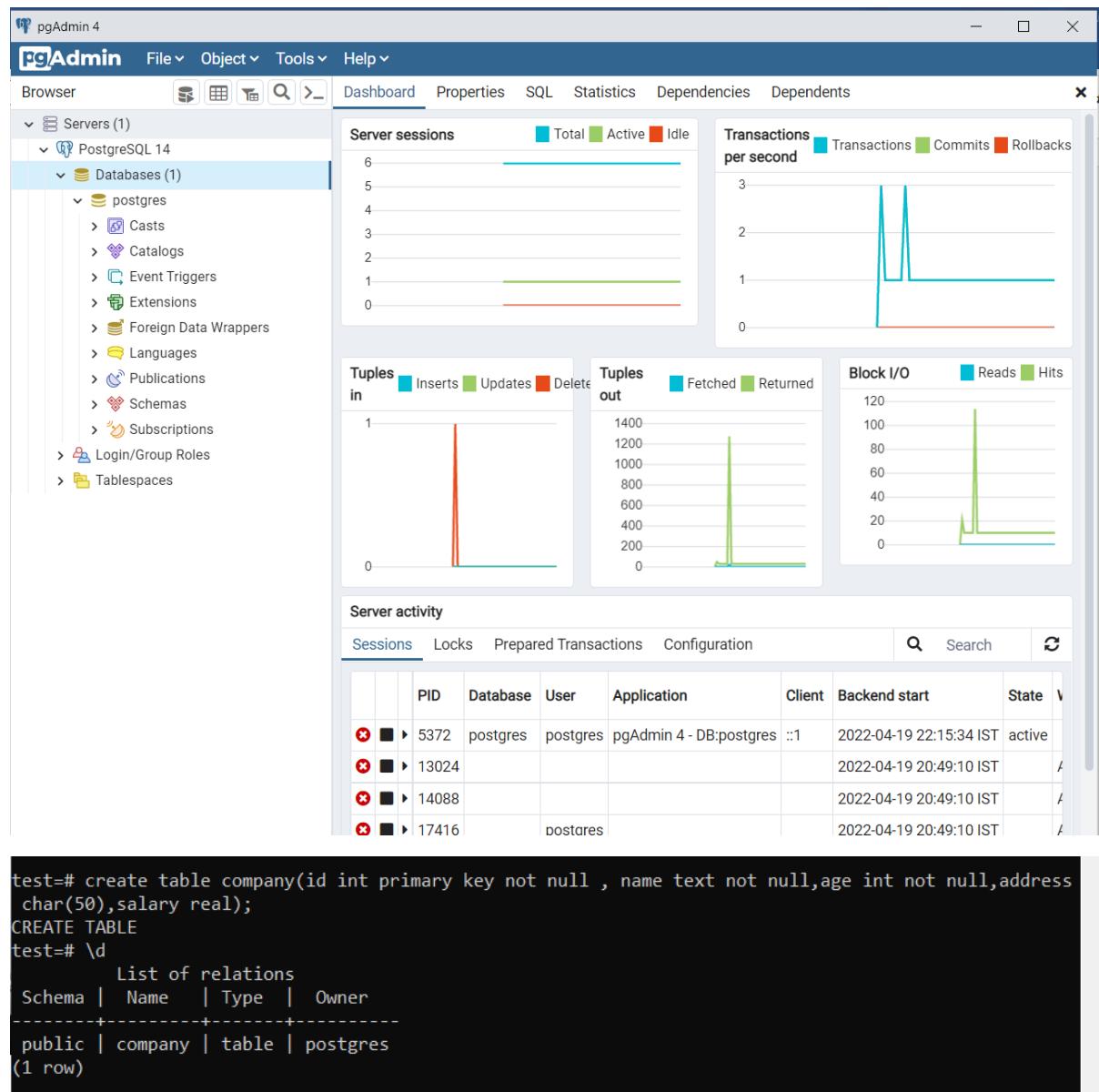
```

drop to remove table.

\q to disconnect all databases.

```
postgres=# \q  
Press any key to continue . . .
```

Using pgAdmin4 also it can be done right click on databases to add / drop new databases.



\d to describe table.

```
SQL Shell (psql)

test=# create table company(id int primary key not null , name text not n
ull,age int not null,address char(50),salary real);
CREATE TABLE
test=# \d
      List of relations
 Schema |   Name    | Type  | Owner
-----+-----+-----+
 public | company | table | postgres
(1 row)

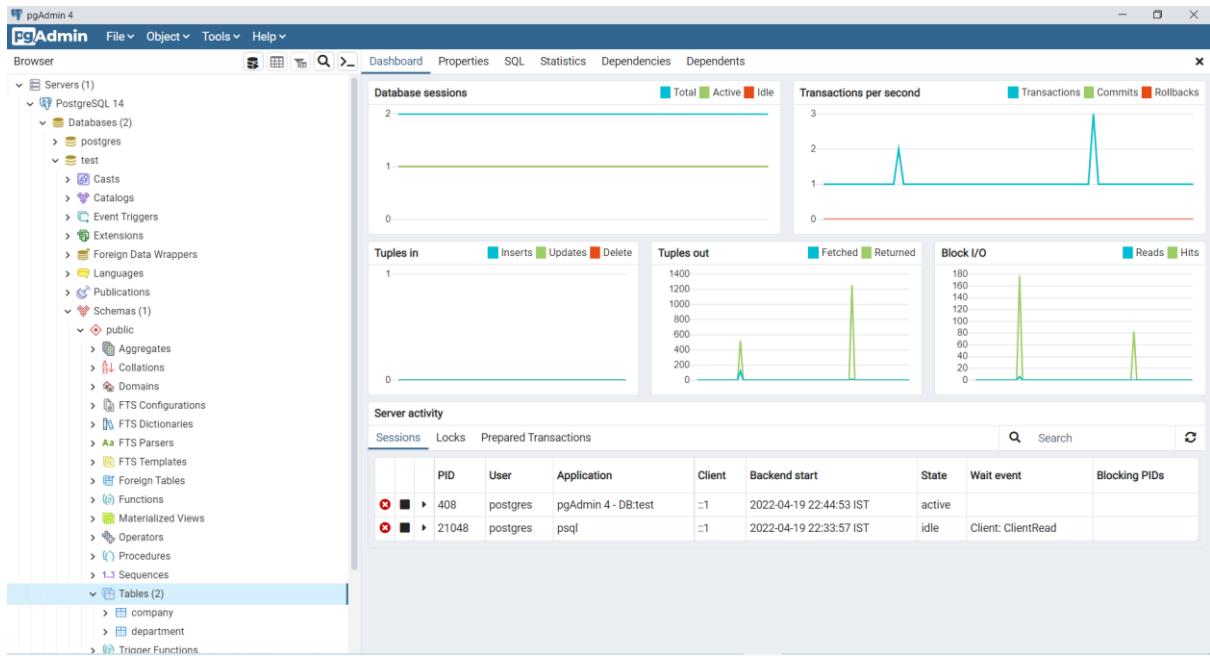
test=# create table department(id int primary key not null,dept char(50)
not null,emp_id int notnull);
ERROR:  syntax error at or near "notnull"
LINE 1: ...ary key not null,dept char(50) not null,emp_id int notnull);
                                ^
test=# create table department(id int primary key not null,dept char(50)
not null,emp_id int not null);
CREATE TABLE
test=# \d
      List of relations
 Schema |     Name    | Type  | Owner
-----+-----+-----+
 public | company    | table | postgres
 public | department | table | postgres
(2 rows)

test=#

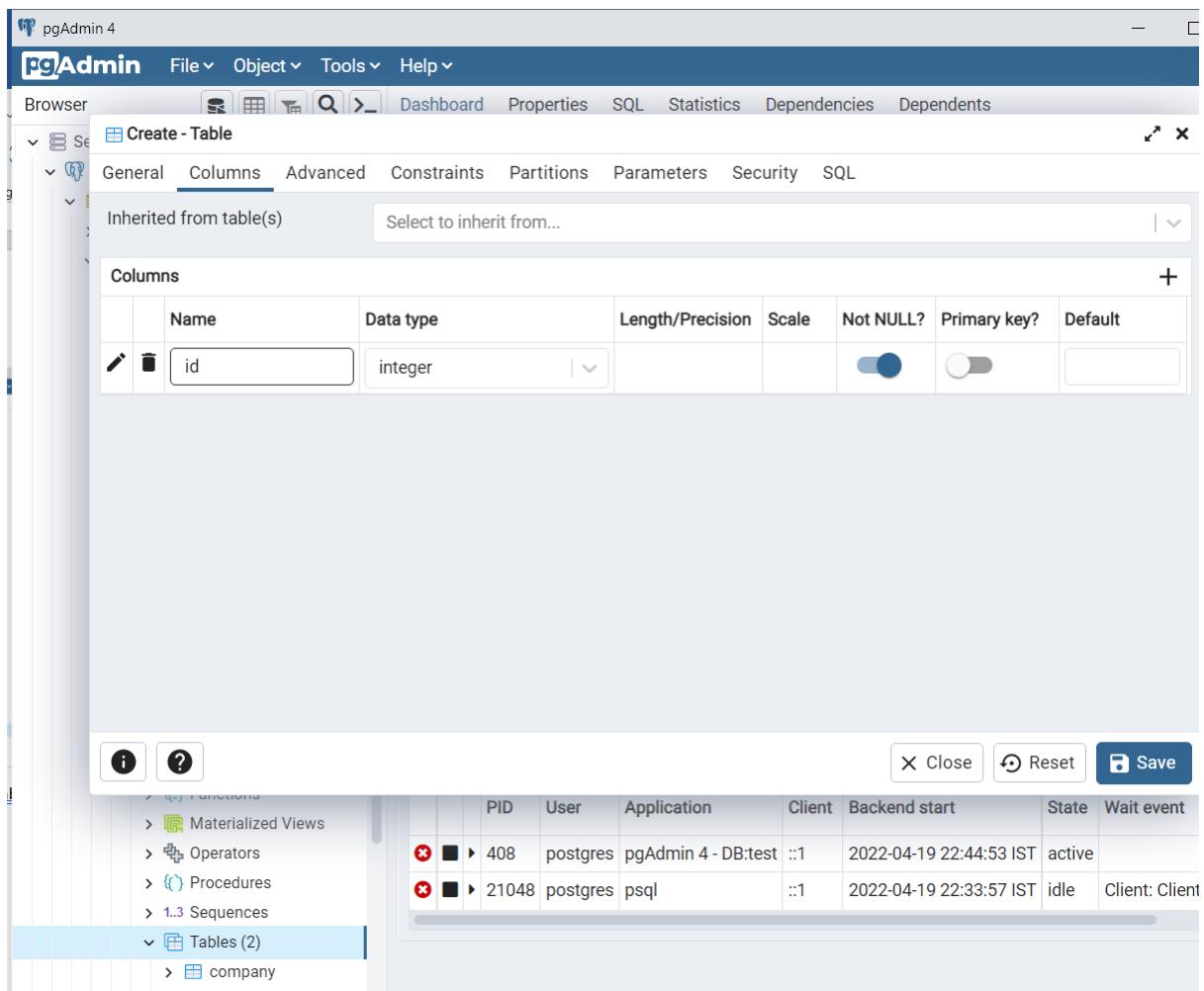
```

After doing all above steps table test will be available in pgadmin4. If not restart.

Belo how to reach to tables in pgadmin4.



Right clicking on Tables(2) icon to create new table. Filling in general ,column where to fill name of column etc.



pgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

- > Casts
- > Catalogs
- > Event Triggers
- > Extensions
- > Foreign Data Wrappers
- > Languages
- > Publications
- > Schemas (1)
  - > public
    - > Aggregates
    - > Collations
    - > Domains
    - > FTS Configurations
    - > FTS Dictionaries
    - > FTS Parsers
    - > FTS Templates
    - > Foreign Tables
    - > Functions
    - > Materialized Views
    - > Operators
    - > Procedures
    - > Sequences
  - > Tables (3)
    - > company
    - > department
    - > student
  - > Trigger Functions
  - > Types
  - > Views

Create - Column

General Definition Constraints Variables Security SQL

Name name

Comment

Close Reset Save Search

PID	User	Application	Client	Backend start	State	Wait event
408	postgres	pgAdmin 4 - DB:test	::1	2022-04-19 22:44:53 IST	active	
21048	postgres	psql	::1	2022-04-19 22:33:57 IST	idle	Client: Client

Transactions Commits R

Block I/O Read:

450  
400  
350  
300  
250  
200  
150  
100  
50  
0

pgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

- > Casts
- > Catalogs
- > Event Triggers
- > Extensions
- > Foreign Data Wrappers
- > Languages
- > Publications
- > Schemas (1)
  - public
    - > Aggregates
    - > Collations
    - > Domains
    - > FTS Configurations
    - > FTS Dictionaries
    - > FTS Parsers
    - > FTS Templates
    - > Foreign Tables
    - > Functions
    - > Materialized Views
    - > Operators
    - > Procedures
    - > 1.3 Sequences
  - > Tables (3)
    - > company
    - > department
    - > student
  - > Trigger Functions
  - > Types
  - > Views

Create - Column

General Definition Constraints Variables Security SQL

Data type: text

Length/Precision:

Scale:

Collation: Select an item...

Close Reset Save

Transactions Commits R

Block I/O Read:

PID	User	Application	Client	Backend start	State	Wait event
408	postgres	pgAdmin 4 - DB:test	::1	2022-04-19 22:44:53 IST	active	
21048	postgres	psql	::1	2022-04-19 22:33:57 IST	idle	Client: Client

Search

pgAdmin 4

File ▾ Object ▾ Tools ▾ Help ▾

Browser

- > Casts
- > Catalogs
- > Event Triggers
- > Extensions
- > Foreign Data Wrappers
- > Languages
- > Publications
- > Schemas (1)
  - public
    - > Aggregates
    - > Collations
    - > Domains
    - > FTS Configurations
    - > FTS Dictionaries
    - > FTS Parsers
    - > FTS Templates
    - > Foreign Tables
    - > Functions
    - > Materialized Views
    - > Operators
    - > Procedures
    - > Sequences
  - > Tables (3)
    - > company
    - > department
    - > student
  - > Trigger Functions
  - > Types
  - > Views

Create - Column

General Definition Constraints Variables Security SQL

Default:

Not NULL?

Type:  NONE  IDENTITY  GENERATED

Block I/O

Reads: 450, 400, 350, 300, 250, 200, 150, 100, 50, 0

PID	User	Application	Client	Backend start	State	Wait event
408	postgres	pgAdmin 4 - DB:test	::1	2022-04-19 22:44:53 IST	active	
21048	postgres	psql	::1	2022-04-19 22:33:57 IST	idle	Client: Client

Id as primary key.

The screenshot shows the pgAdmin 4 interface. On the left, the 'Browser' pane displays a tree view of database objects under the 'public' schema, including Languages, Publications, Schemas (1), Tables (3), and student (with Columns: id, name). A 'Constraints' node is selected. In the center, a modal window titled 'Create - Primary key' is open, showing the 'Definition' tab with 'Columns' set to 'id'. To the right, a monitoring interface displays a 'Transactions' chart with a single sharp peak and a 'Block I/O' chart showing activity over time.

**Create - Primary key**

General Definition SQL

Columns: id

Include columns: Select an item...

Tablespace: Select an item...

Index: Select an item...

Fill factor:

Deferrable? (switch off)

Deferred? (switch off)

**Block I/O**

PID	User	Application	Client	Backend start	State	Wait event
408	postgres	pgAdmin 4 - DB:test	::1	2022-04-19 22:44:53 IST	active	
21048	postgres	psql	::1	2022-04-19 22:33:57 IST	idle	Client: Client

To write query use editor from tools.

Right click on tables red. To delete.

To delete a table use – drop table table-name;

The screenshot shows the pgAdmin 4 interface. The left sidebar (Browser) displays a tree view of database objects under the 'test/postgres@...' connection. The 'Tables' node is expanded, showing 'company', 'department', and 'student'. The 'student' table is further expanded to show its columns ('id', 'name') and constraints ('student\_pkey'). The right pane (Query Editor) contains the SQL command 'drop table student;' and its execution results: 'DROP TABLE' and 'Query returned successfully in 77 msec.'

```
1 drop table student;
```

```
DROP TABLE
```

```
Query returned successfully in 77 msec.
```

```
test=# drop table student;
ERROR:  table "student" does not exist
test=#
```

As done after doing from query editor.

```
insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000)
```

test/postgres@PostgreSQL 14 ▾

Query Editor    Query History    Scratch Pad

```
1 insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000)
```

Data Output    Explain    Messages    Notifications

INSERT 0 1

Query returned successfully in 117 msec.

test/postgres@PostgreSQL 14 ▾

Query Editor    Query History

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000)
2 select * from company;
```

Data Output    Explain    Messages    Notifications

	<u>id</u> [PK] integer	<u>name</u> text	<u>age</u> integer	<u>address</u> character (50)	<u>salary</u> real	
1	7	Rahul	20	Kolkata	...	50000

test/postgres@PostgreSQL 14 ▾

Query Editor Query History

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000)
2 --select * from company;
3 insert into company values(4,'Abhay',24,'Delhi',30000)
```

Data Output Explain Messages Notifications

INSERT 0 1

Query returned successfully in 87 msec.

test/postgres@PostgreSQL 14 ▾

Query Editor Query History

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000)
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000)
4 insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000)
```

Data Output Explain Messages Notifications

INSERT 0 2

Query returned successfully in 75 msec.

FORGOT SEMICOLON IN ABOVE STEPS NEED TO PUT SEMICOLON AFTER EACH QUERY.

```

1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);

```

	<b>id</b> [PK] integer	<b>name</b> text	<b>age</b> integer	<b>address</b> character (50)	<b>salary</b> real
1	7	Rahul	20	Kolkata	50000
2	4	Abhay	24	Delhi	30000
3	1	Sunil	12	Mumbai	30000
4	2	Henna	16	Mumbai	30000

	<b>id</b> [PK] integer	<b>name</b> text	<b>age</b> integer	<b>address</b> character (50)	<b>salary</b> real
1	1	Sunil	12	Mumbai	30000
2	2	Henna	16	Mumbai	30000
3	4	Abhay	24	Delhi	30000
4	7	Rahul	20	Kolkata	50000

FOR VIEWING DATA IN THE TABLE SELECT COMPANY BY MOUSE AND CLICK ON VIEW DATA BUTTON MARKED BY ARROW. We Can also enter new data by clicking on the cells below.

```
1 SELECT * FROM public.company  
2 ORDER BY id ASC
```

Data Output Explain Messages Notifications

	<b>id</b> [PK] integer		<b>name</b> text		<b>age</b> integer		<b>address</b> character (50)		<b>salary</b> real	
1			1 Sunil		12	Mumbai	...		30000	
2			2 Henna		16	Mumbai	...		30000	
3			4 Abhay		24	Delhi	...		30000	
4			7 Rahul		20	Kolkata	...		50000	
5			5 Rashi		34	Bangalore			[null]	

Make sure to save the data.

test/postgres@PostgreSQL 14 ▾

	Query Editor	Scratch Pad
1	--insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000); 2 --select * from company; 3 --insert into company values(4,'Abhay',24,'Delhi',30000); 4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000); 5 select id,name from company;	

Data Output Explain Messages Notifications

	<b>id</b> [PK] integer	<b>name</b> text
1	7	Rahul
2	4	Abhay
3	1	Sunil
4	2	Henna
5	5	Rashi

test/postgres@PostgreSQL 14 ▾

	Query Editor	Scratch Pad
1	--insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000); 2 --select * from company; 3 --insert into company values(4,'Abhay',24,'Delhi',30000); 4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000); 5 --select id,name from company; 6 select * from company;	

Data Output Explain Messages Notifications

	<b>id</b> [PK] integer	<b>name</b> text	<b>age</b> integer	<b>address</b> character (50)	<b>salary</b> real
1	7	Rahul	20	Kolkata	50000
2	4	Abhay	24	Delhi	30000
3	1	Sunil	12	Mumbai	30000
4	2	Henna	16	Mumbai	30000
5	5	Rashi	34	Bangalore	54000

```
SQL Shell (psql)
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Password for user postgres:
psql (14.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=# \c test
You are now connected to database "test" as user "postgres".
test=# \d company
           Table "public.company"
 Column |      Type       | Collation | Nullable | Default
-----+-----+-----+-----+-----+
 id    | integer        |           | not null |
 name  | text           |           | not null |
 age   | integer        |           | not null |
 address | character(50) |           |           |
 salary | real           |           |           |
Indexes:
"company_pkey" PRIMARY KEY, btree (id)
```

```
test=# select * from company;
 id | name | age | address | salary
----+-----+-----+-----+-----+
 7 | Rahul | 20 | Kolkata | 50000
 4 | Abhay | 24 | Delhi | 30000
 1 | Sunil | 12 | Mumbai | 30000
 2 | Henna | 16 | Mumbai | 30000
 5 | Rashi | 34 | Bangalore | 54000
(5 rows)

test=# select * from department;
 id | dept | emp_id
----+-----+-----+
(0 rows)

test=# select * from company;
 id | name | age | address | salary
----+-----+-----+-----+-----+
 7 | Rahul | 20 | Kolkata | 50000
 4 | Abhay | 24 | Delhi | 30000
 1 | Sunil | 12 | Mumbai | 30000
 2 | Henna | 16 | Mumbai | 30000
 5 | Rashi | 34 | Bangalore | 54000
(5 rows)

test=#

```

 SQL Shell (psql)

```
Server [localhost]:  
Database [postgres]:  
Port [5432]:  
Username [postgres]:  
Password for user postgres:  
psql (14.2)  
WARNING: Console code page (437) differs from Windows code page (1252)  
          8-bit characters might not work correctly. See psql reference  
          page "Notes for Windows users" for details.  
Type "help" for help.  
  
postgres=# select 2+3;  
?column?  
-----  
      5  
(1 row)  
  
postgres=# select 10/5;  
?column?  
-----  
      2  
(1 row)  
  
postgres=#
```

```
postgres=# \! cls
```

Above command to clear screen.

```
postgres=# select 2^3 ;
?column?
-----
     8
(1 row)
```

```
postgres=# select 14%4;
?column?
-----
     2
(1 row)
```

```
postgres=#
```

SQL Shell (psql)

```
postgres=# select |/25;
?column?
-----
      5
(1 row)
```

```
postgres=#
```

Square root of 25 is 5.

```
SQL Shell (psql)
postgres=# select |/25;
?column?
-----
      5
(1 row)

postgres=# select ||/27;
?column?
-----
      3
(1 row)

postgres=#
```

```
test=# select name from company where age = 34;
 name
-----
 Rashi
(1 row)
```

```
test=# select name from company where age != 34;
 name
-----
 Rahul
 Abhay
 Sunil
 Henna
(4 rows)
```

```
test=# select * from company where age <> 34;
 id | name | age | address | salary
----+-----+-----+-----+
 7 | Rahul | 20 | Kolkata | 50000
 4 | Abhay | 24 | Delhi | 30000
 1 | Sunil | 12 | Mumbai | 30000
 2 | Henna | 16 | Mumbai | 30000
(4 rows)
```

`<>` is equivalent to (not equal to) or `!=` .

```
test=# select * from company where age > 12;
+----+-----+-----+-----+
| id | name | age | address | salary |
+----+-----+-----+-----+
| 7  | Rahul | 20  | Kolkata  | 50000  |
| 4  | Abhay | 24  | Delhi    | 30000  |
| 2  | Henna | 16  | Mumbai   | 30000  |
| 5  | Rashi | 34  | Bangalore| 54000  |
+----+-----+-----+-----+
(4 rows)
```

For greater than or equal to use → `>=` .

```
test=# select * from company where age>13 and salary>2000;
+----+-----+-----+-----+
| id | name | age | address | salary |
+----+-----+-----+-----+
| 7  | Rahul | 20  | Kolkata  | 50000  |
| 4  | Abhay | 24  | Delhi    | 30000  |
| 2  | Henna | 16  | Mumbai   | 30000  |
| 5  | Rashi | 34  | Bangalore| 54000  |
+----+-----+-----+-----+
(4 rows)
```

```
test=# select * from company where age>13 or salary>2000;
+----+-----+-----+-----+
| id | name | age | address | salary |
+----+-----+-----+-----+
| 7  | Rahul | 20  | Kolkata  | 50000  |
| 4  | Abhay | 24  | Delhi    | 30000  |
| 1  | Sunil | 12  | Mumbai   | 30000  |
| 2  | Henna | 16  | Mumbai   | 30000  |
| 5  | Rashi | 34  | Bangalore| 54000  |
+----+-----+-----+-----+
(5 rows)
```

```
test=# select * from company where salary is null;
+----+-----+-----+-----+
| id | name | age | address | salary |
+----+-----+-----+-----+
+----+-----+-----+-----+
(0 rows)
```

```
test=# select * from company where name like 'R%';
+----+-----+-----+-----+
| id | name | age | address | salary |
+----+-----+-----+-----+
| 7  | Rahul | 20  | Kolkata  | 50000  |
| 5  | Rashi | 34  | Bangalore| 54000  |
+----+-----+-----+-----+
(2 rows)
```

```
test=# select * from company where name like '%a%';
 id | name | age | address | salary
---+-----+-----+-----+-----+
 7 | Rahul | 20 | Kolkata | 50000
 4 | Abhay | 24 | Delhi | 30000
 2 | Henna | 16 | Mumbai | 30000
 5 | Rashi | 34 | Bangalore | 54000
(4 rows)
```

```
test=# select * from company where age in ('20','34','16');
 id | name | age | address | salary
---+-----+-----+-----+-----+
 7 | Rahul | 20 | Kolkata | 50000
 2 | Henna | 16 | Mumbai | 30000
 5 | Rashi | 34 | Bangalore | 54000
(3 rows)
```

Above query means select row where age is 20 or 34 or 16.

```
test=# select * from company where age between 20 and 34;
 id | name | age | address | salary
---+-----+-----+-----+-----+
 7 | Rahul | 20 | Kolkata | 50000
 4 | Abhay | 24 | Delhi | 30000
 5 | Rashi | 34 | Bangalore | 54000
(3 rows)
```

-- used for single line comment.

/\*-----\*/ used for multiline comment.

The screenshot shows a PostgreSQL client interface with the following details:

- Query Editor:** Contains the following SQL code:

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);
5 --select id,name from company;
6 --select * from company;
7 select name as employee_name from company;
```
- Data Output:** Shows a table with the column `employee_name` and 5 rows of data:

	employee_name
1	Rahul
2	Abhay
3	Sunil
4	Henna
5	Rashi

test/postgres@PostgreSQL 14 ▾

Query Editor    Query History

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);
5 --select id,name from company;
6 --select * from company;
7 --select name as employee_name from company;
8 select count(*) from company;
```

Data Output    Explain    Messages    Notifications

	count	bigint
1	5	

test/postgres@PostgreSQL 14 ▾

Query Editor    Query History

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);
5 --select id,name from company;
6 --select * from company;
7 --select name as employee_name from company;
8 --select count(*) from company;
9 select max(age) from company;
```

Data Output    Explain    Messages    Notifications

	max	integer
1	34	

```

1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);
5 --select id,name from company;
6 --select * from company;
7 --select name as employee_name from company;
8 --select count(*) from company;
9 select min(age) from company;

```

Data Output Explain Messages Notifications

	min	integer
1	12	

The screenshot shows a pgAdmin 4 interface. At the top is a toolbar with various icons for file operations, search, and navigation. Below the toolbar is a header bar displaying the connection information: **test/postgres@PostgreSQL 14**. Underneath the header is a tab bar with **Query Editor** and **Query History**, with **Query Editor** currently selected. The main area contains the following SQL code:

```

1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);
5 --select id,name from company;
6 --select * from company;
7 --select name as employee_name from company;
8 --select count(*) from company;
9 --select min(age) from company;
10 select sum(salary) from company;
11

```

Below the code is a **Data Output** section with tabs for **Explain**, **Messages**, and **Notifications**. A results table is displayed:

	sum	real
1	194000	



test/postgres@PostgreSQL 14 ▾

Query Editor Query History

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);
5 --select id,name from company;
6 --select * from company;
7 --select name as employee_name from company;
8 --select count(*) from company;
9 --select min(age) from company;
10 select avg(salary) from company;
11
```

Data Output Explain Messages Notifications

	avg	double precision
1	38800	



test/postgres@PostgreSQL 14 ▾

Query Editor Query History

```
1 --insert into company(id,name,age,address,salary)values(7,'Rahul',20,'Kolkata',50000);
2 --select * from company;
3 --insert into company values(4,'Abhay',24,'Delhi',30000);
4 --insert into company values(1,'Sunil',12,'Mumbai',30000),(2,'Henna',16,'Mumbai',30000);
5 --select id,name from company;
6 --select * from company;
7 --select name as employee_name from company;
8 --select count(*) from company;
9 --select min(age) from company;
10 --select avg(salary) from company;
11 select current_timestamp;
```

Data Output Explain Messages Notifications

	current_timestamp	timestamp with time zone
1	2022-04-24 21:35:28.636283+05:30	

Above query to get current time,date etc.



product/postgres@PostgreSQL 14 ▾

Query Editor Query History

```
1 create table products(
2 product_no integer,
3 name text,
4 price numeric CHECK (price>0)
5 );
```

Data Output Explain Messages Notifications

CREATE TABLE

Query returned successfully in 134 msec.

CHECK constraint to check for price always &gt; 0.



product/postgres@PostgreSQL 14 ▾

Query Editor Query History

```
1 /*create table products(
2 product_no integer,
3 name text,
4 price numeric CHECK (price>0)
5 */;
6 insert into products values(10,'Soap',-10);
```

Data Output Explain Messages Notifications

ERROR: new row for relation "products" violates check constraint "products\_price\_check"  
DETAIL: Failing row contains (10, Soap, -10).  
SQL state: 23514



product/postgres@PostgreSQL 14 ▾

Query Editor    Query History

```
1 /*create table products(
2 product_no integer,
3 name text,
4 price numeric CHECK (price>0)
5 );*/
6 insert into products values(10,'Soap',10);
```

Data Output    Explain    Messages    Notifications

INSERT 0 1

Query returned successfully in 121 msec.

The screenshot shows a PostgreSQL query editor interface. At the top is a toolbar with various icons for database management tasks like creating tables, folders, and queries. To the right of the toolbar is a dropdown menu set to "No limit". Below the toolbar, there are two tabs: "Query Editor" (which is selected) and "Query History". The main area contains a code editor with the following SQL script:

```
1 create table product_2(
2     product_no integer,
3     name text,
4     price numeric CHECK(price>0),
5     discounted_price numeric CHECK(discounted_price>0),
6     CHECK(price>discounted_price)
7 )
```

Data Output Explain Messages Notifications

CREATE TABLE

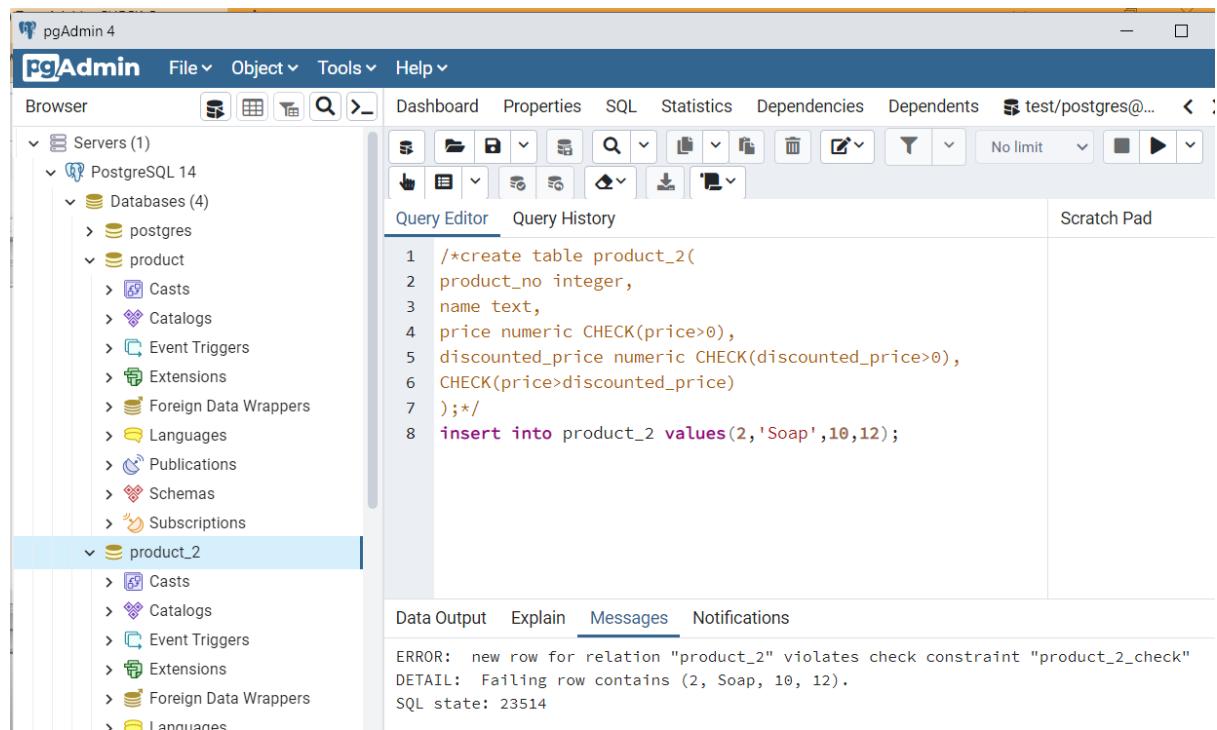
Query returned successfully in 179 msec.

```
1 /*create table product_2(
2 product_no integer,
3 name text,
4 price numeric CHECK(price>0),
5 discounted_price numeric CHECK(discounted_price>0),
6 CHECK(price>discounted_price)
7 );*/
8 insert into product_2 values(2, 'Soap', 10, -4);
```

Scratch

Data Output Explain Messages Notifications

```
ERROR: new row for relation "product_2" violates check constraint  
"product_2_discounted_price_check"  
DETAIL: Failing row contains (2, Soap, 10, -4).  
SQL state: 23514
```



The screenshot shows a PostgreSQL query editor interface. The top bar contains various icons for file operations, search, and filtering. Below the toolbar, tabs for "Query Editor" and "Query History" are visible, with "Query Editor" being active. A "Scratch Pad" tab is also present. The main area displays the following SQL code:

```
1 /*create table product_2
2 product_no integer,
3 name text,
4 price numeric CHECK(price>0),
5 discounted_price numeric CHECK(discounted_price>0),
6 CHECK(price>discounted_price)
7 */;
8 insert into product_2 values(2,'Soap',10,8);
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the output:

INSERT 0 1

Query returned successfully in 99 msec.

The screenshot shows a PostgreSQL query editor interface. The top bar contains various icons for file operations, search, and filtering. Below the toolbar, tabs for "Query Editor" and "Query History" are visible, with "Query Editor" being active. A "Scratch Pad" tab is also present. The main area displays the following SQL code:

```
1 create table product_3(
2 product_no integer NOT NULL,
3     name text NOT NULL,
4     price numeric NOT NULL CHECK(price>0)
5 );
```

Below the code, there are tabs for "Data Output", "Explain", "Messages", and "Notifications". The "Messages" tab is selected, showing the output:

CREATE TABLE

Query returned successfully in 113 msec.

Query Editor    Query History

Scratch Pad X

```
1 /*create table product_3(
2   product_no integer NOT NULL,
3     name text NOT NULL,
4     price numeric NOT NULL CHECK(price>0)
5 );*/
6 insert into product_3 values(12,NULL,-5);
```

Data Output    Explain    Messages    Notifications

ERROR: null value in column "name" of relation "product\_3" violates not-null constraint  
DETAIL: Failing row contains (12, null, -5).  
SQL state: 23502

Query Editor    Query History

Scratch Pad

```
1 /*create table product_3(
2   product_no integer NOT NULL,
3     name text NOT NULL,
4     price numeric NOT NULL CHECK(price>0)
5 );*/
6 insert into product_3 values(12,'Soap',67);
```

Data Output    Explain    Messages    Notifications

INSERT 0 1

Query returned successfully in 111 msec.

Query Editor   Query History

Scratch Pad

```
1 create table product_4(
2     product_no integer,
3     name text,
4     price numeric,
5     UNIQUE(product_no, name)
6 );
```

Data Output   Explain   Messages   Notifications

CREATE TABLE

Query returned successfully in 195 msec.

Query Editor   Query History

Scratch Pad

```
1 /*create table product_4_1(
2     product_no integer UNIQUE,
3     name text UNIQUE,
4     price numeric,
5     UNIQUE(product_no, name)
6 ); */
7 --insert into product_4_1 values(3,'Plate',100);
8 insert into product_4 values(7,'Cup',100);
```

Data Output   Explain   Messages   Notifications

ERROR: duplicate key value violates unique constraint "product\_4\_product\_no\_name\_key"  
DETAIL: Key (product\_no, name)=(7, Cup) already exists.  
SQL state: 23505

Query Editor    Query History    Scratch Pad

```

1 /*create table product_5(
2     product_no integer,
3         name text,
4         price numeric,
5             PRIMARY KEY(name,product_no)
6 );*/
7 create table product_5_1(
8     product_no integer PRIMARY KEY,
9         name text,
10        price numeric
11 );

```

Data Output   Explain   Messages **Messages**   Notifications

CREATE TABLE

Query returned successfully in 91 msec.

INSSERTING DATA BY→

The screenshot shows the PgAdmin 4 application interface. On the left is the 'Browser' pane, which displays the database schema structure. Under the 'Schemas' node, the 'public' schema is expanded, showing various objects like Aggregates, Collations, Domains, FTS Configurations, FTS Dictionaries, FTS Parsers, FTS Templates, Foreign Tables, Functions, Materialized Views, Operators, Procedures, Sequences, and Tables. The 'product\_5\_1' table is selected in the 'Tables' section. On the right is the 'Query Editor' pane, containing the following SQL code:

```

1 SELECT * FROM public.product_5_1
2 ORDER BY product_no ASC

```

The 'Messages' tab is active in the Query Editor. Below the Query Editor is a preview of the table structure:

product_no	[PK] integer	name	text	price	numeric

Error comes when order\_id or product\_no does not exist.

Query Editor    Query History    Scratch Pad

```
1 create table orders(
2     order_id integer PRIMARY KEY,
3     product_no integer REFERENCES product_6(product_no),
4     quantity integer
5 )
```

	order_id [PK] integer	product_no integer	quantity integer
1	1	1	2
2	2	3	4
*			

An error has occurred:

ERROR: insert or update on table "orders" violates foreign key constraint "orders\_product\_no\_fkey"  
DETAIL: Key (product\_no)=(3) is not present in table "products".

OK

SQL Editor    Graphical Query Builder

Previous queries

```
create table order_items
(
    product_no integer REFERENCES products(product_no),
    order_id integer REFERENCES orders(order_id),
    quantity integer,
    PRIMARY KEY(product_no,order_id)
);
```

On Delete Restrict: **when data is removed from a parent table, and there is a foreign key associated with child table it gives error**, you can not delete the record.

ON DELETE RESTRICT means **you can't delete a given parent row if a child row exists that references the value for that parent row**. If the parent row has no referencing child rows, then you can delete that parent row.

**SQL Editor** Graphical Query Builder

Previous queries

```
create table order_items
(
    product_no integer REFERENCES products(product_no) ON DELETE RESTRICT,
    order_id integer REFERENCES orders(order_id),
    quantity integer,
    PRIMARY KEY(product_no , order_id)
);
```

ON DELETE CASCADE clause is **used to automatically remove the matching records from the child table when we delete the rows from the parent table**. It is a kind of referential action related to the foreign key.

**SQL Editor** Graphical Query Builder

Previous queries

```
create table order_items
(
    product_no integer REFERENCES products(product_no) ON DELETE RESTRICT,
    order_id integer REFERENCES orders(order_id) ON DELETE CASCADE,
    quantity integer,
    PRIMARY KEY(product_no , order_id)
);
```

**SQL Editor** Graphical Query Builder

Previous queries

```
alter table company ADD gender char(1);
```

Query - test on postgres@localhost:5432 \*

File Edit Query Favourites Macros View Help

SQL Editor Graphical Query Builder

Previous queries

```
select * from company;
```

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>	<b>likes</b> <b>text[]</b>	<b>gender</b> <b>character(1)</b>
1	8	Priyanashi	23	Delhi		{ZYA, ADD}	
2	10	Rishi	34	Mumbai		{ZYA, ADD}	
3	2	Heena	16	Mumbai	3000	{ZYA, ADD}	
4	5	Rashi	34	Bangalore	4000	{ZYA, ADD}	
5	9	Kumar	40	Delhi	2000	{ZYA, ADD}	
6	4	Priya	24	Delhi	3000	{ZYA, ADD}	

OK. | Back | Forward | Stop | Search the web and Windows | 1:00 / 4:40 | Icons | DO

Query - test on postgres@localhost:5432 \*

File Edit Query Favourites Macros View Help

SQL Editor Graphical Query Builder

Previous queries

```
alter table company DROP COLUMN gender;
```

SQL Editor Graphical Query Builder

Previous queries

```
select * from company;
```

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>	<b>likes</b> <b>text[]</b>
1	8	Priyanshi	23	Delhi		{ZYA,ADD}
2	10	Rishi	34	Mumbai		{ZYA,ADD}
3	2	Heena	16	Mumbai	3000	{ZYA,ADD}
4	5	Rashi	34	Bangalore	4000	{ZYA,ADD}
5	9	Kumar	40	Delhi	2000	{ZYA,ADD}
6	4	Friya	24	Delhi	3000	{ZYA,ADD}

OK. |◀▶| ⏪ ⏩ |Speaker| 1:42 / 4:40

Search the web and Windows

The screenshot shows a SQL editor interface with two panes. The top pane is titled 'SQL Editor' and contains a query window with the following text:

```
alter table company ADD CONSTRAINT pri PRIMARY KEY(id);
```

The bottom pane is titled 'Output pane' and contains the following message:

```
Query returned successfully with no result in 202 msec.
```

The above query used to add pri named primary key constraint .

The screenshot shows a SQL editor interface with two panes. The top pane is titled 'SQL Editor' and contains a query window with the following text:

```
ALTER TABLE COMPANY DROP CONSTRAINT pri;
```

The bottom pane is titled 'Output pane' and contains the following message:

```
Query returned successfully with no result in 202 msec.
```

The above command to drop constraint pri.

SQL Editor Graphical Query Builder

Previous queries

```
select * from company;
```

I

Output pane

Data Output Explain Messages History

	<b>id</b> integer	<b>name</b> text	<b>age</b> integer	<b>address</b> character(50)	<b>salary</b> real
1	8	Priyanshi	23	Delhi	
2	10	Rishi	34	Mumbai	
3	2	Heena	16	Mumbai	3000
4	5	Rashi	34	Bangalore	4000
5	9	Kumar	40	Delhi	2000
6	4	Priya	24	Delhi	3000

OK ⏪ ⏩ ⏴ ⏵ 1:11 / 3:14

Search the web and Windows

SQL Editor Graphical Query Builder

Previous queries

```
update company set age=24 where id=8;
```

SQL Editor Graphical Query Builder

Previous queries

```
select * from company;
```

Output pane

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
3	3	Kashif	34	bangalore	4000
4	9	Kumar	40	Delhi	2000
5	4	Priya	24	Delhi	3000
6	7	Rahul	20	Kolkata	5000
7	1	Sunil	12	Mumbai	3000
8	8	Priyanshi	24	Delhi	

OK. |◀▶| Search the web and Windows 1:51 / 3:14

SQL Editor Graphical Query Builder

Previous queries

```
delete from company where id = 8;
```

SQL Editor Graphical Query Builder

Previous queries

```
select * from company limit 4;
```

<

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	10	Rishi	34	Mumbai	
2	2	Heena	16	Mumbai	3000
3	5	Rashi	34	Bangalore	4000
4	9	Kumar	40	Delhi	2000

SQL Editor Graphical Query Builder

Previous queries

```
select * from company limit 4 offset 4;
```

<

Output pane

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	4	Priya	24	Delhi	3000
2	7	Rahul	20	Kolkata	5000
3	11	Prayag	33	Kolkata	9000
4	12	Rohit	34	Chennai	

Limit 4 is used to get first 4. Offset 4 is used to get next 4 after first 4.

SQL Editor Graphical Query Builder

Previous queries

```
select age, count(*) from company group by age;
```

Output pane

Data Output Explain Messages History

	age	count
	integer	bigint
1	40	1
2	34	3
3	35	1
4	20	1
5	16	1
6	24	1

OK. | Back | Forward | Stop | 1:22 / 3:21 | Search the web and Windows

SQL Editor Graphical Query Builder

Previous queries

```
select address, count(*) from company group by address;
```

Output pane

Data Output Explain Messages History

	address character(50)	count bigint
1	Chennai	1
2	Bangalore	1
3	Kolkata	2
4	Bengaluru	1
5	Mumbai	2
6	Delhi	2

OK. | Search the web and Windows | 1:58 / 3:21 |

SQL Editor Graphical Query Builder

Previous queries

```
select address,count(*) from company group by address having MAX(salary)>2000;
```

I

Output pane

Data Output Explain Messages History

	address character(50)	count bigint
1	Bangalore	1
2	Kolkata	2
3	Bengaluru	1
4	Mumbai	2
5	Delhi	2

SQL Editor Graphical Query Builder

Previous queries

```
select * from company order by salary;
```

Output pane

Data Output Explain Messages History

	<b>id</b> integer	<b>name</b> text	<b>age</b> integer	<b>address</b> character(50)	<b>salary</b> real
1	9	Kumar	40	Delhi	2000
2	4	Priya	24	Delhi	3000
3	2	Heena	16	Mumbai	3000
4	13	Kumar	35	Bengaluru	4000
5	5	Rashi	34	Bangalore	4000
6	7	Rahul	20	Kolkata	5000

OK. | Search the web and Windows | 1:00 / 4:12 |

SQL Editor Graphical Query Builder

Previous queries

```
select * from company order by age;
```

I

<

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	2	Heena	16	Mumbai	3000
2	7	Rahul	20	Kolkata	5000
3	4	Priya	24	Delhi	3000
4	11	Prayag	33	Kolkata	9000
5	10	Rishi	34	Mumbai	
6	5	Rashi	34	Bangalore	4000

OK. |◀▶| 🔍 1:21 / 4:12

Search the web and Windows

```
select * from company order by age DESC;
```

Employee Data					
	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1		9 Kumar	40	Delhi	2000
2		13 Kumar	35	Bengaluru	4000
3		10 Rishi	34	Mumbai	
4		5 Rashi	34	Bangalore	4000
5		12 Rohit	34	Chennai	
6		11 Prayag	33	Kolkata	9000

DESC FOR DESCENDING AND ASC FOR ASCENDING.

SQL Editor Graphical Query Builder

Previous queries

```
select * from company order by age,salary DESC;
```

Output pane

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	9	rriyd	24	Delhi	3000
4	11	Prayag	33	Kolkata	9000
5	10	Rishi	34	Mumbai	8000
6	5	Rashi	34	Bangalore	4000
7	12	Rohit	34	Chennai	4000
8	13	Kumar	35	Bengaluru	4000
9	9	Munir	40	Delhi	2000

OK. |◀|▶|>|🔉| 3:38 / 4:12 | Search the web and Windows |

ORDERED BY AGE IN ASCENDING ORDER AND SALARY IN DESCENDING ORDER.

SQL Editor Graphical Query Builder

Previous queries

```
create type mood as ENUM('sad','ok','happy');
create table person(
    name text,
    current_mood mood);
```

SQL Editor Graphical Query Builder

Previous queries

```
insert into person values('rahul','happy');
```

SQL Editor Graphical Query Builder

Previous queries

```
select * from person;
```

<

Output pane

Data Output Explain Messages History

	name	current_mood
	text	mood
1	rahul	happy
2	Rishi	ok
3	Rashi	sad

SQL Editor Graphical Query Builder

Previous queries

○ insert into person values('rahul','extremely happy');

< I

Output pane

Data Output Explain **Messages** History

```
ERROR: invalid input value for enum mood: "extremely happy"
LINE 1: insert into person values('rahul','extremely happy');
          ^
*****
***** Error *****

ERROR: invalid input value for enum mood: "extremely happy"
SQL state: 22P02
Character: 35
```

ERROR AS EXTREMELY HAPPY WAS NOT DECLARED IN THE ENUMERATION.

SQL Editor Graphical Query Builder

Previous queries

```
select * from person order by current_mood;
```

<

Output pane

Data Output Explain Messages History

	name	current_mood
	text	mood
1	Rashi	sad
2	Rishi	ok
3	rahul	happy

SQL Editor Graphical Query Builder

Previous queries

```
select min(current_mood) from person;
```

<

Output pane

Data Output Explain Messages History

	min mood
1	sad

SQL Editor Graphical Query Builder

Previous queries

```
select max(current_mood) from person;
```

<

Output pane

Data Output Explain Messages History

	max mood
1	happy

SQL Editor Graphical Query Builder

Previous queries

```
select count(*) from company;
```

<

Output pane

Data Output Explain Messages History

	count	bigint
1	9	

SQL Editor Graphical Query Builder

Previous queries

```
select max(age) from company;
```

<

Output pane

Data Output Explain Messages History

	max integer
1	40

SQL Editor Graphical Query Builder

Previous queries

```
select avg(age) from company;
```

<

Output pane

Data Output Explain Messages History

	avg numeric
1	30.000000000000000000

SQL Editor Graphical Query Builder

Previous queries

```
select sum(age) from company;
```

<

Output pane

Data Output Explain Messages History

	sum	bigint
1	270	

SQL Editor Graphical Query Builder

Previous queries

```
select array_agg(name) from company;
```

<

Output pane

Data Output Explain Messages History

	array_agg text[]
1	{Heena, Rashi, Kumar, Priya, Rahul, Prayag, Kumar, Rishi, Roh}

THE ABOVE QUERY RETURNS AN ARRAY OF NAMES FROM COMPANY TABLE.

SQL Editor Graphical Query Builder

Previous queries

```
select json_agg(name) from company;
```

Output pane

Data Output Explain Messages History

	json_agg json
1	[ "Heena", "Rashi", "Kumar", "Priya", "Rahul", "Prayag", "Kumar", "Rishi", "Rohit" ]

THE ABOVE QUERY GIVES JSON FORM OF OUTPUT OF NAMES.

SQL Editor Graphical Query Builder

Previous queries

```
select json_object_agg(name,age) from company;
```

Output pane

Data Output Explain Messages History

	json_object_agg json
1	{ "Heena" : 16, "Rashi" : 34, "Kumar" : 40, "Priya" : 24, "Rahul" : 20, "Prayag" : 33, "Kumar" : 35, "Rishi" : 34, "Rohit" : 34 }

THE ABOVE QUERY GIVES KEY-VALUE FORMAT OF NAME AND AGE IN JSON FORMAT.

SQL Editor Graphical Query Builder

Previous queries

```
select stddev(age) from company;
```

< Output pane

Data Output Explain Messages History

	stddev numeric
1	8.0156097709406987

SQL Editor | Graphical Query Builder

Previous queries

```
select variance(age) from company;
```

<

Output pane

Data Output Explain Messages History

	variance numeric
1	64.25000000000000

SQL Editor | Graphical Query Builder

Previous queries

```
select mode() within group(order by age) from company;
```

<

Output pane

Data Output Explain Messages History

	mode integer
1	34

THE ABOVE QUERY RETURNS THE MOST OCCURING AGE IN THE COMPANY TABLE.

SQL Editor Graphical Query Builder

Previous queries

```
select mode() within group(order by salary) from company;
```

I

<

Output pane

Data Output Explain Messages History

	mode real
1	4000

RETURNS THE MOST OCCURRING SALARY FROM THE TABLE.

SQL Editor Graphical Query Builder

Previous queries

```
select * from company where name like 'Pt';
```

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	4	Priya	24	Delhi	3000
2	11	Prayag ^o	33	Kolkata	9000

SQL Editor Graphical Query Builder

Previous queries

```
select * from company where name like '_rat';
```

I

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	11	Prayag	33	Kolkata	9000

SQL Editor | Graphical Query Builder

Previous queries

```
select * from company where name like '%at%';
```

<

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
3	9	Kumar	40	Delhi	2000
4	4	Priya	24	Delhi	3000
5	7	Rahul	20	Kolkata	5000
6	11	Prayag	33	Kolkata	9000
7	12	Kumar	25	Bengaluru	4000

OK. |◀|▶|▶| 5:04 / 7:41

SQL Editor Graphical Query Builder

Previous queries

```
select * from company where name like 'P_#_#_#';
```

I

<

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	4	Priya	24	Delhi	3000
2	11	Prayag	33	Kolkata	9000

Name to be atleast 5 characters long.

SQL Editor Graphical Query Builder

Previous queries

```
select * from company where name like '%a';
```

I

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
1	2	Heena	16	Mumbai	3000
2	4	Priya	24	Delhi	3000

The above query for words ending with 'a'.

SQL Editor | Graphical Query Builder

Previous queries

```
select * from company
```

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>
3	9	Kumar	40	Delhi	2000
4	4	Priya	24	Delhi	3000
5	7	Rahul	20	Kolkata	5000
6	11	Prayag	33	Kolkata	9000
7	13	Kumar	35	Bengaluru	4000
8	10	Rishi	34	Mumbai	8000
9	12	Rohit	34	Chennai	4000

SQL Editor | Graphical Query Builder

Previous queries

```
select * from department;
```

Output pane

	<b>id</b> <b>integer</b>	<b>dept</b> <b>character(50)</b>	<b>emp_id</b> <b>integer</b>
1	1	IT Billing	1
2	2	Engineering	2
3	3	Finance	7

IF TABLE X HAS x ROWS AND TABLE Y HAS y ROWS THEN THE TABLE FORMED BY THEIR CROSS JOIN HAS  $x \times y$  ROWS.

SQL Editor | Graphical Query Builder

Previous queries

```
select emp_id, name, dept from company cross join department;
```

Output pane

	<b>emp_id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>dept</b> <b>character(50)</b>
1	1	Heena	IT Billing
2	1	Rashi	IT Billing
3	1	Kumar	IT Billing
4	1	Priya	IT Billing
5	1	Rahul	IT Billing
6	1	Prayag	IT Billing
7	1	Kumar	IT Billing
8	1	Rishi	IT Billing
9	1	Rohit	IT Billing
10	2	Heena	Engineering
11	2	Rashi	Engineering
12	2	Kumar	Engineering
13	2	Priya	Engineering
14	2	Rahul	Engineering
15	2	Prayag	Engineering
16	2	Kumar	Engineering
17	2	Rishi	Engineering
18	2	Rohit	Engineering
19	7	Heena	Finance
20	7	Rashi	Finance
21	7	Kumar	Finance
22	7	Priya	Finance
23	7	Rahul	Finance
24	7	Prayag	Finance

THE ABOVE TABLE FORMED BY CROSS JOIN HAS  $9 \times 3 = 27$  ROWS.

SQL Editor Graphical Query Builder

Previous queries

Output pane

Data Output Explain Messages History

	emp_id integer	name text	dept character(50)
2	1	Rashi	IT Billing
3	1	Kumar	IT Billing
4	1	Priya	IT Billing
5	1	Rahul	IT Billing
6	1	Prayag	IT Billing
7	1	Kumar	IT Billing
8	1	Rishi	IT Billing
9	1	Rohit	IT Billing
10	2	Heena	Engineering
11	2	Rashi	Engineering
12	2	Kumar	Engineering
13	2	Priya	Engineering
14	2	Rahul	Engineering
15	2	Prayag	Engineering
16	2	Kumar	Engineering
17	2	Rishi	Engineering
18	2	Rohit	Engineering
19	7	Heena	Finance
20	7	Rashi	Finance
21	7	Kumar	Finance
22	7	Priya	Finance
23	7	Rahul	Finance
24	7	Prayag	Finance
25	7	Kumar	Finance
26	7	Rishi	Finance
27	7	Rohit	Finance

SQL Editor Graphical Query Builder

Previous queries

```
select name,age,address,dept from company inner join department on company.id=department.id
```

Output pane

	name text	age integer	address character(50)	dept character(50)
1	Heena	16	Mumbai	Engineering

SQL Editor Graphical Query Builder

Previous queries

```
select name,age,address,dept from company c , department d
where c.id=d.id
```

Output pane

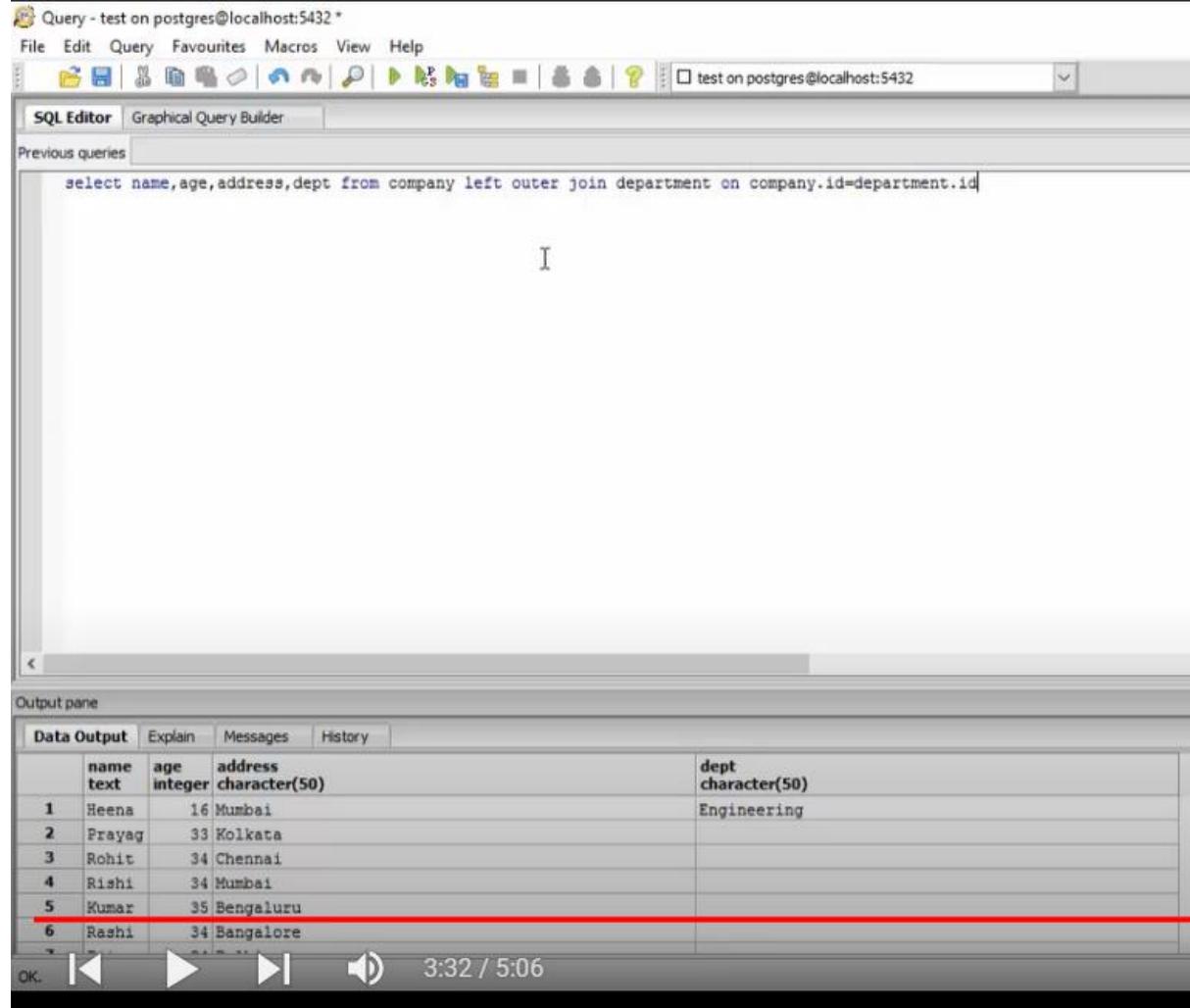
	name text	age integer	address character(50)	dept character(50)
1	Heena	16	Mumbai	Engineering

IN THE ABOVE 2 QUERIES NAME,AGE AND ADDRESS ARE FROM COMPANY TABLE AND DEPT FROM DEPARTMENT TABLE.

INNER JOIN RETURNS ONLY THE COLUMNS COMMON TO BOTH TABLES.

IN OUTER JOIN FIRST INNER JOIN IS DONE THEN THE VALUES THAT DON'T MATCH ARE REPLACED BY NULL VALUES.

X LEFT OUTER JOIN Y → MEANS ALL COLUMN VALUES OF X IS FULLY DISPLAYED BUT THOSE VALUES IN Y THAT DON'T MATCH WITH ANY ROW ARE KEPT NULL.



The screenshot shows a PostgreSQL client interface with the following details:

- Toolbar:** File, Edit, Query, Favourites, Macros, View, Help.
- Query Bar:** "Query - test on postgres@localhost:5432 \*".
- SQL Editor:** "SQL Editor" tab selected, "Graphical Query Builder" tab visible. The query entered is: "select name,age,address,dept from company left outer join department on company.id=department.id".
- Output pane:** "Data Output" tab selected, showing the results of the query. The results are:

	name	age	address	dept
	text	integer	character(50)	character(50)
1	Heena	16	Mumbai	Engineering
2	Prayag	33	Kolkata	
3	Rohit	34	Chennai	
4	Rishi	34	Mumbai	
5	Kumar	35	Bengaluru	
6	Rashi	34	Bangalore	

A red horizontal line highlights the last two rows (Kumar and Rashi). The status bar at the bottom shows "OK." and playback controls.

X RIGHT OUTER JOIN Y MEANS Y IS FULLY DISPLAYED BUT X VALUES THAT ARE NOT COMMON TO ANY ROW OF Y IS DISPLAYED AS NULL.

SQL Editor Graphical Query Builder

Previous queries

```
select name,age,address,dept from company right outer join department on company.id=department.id
```

Output pane

	name text	age integer	address character(50)	dept character(50)
1				IT Billing
2	Heena	16	Mumbai	Engineering
3				Finance

FULL OUTER JOIN HAS ALL VALUES FROM TABLE X AND Y AND EVEN FOR THOSE VALUES WHERE X OR Y IS NULL OUTPUT IS DISPLAYED.

SQL Editor Graphical Query Builder

Previous queries

```
select name , salary from company full outer join department on company.id=department.id
```

Output pane

Data Output Explain Messages History

	name text	salary real
1		
2	Heena	3000
3		
4	Prayag	9000
5	Rohit	4000
6	Rishi	8000

OK. |◀|▶|▶| 2:18 / 3:52

SQL Editor Graphical Query Builder

Previous queries

```
select name , salary,dept from company full outer join department on company.id=department.id
```

Output pane

Data Output Explain Messages History

	name text	salary real	dept character(50)
1			IT Billing
2	Heena	3000	Engineering
3			Finance
4	Prayag	9000	
5	Rohit	4000	
6	Rishi	8000	

OK. |◀|▶|▶| 3:34 / 3:52

SQL Editor | Graphical Query Builder

Previous queries

```
select name,age from company c , department d  
where c.id=d.id;
```

<

Output pane

Data Output Explain Messages History

	name	age
1	Heena	16

SQL Editor Graphical Query Builder

Previous queries

```
select c.name,c.age from company c , department d  
where c.id=d.id;
```

<

Output pane

Data Output Explain Messages History

	name text	age integer
1	Heena	16

VIEWS ARE PSEUDO TABLES I.E NOT REAL TABLE OR PART OF TABLE . IT IS SUBSET OF TABLE AS IT SELECTS SOME COLUMNS AND ROWS FROM THE TABLE.

SQL Editor Graphical Query Builder

Previous queries

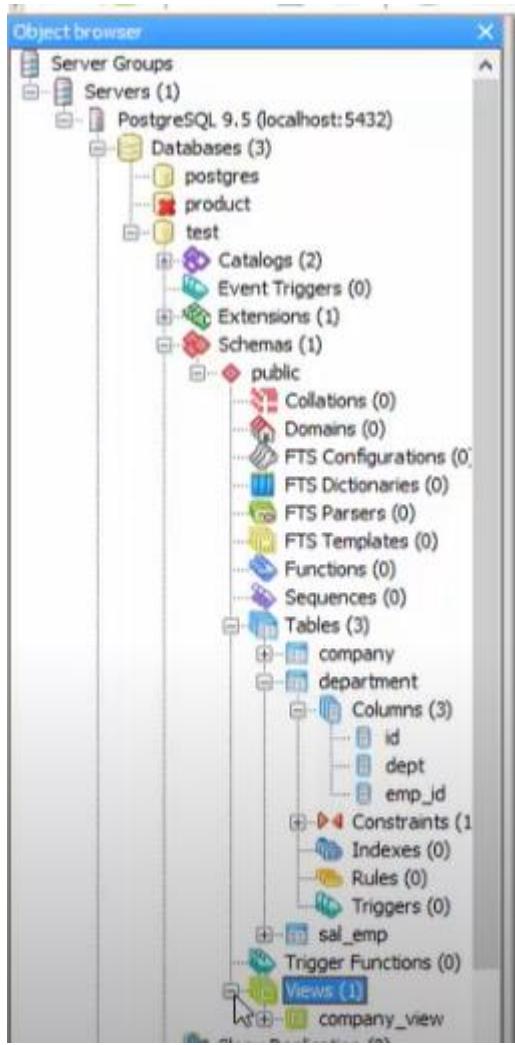
```
create view company_view as
select id,name,age from company;
```

<

Output pane

Data Output Explain **Messages** History

Query returned successfully with no result in 313 msec.



Edit Data - PostgreSQL 9.5 (localhost:5432) - test - public.company\_view

File Edit View Tools Help

No limit

	<b>id</b> integer	<b>name</b> text	<b>age</b> integer
<b>1</b>	2	Heena	16
<b>2</b>	5	Rashi	34
<b>3</b>	9	Kumar	40
<b>4</b>	4	Priya	24
<b>5</b>	7	Rahul	20
<b>6</b>	11	Prayag	33
<b>7</b>	13	Kumar	35
<b>8</b>	10	Rishi	34
<b>9</b>	12	Rohit	34
<b>10</b>	14	Pavan	45

IN VIEWS WE CAN'T UPDATE, DELETE OR MODIFY DATA.

UNION OR UNION ALL IS USED TO COMBINE 2 SELECT STATEMENTS.

IN UNION IF WE HAVE ANY DUPLICATE ROWS THEN IT CAN BE COMBINED INTO SINGLE ROW.

IN UNION ALL WE CAN SEE DUPLICATE ROWS AND THEY ARE PART OF RESULT.

SQL Editor | Graphical Query Builder

Previous queries

```
select company.id, name, department from company inner join department on company.id=department.id
```

Output pane

Data Output Explain Messages History

	<b>id</b> integer	<b>name</b> text	<b>department</b> character(100)
1	2	Heena	IT Billing

SQL Editor | Graphical Query Builder

Previous queries

```
select company.id, name, department from company inner join department on company.id=department.emp_id
```

Output pane

Data Output Explain Messages History

	<b>id</b> integer	<b>name</b> text	<b>department</b> character(100)
1	2	Heena	IT Billing
2	7	Rahul	Finance

SQL Editor Graphical Query Builder

Previous queries

```
select company.id, name, department from company left outer join department on company.id=department.emp_id
```

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>department</b> <b>character(100)</b>
3	11	Prayag	IT Billing
4	12	Rohit	Finance
5	10	Rishi	IT Billing
6	13	Kumar	IT Billing
7	5	Rashi	IT Billing
8	4	Priya	Finance

OK. |◀▶| 4:15 / 6:01 DOS Ln 1, Co

SQL Editor Graphical Query Builder

Previous queries

```
select company.id, name, department from company inner join department on company.id=department.emp_id  
union  
select company.id, name, department from company left outer join department on company.id=department.emp_id
```

Output pane

Data Output Explain Messages History

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>department</b> <b>character(100)</b>
1	9	Kumar	Finance
2	7	Rahul	Finance
3	10	Rishi	IT Billing
4	5	Rashi	IT Billing
5	2	Heena	IT Billing
6	4	Priya	Finance

OK. |◀▶| 5:07 / 6:01

THE ABOVE QUERY IS FOR COMBINING OUTPUTS DUE TO INNER JOIN AND LEFT OUTER JOIN.

The screenshot shows a SQL Editor interface with two panes. The top pane, titled 'SQL Editor' and 'Graphical Query Builder', contains the following SQL code:

```
select company.id, name, department from company inner join department on company.id=department.emp_id
union all
select company.id, name, department from company left outer join department on company.id=department.emp_id
```

The bottom pane, titled 'Output pane', displays the results of the queries. It has tabs for 'Data Output', 'Explain', 'Messages', and 'History'. The 'Data Output' tab shows a table with columns: id (integer), name (text), and department (character(100)). The data is as follows:

	id integer	name text	department character(100)
1	2	Heena	IT Billing
2	7	Rahul	Finance
3	2	Heena	IT Billing
4	7	Rahul	Finance
5	11	Prayag	IT Billing
6	12	Rohit	Finance

The rows for Heena and Rahul are highlighted with blue and red respectively, demonstrating the effect of UNION ALL.

UNION ALL SHOWS DUPLICATE ROWS ALSO. EG IN ABOVE TABLE RAHUL,HEENA ROWS ARE DUPLICATED.

UNION DOES NOT HAVE DUPLICATE ROWS BUT UNION ALL HAS DUPLICATE ROWS.

TRUNCATE IS USED TO DELETE ALL DATA INSIDE ROWS AND COLUMNS OF TABLE BUT STRUCTURE OF TABLE REMAIN AS IT IS.

DROP TABLE -> USED TO DELETE TABLE STRUCTURE AS WELL AS DATA.

The screenshot shows a PostgreSQL data editor window titled 'Edit Data - PostgreSQL 9.5 (localhost:5432) - test - public.sal\_emp'. The window has a menu bar with File, Edit, View, Tools, and Help. Below the menu is a toolbar with various icons. A dropdown menu shows 'No limit'. The main area is a table with three columns: name (text), pay (integer[]), and schedule (text[]). The data is as follows:

	name text	pay integer[]	schedule text[]
1	Rahul	{25000,25000,25000,25000}	{(meeting,lunch),(training,presentation)}
2	Rishi	{10000,30000,30000,23000}	{(meeting,dinner),(walk,presentation)}

SQL Editor | Graphical Query Builder

Previous queries

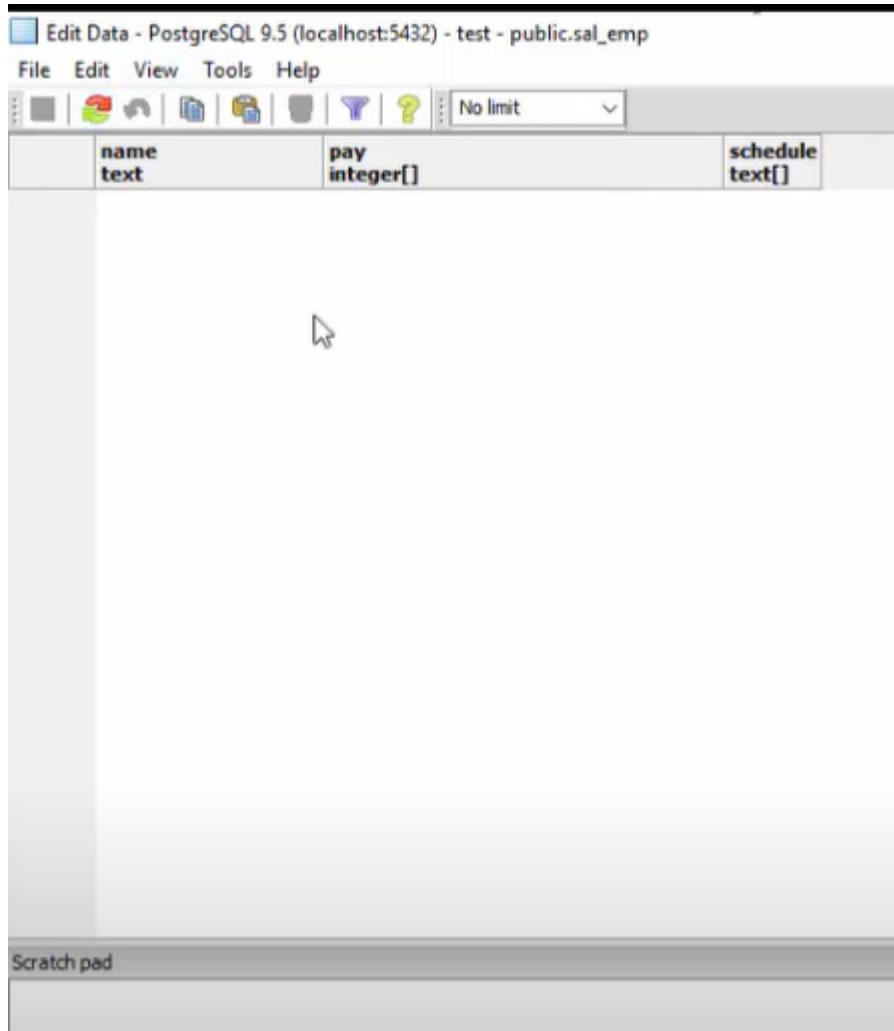
```
truncate table sal_emp
```

<

Output pane

Data Output Explain **Messages** History

Query returned successfully With no result in 170 msec.



TRUNCATED CASCADED MEANS IF ANY FOREIGN KEY ASSOCIATED WITH PARENT TABLE THEN IF PARENT TABLE IS DELETED THEN FOREIGN KEY ASSOCIATED IS ALSO DELETED.

SQL Editor Graphical Query Builder

Previous queries

```
select * from company where age in(select age from company where salary>4000)
```

Output pane

	<b>id</b> <b>integer</b>	<b>name</b> <b>text</b>	<b>age</b> <b>integer</b>	<b>address</b> <b>character(50)</b>	<b>salary</b> <b>real</b>	<b>department</b> <b>character(100)</b>
1	4	Priya	24	Delhi	3000	Finance
2	9	Kumar	40	Delhi	2000	Finance
3	12	Rohit	34	Chennai	4000	Finance
4	14	Pavan	45	Delhi	5000	Finance
5	7	Rahul	20	Kolkata	5000	Finance
6	5	Rashi	34	Bangalore	4000	IT Billing

OK. |◀|▶|▶| 3:50 / 8:10

USED SUBQUERIES IN THE ABOVE.

WE CAN ALSO SEE PEOPLE WITH < 4000 SALARY. this is due to the subquery , and the values of the ages that are being returned , for example in the first query there maybe a person aged 25 who has a salary above 4000 , hence 25 is included in the list , while in the outer query all people are selected whose age is 25. Hence there is no restriction of the salary in the outer query.

The screenshot shows a SQL Editor interface with two panes. The top pane is labeled 'SQL Editor' and contains the following SQL code:

```
update company set salary=salary*1.50 where age in(select age from company where age>=27);
```

The bottom pane is labeled 'Output pane' and contains the following message:

```
Query returned successfully: 75 rows affected, 107 msec execution time.
```

The screenshot shows a SQL Editor interface with two panes. The top pane is labeled 'SQL Editor' and contains the following SQL code:

```
delete from company where age in(select age from company where age>50)
```

#### SOME POSTGRE SQL IMP COMMANDS REVISED—

```
SELECT * FROM employees;
```

```
SELECT * FROM employees WHERE department = 'Sports';
```

```
##below finding total salary for each department.
```

```
SELECT SUM(salary) as total_salary, department FROM employees GROUP BY department;
```

```
##below finding departments having > 50 employees.
```

```
SELECT count(*) as total_employee,department FROM employees GROUP BY department HAVING COUNT(*) > 50;
```

# Find total salary by each department and sort **descending** by total salary column.

```
SELECT SUM(salary) as total_salary,department FROM employees  
GROUP BY department  
ORDER BY total_salary desc
```

# Write a query that finds the first 5 employees with their first\_name, department and salary and sorted by their first\_name.

```
SELECT first_name, department, salary from employees  
ORDER BY first_name  
LIMIT 5
```

*Note: Default Order By clause sorts the result in ASCENDING order.*

## 5. Date Functions

In PostgreSQL, you can easily extract values from date columns. You will see the most used date functions below.

```
SELECT  
date_part('year',hire_date) as year,  
date_part('month',hire_date) as month,  
date_part('day',hire_date) as day,  
date_part('dow',hire_date) as dayofweek,  
to_char(hire_date, 'Dy') as day_name,  
to_char(hire_date,'Month') as month_name,  
hire_date  
FROM employees
```

year double precision	month double precision	day double precision	dayofweek double precision	day_name text	month_name text	hire_date date
2006	4	20	4	Thu	April	2006-04-20
2009	1	26	1	Mon	January	2009-01-26
2010	5	17	1	Mon	May	2010-05-17
2014	8	2	6	Sat	August	2014-08-02
2003	1	14	2	Tue	January	2003-01-14
2003	6	8	0	Sun	June	2003-06-08

Date Functions Output

## Inner, Left or Right Joins

**Inner Join** clause creates a new table (not physical) by combining rows that have matching values in two or more tables.

**Example:** Query all employee information and their divisions of the department.

```
SELECT * FROM employees e  
INNER JOIN departments d  
ON e.department = d.department
```

employee_id	first_name	last_name	email	hire_date	department	gender	salary	region_id	department	division
	character varying(50)	character varying(50)	character varying(50)	date	character varying(17)	character varying(1)	integer		character varying(100)	character varying(100)
1	Berrie	Manuska	bmanuska@odion.co.jp	2006-08-20	Sports	F	15866	4	Sports	Outdoors
2	Aerial	McNee	amcnee@google.es	2009-12-16	Tools	F	56752	3	Tools	Hardware
3	Sydney	Symonds	ssymonds@hrs.gov	2010-05-17	Clothing	F	95312	4	Clothing	Home
4	Avrom	Rowntree	irnull	2014-08-02	Phones & Tablets	M	119574	7	Phones & Tablets	Electronics
5	Feliks	Moffett	fmoffett@q43.net	2003-11-14	Computers	M	35307	5	Computers	Electronics
6	Bethwe	Trow	btrow5@charonweli.com	2009-06-08	Sports	F	134501	3	Sports	Outdoors
7	Aldan	Curwood	acurwood@lundt.de	2002-02-19	Clothing	F	28905	7	Clothing	Home
8	Seline	Dubber	sdubber7@t-online.de	2012-05-28	Phones & Tablets	F	101066	3	Phones & Tablets	Electronics
9	Dayle	Trall	dtrall@emu.edu	2003-03-01	First Aid	F	82759	1	First Aid	Health
10	Pedrof	Roberti	irnull	2008-07-21	Clothing	M	72225	7	Clothing	Home

The **Left Join** returns all rows from the left table and the matching rows from the right table. If no matching rows are found in the right table, **NULL** is used. (vice versa for Right Join)

employee_id	first_name	last_name	email	hire_date	department	gender	salary	region_id	department	division
integer	character varying (50)	character varying (50)	character varying (50)	date	character varying (17)	character varying (1)	integer	integer	character varying (100)	character varying (100)
1	Berrie	Maneuve	bmaneuve@odion.ne.jp	2006-04-20	Sports	F	154864	4	Sports	Outdoors
2	Aerial	McNee	amcnnee@google.es	2009-01-26	Tools	F	567532	3	Tools	Hardware
3	Sydney	Symonds	ssymonds2@hna.gov	2010-05-17	Clothing	F	95313	4	Clothing	Home
4	Avrom	Rowntree	[null]	2014-08-02	Phones & Tablets	M	119674	7	Phones & Tablets	Electronics
5	Feikus	Moffew	fmoffew4@a8.net	2003-01-14	Computers	M	55307	5	Computers	Electronics
6	Bethena	Trow	btrow8@technorati.com	2003-06-08	Sports	F	134501	3	Sports	Outdoors
7	Arden	Curwood	acurwood8@lundi.net	2006-02-19	Clothing	F	28995	7	Clothing	Home
8	Seline	Dubber	sdubber7@t-online.de	2012-05-28	Phones & Tablets	F	101066	3	Phones & Tablets	Electronics
9	Dayle	Trail	dtrail8@stamu.edu	2003-03-01	First Aid	F	82753	1	First Aid	Health
10	Redford	Robert	[null]	2008-07-21	Clothing	M	72225	7	Clothing	Home

1 IS EMPLOYEE TABLE AND 2 IS DEPARTMENT TABLE.

**Example:** Write a query that prints all departments from employees and matches departments from the department table.

```
SELECT e.department,d.department FROM employees e  
LEFT JOIN departments d  
ON e.department = d.department
```

department character varying (17)	department character varying (100)
Sports	Sports
Automotive	Automotive
Camping	[null]
Clothing	Clothing
Computers	Computers
Tools	Tools

**Example:** Query first\_name, department, and salary of each employee and also maximum salary given.

```
SELECT first_name,department,salary, (SELECT max(salary)  FROM  
employees)  
FROM employees
```

first_name character varying (50)	department character varying (17)	salary integer	max integer
Berrie	Sports	154864	166976
Aeriell	Tools	56752	166976
Sydney	Clothing	95313	166976
Avrom	Phones & Tablets	119674	166976
Feliks	Computers	55307	166976
Bethena	Sports	134501	166976

QUE

**Example:** Write a query that finds the first name, salary, department, and average salary by department.

```
SELECT first_name,salary,department,round((SELECT AVG(salary)
    FROM employees e2
    WHERE e1.department = e2.department
    GROUP BY department )) as avg_salary_by_department
FROM employees e1
WHERE salary > (SELECT AVG(salary)
    FROM employees e2
    WHERE e1.department = e2.department
    GROUP BY department )
ORDER BY salary
```

first_name character varying (50)	salary integer	department character varying (17)	avg_salary_by_department numeric
Jobi	80331	Maintenance	76793
Dexter	80657	Jewelry	80576
Weylin	86194	Clothing	83933
Worth	86252	Tools	83530
Basile	86269	Jewelry	80576
Port	86352	Clothing	83933

**Example:** Write a query to print the first name, salary, and average salary as well as a new column that shows whether employees' salary is higher than average or not.

```
SELECT first_name, salary, (SELECT ROUND(AVG(salary)) FROM employees) as
average_salary,
(CASE WHEN salary > (SELECT AVG(salary) FROM employees) THEN
'higher_than_average'
ELSE 'lower_than_average' END) as Salary_Case
FROM employees
```

first_name	salary	average_salary	salary_case
Berrie	154864	91572	higher_than_average
Aeriell	56752	91572	lower_than_average
Sydney	95313	91572	higher_than_average
Avrom	119674	91572	higher_than_average
Feliks	55307	91572	lower_than_average
Bethena	134501	91572	higher_than_average

The following query will give you the average salary for each department.

```
SELECT first_name, salary, department,
ROUND(AVG(salary) OVER(PARTITION BY department)) as avg_sales_by_dept
FROM employees
ORDER BY salary DESC
```

first_name character varying (50)	salary integer	department character varying (17)	avg_sales_by_dept numeric
Jacklyn	166976	Clothing	83933
Carissa	166765	Music	92361
Riley	166569	Camping	97302
Lauren	166016	Pharmacy	93372
Lucy	165660	Sports	87832
Barby	164588	Clothing	83933

Aggregate Window Functions

## 10.2.Ranking the Values

The Rank() function is a window function that assigns a rank to each row within a partition of a result set.

The following example orders the table by the salary (descending). A rank value of 1 is the highest salary value.

```
SELECT first_name,salary,RANK() OVER(ORDER BY salary DESC)  
FROM employees
```

first_name character varying (50)	salary integer	rank bigint
Jacklyn	166976	1
Carissa	166765	2
Riley	166569	3
Lauren	166016	4
Lucy	165660	5
Barby	164588	6

## Understanding the Basics

The basics are more than just the basic commands, what matters most when answering difficult questions is the ***order of operations!*** Make sure you know this by heart. If you get in the habit of writing each line of a query in this order rather than from top to bottom I promise you'll be a SQL wizard in no time!

### 1. ***Order of Operations***

*FROM : the table your data is in*

*WHERE: filters the table down to the contents where a certain condition is met*

*GROUPBY: clusters the data into groups*

*SELECT: actually pulls the data*

*ORDER BY: orders the data*

*LIMIT: puts a cap on the number of returns*

## ***Query Order***

*SELECT*

*FROM*

*WHERE*

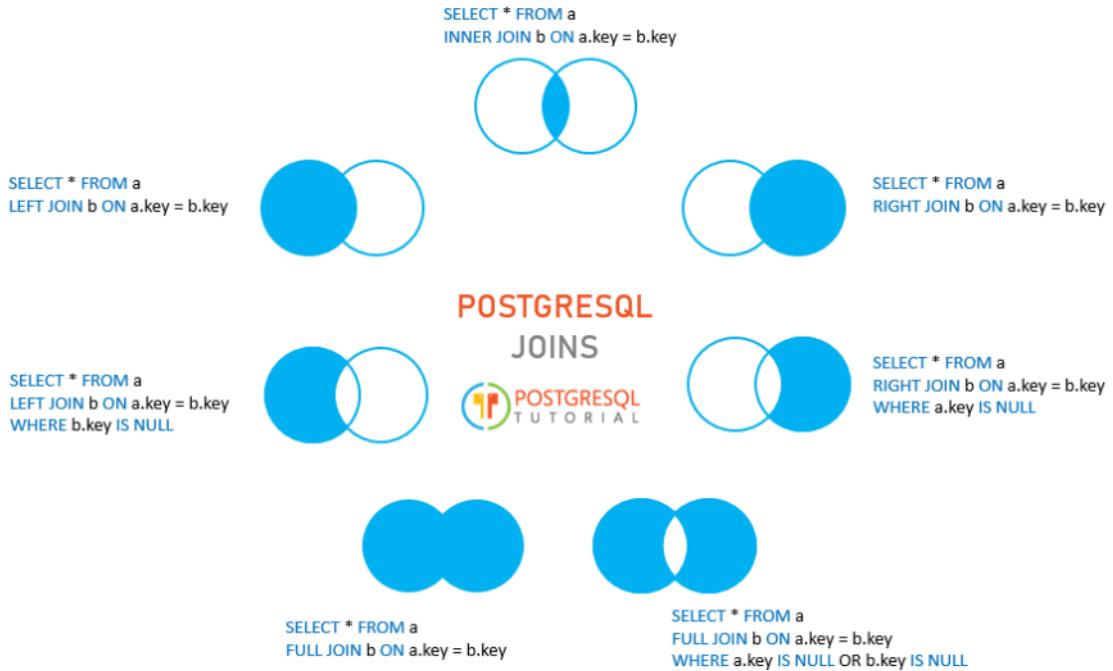
*GROUP BY*

*ORDER BY*

*LIMIT*

You might wonder why the code executes in one order but is written in another. Unfortunately, this is just the way it's designed.

## 2. Joins



Almost every SQL query you'll see in entry-level interviews will use inner joins — if you must guess, go for that.

### Easy Questions

You have a table called `foods` and it looks like this:

	<b>id</b>	<b>name</b>	<b>restaurant_id</b>	<b>calories</b>	<b>fat</b>	<b>carbs</b>	<b>created_at</b>	<b>updated_at</b>
0	26	Blue Raspberry Slushee	1.0	370.0	0.0	91.0	2017-04-17 19:43:29.627441	2017-04-17 19:43:29.627441
1	27	Blue Raspberry Slushee	1.0	570.0	0.0	142.0	2017-04-17 19:43:29.634462	2017-04-17 19:43:29.634462
2	28	Blue Raspberry Slushee	1.0	740.0	0.0	182.0	2017-04-17 19:43:29.641432	2017-04-17 19:43:29.641432
3	29	Breaded Onion Rings (Large)	1.0	480.0	27.0	62.0	2017-04-17 19:43:29.653546	2017-04-17 19:43:29.653546
4	30	Breaded Onion Rings (Regular)	1.0	350.0	16.0	45.0	2017-04-17 19:43:29.659697	2017-04-17 19:43:29.659697

**Q1.** Display the entire table.

**Q2.** Display the `name` (renamed "food") and `calories` from the `foods` schema.

Sort it by `calories` first and then by `name` in alphabetical order.

## Answers

### Q1.

```
SELECT *
FROM foods as f;
```

### Q2.

```
SELECT f.name as "food", f.fat
FROM foods f
ORDER BY f.calories DESC, f.name ASC;
```

#### **Medium Questions**

**Q3.** Display the `name`, `carbs`, and total `calories` of the Top 5 foods where carbs make up between 25% and 75% of the total calories from fat

**Q4.** You are told there is another table called *restaurants* that has a foreign key which links to the *foods* column named `id`. Name that column appropriately and join the tables.

### Q3.

```
SELECT f.name, f.carbs, f.calories
FROM foods as f
WHERE 100*f.carbs/(f.calories+0.001) BETWEEN 25 and 75
ORDER BY 100*f.carbs/(f.calories+0.001) DESC
LIMIT 5;
```

### Q4.

```

SELECT f.id
FROM foods as f
INNER JOIN restaurants as r
ON f.id=r.foods_id;

```

This question tests your knowledge of appropriate naming schemas, note that foreign keys should reference the table they are a key for!

### **Hard Questions**

Now you have a table called *salaries* that looks like this:

name	salary	date of hire
Amanda	50,000	05/20/18
Carlos	60,000	02/22/17
Ben	55,000	06/30/18

salaries

**Q5.** Return the second highest salary. (Hint ——— use a subquery)

**Q6 (from CodeWars).**

#	Pokémon	HP	Attack	Defense
493	Arceus	120	120	120
646	Kyurem (Black Kyurem)	125	170	100
646	Kyurem (White Kyurem)	125	120	90
249	Lugia	106	90	130
644	Zekrom	100	150	120
483	Dialga	100	120	120

644		Zekrom	100	150	120
483		Dialga	100	120	120

You have arrived at the Celadon Gym to battle Erika for the Rainbow Badge.

She will be using Grass-type Pokemon. Any fire pokemon you have will be strong against grass, but your water types will be weakened. The multipliers table within your Pokedex will take care of that.

Using the following tables, return the *pokemon\_name*, *modifiedStrength* and *element* of the Pokemon whose strength, after taking these changes into account, is greater than or equal to 40, ordered from strongest to weakest.

pokemon schema

- id
- pokemon\_name
- element\_id
- str

multipliers schema

- id
- element
- multiplier

## Q5.

```
SELECT Max(salary)
FROM salaries
WHERE salary NOT IN (SELECT Max(salary) FROM salaries);
```

## Q6.

```
SELECT pkmn.pokemon_name,
pkmn.str*mult.multiplier as modifiedStrength,
mult.element
FROM pokemon as pkmn
INNER JOIN multipliers as mult ON pkm.element_id = mult.id
WHERE pkm.str*mult.multiplier ≥ 40
ORDER BY modifiedStrength DESC
```