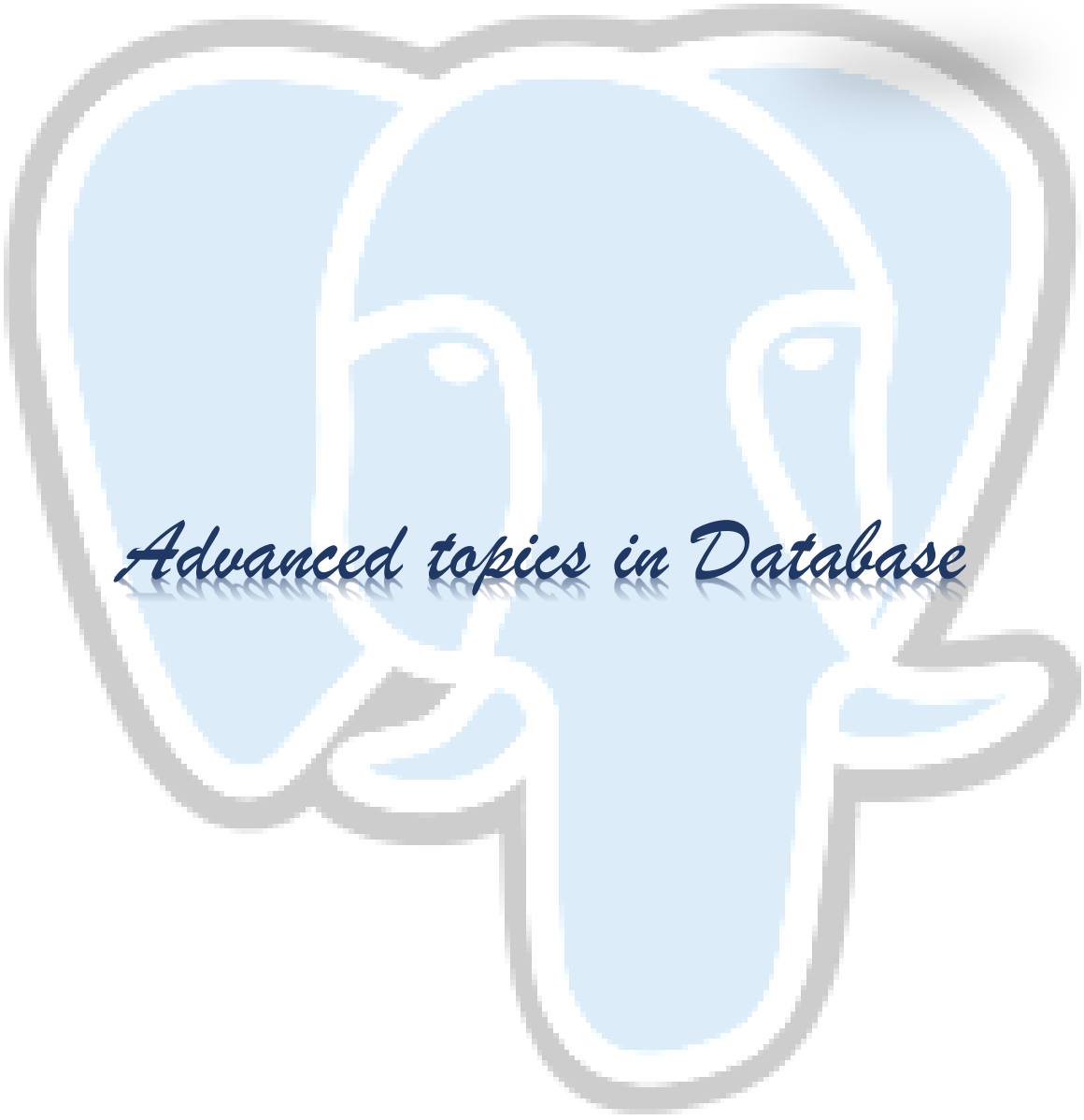
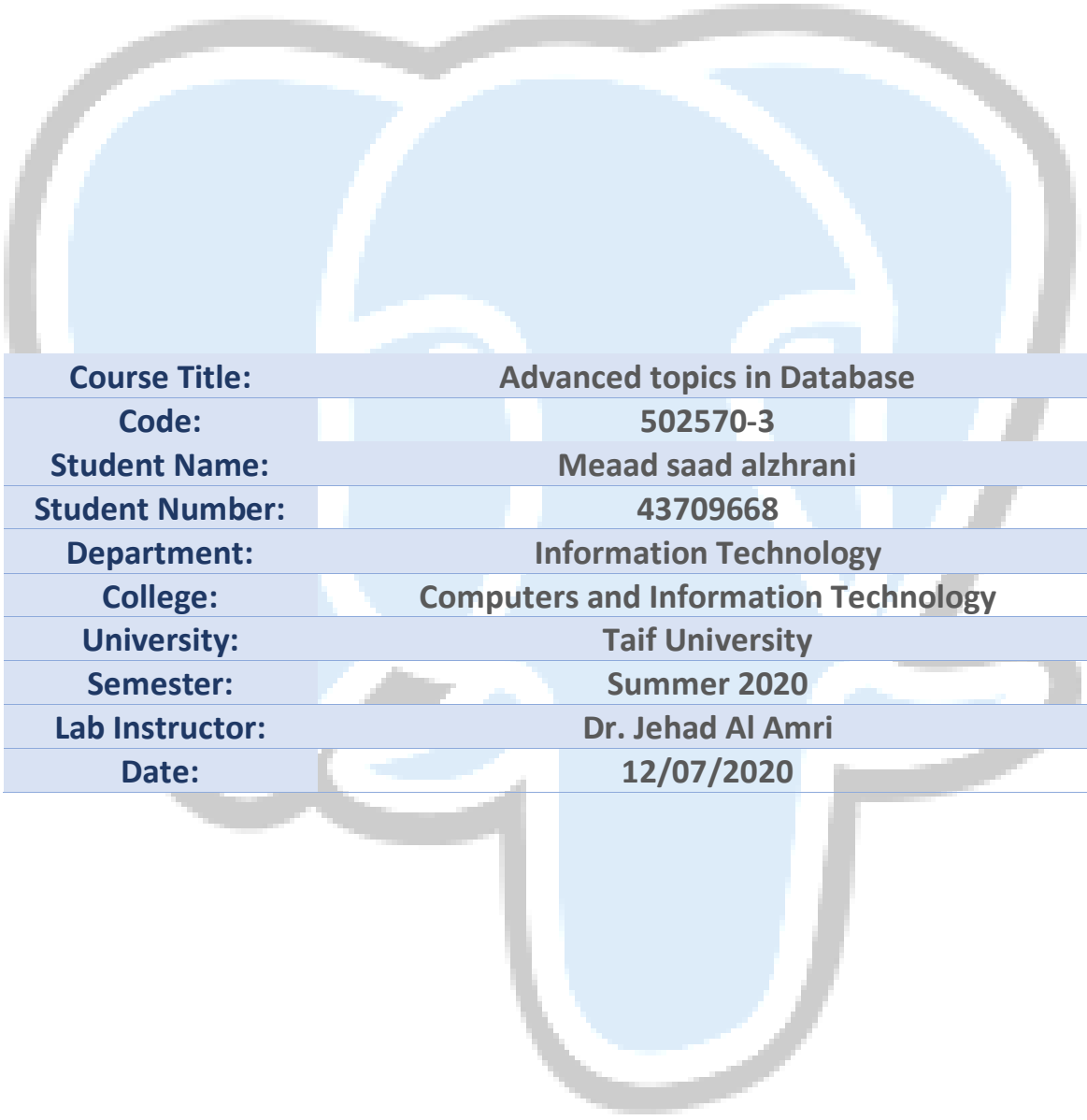


Advanced topics in Database



Course Title:	**Advanced topics in Database**
Code:	**502570-3**
Student Name:	**Meaad saad alzhrani**
Student Number:	**43709668**
Department:	**Information Technology**
College:	**Computers and Information Technology**
University:	**Taif University**
Semester:	**Summer 2020**
Lab Instructor:	**Dr. Jehad Al Amri**
Date:	**12/07/2020**

1.Introduction, Installation and connection with database of PostgreSQL ..

2.Customer Table with different Data in PostgreSQL …

3.Different types of fetching data in PostgreSQL ..

4.PostgreSQL (Boolean, Timestamp and others)..

5.PostgreSQL GROUP BY, HAVING, AGGEREGATE FUNCTIONS ..

6.Joins in PostgreSQL ..

7.PostgreSQL (UUID, hstore, json, Array)..

8.Conditional Expressions & Operators..

9.PostgreSQL Database Constraints ..

10.PostgreSQL NTILE Function, LAG Function, LEAD Function, NTH_VALUE Function, FIRST_VALUE Function, LAST_VALUE Function ..

11.Object Oriented Database in PostgreSQL ..

12.Object Oriented Querying (jOOQ) with PostgreSQL ..

13.Revision For PostgreSQL Advanced Object Oriented Queries

1:Introduction, Installation and connection with database of PostgreSQL

There are three steps to complete the PostgreSQL installation:

1. Download PostgreSQL installer for Windows
2. Install PostgreSQL
3. Verify the installation

Download PostgreSQL Installer for Windows

First, you need to go to the download page of PostgreSQL installers on the EnterpriseDB.

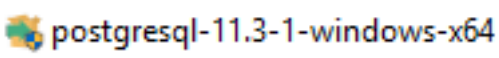
Second, click the download link as shown below:

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
11.3	N/A	N/A	Download	Download	N/A
10.8	Download	Download	Download	Download	Download
9.6.13	Download	Download	Download	Download	Download
9.5.17	Download	Download	Download	Download	Download
9.4.22	Download	Download	Download	Download	Download
9.3.25 (Not Supported)	Download	Download	Download	Download	Download

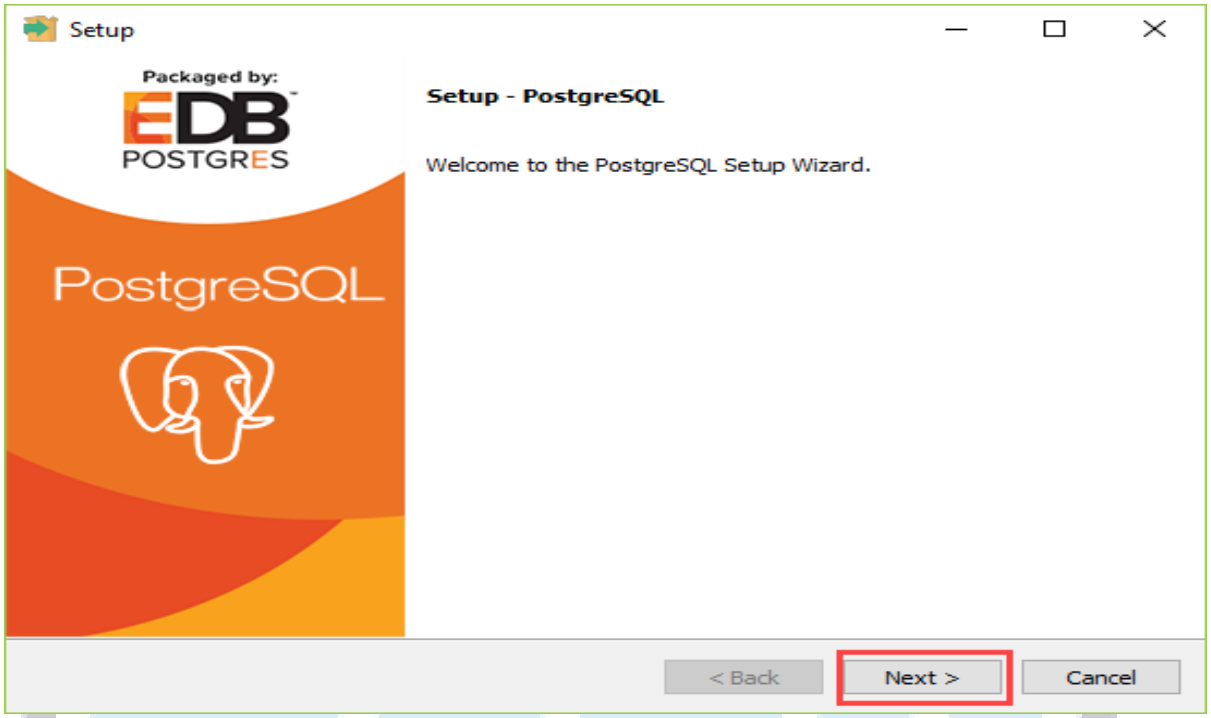
It will take a few minutes to complete the download.

Install PostgreSQL step by step

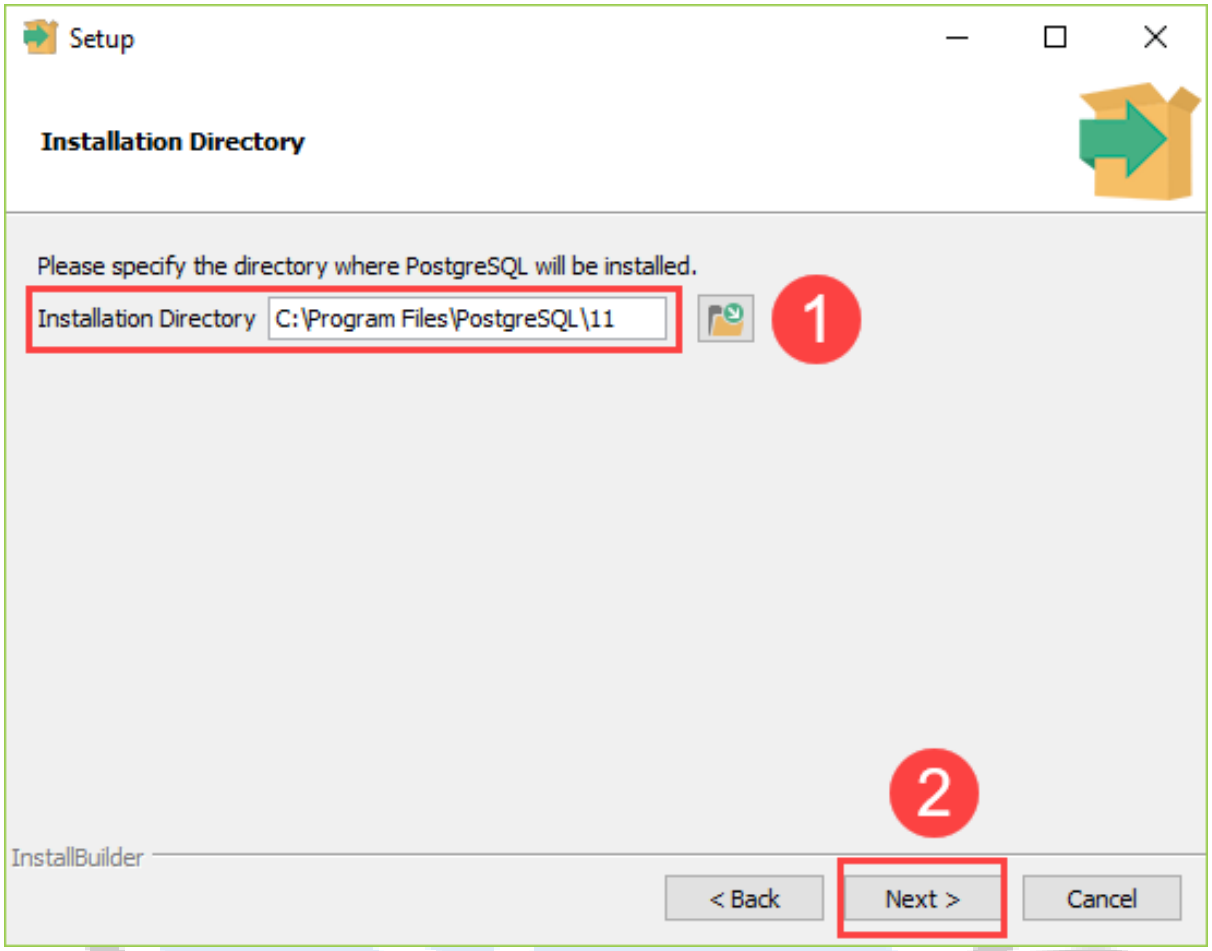
Step 1. Double click on the installer file, an installation wizard will appear and guide you through multiple steps where you can choose different options that you would like to have in PostgreSQL.



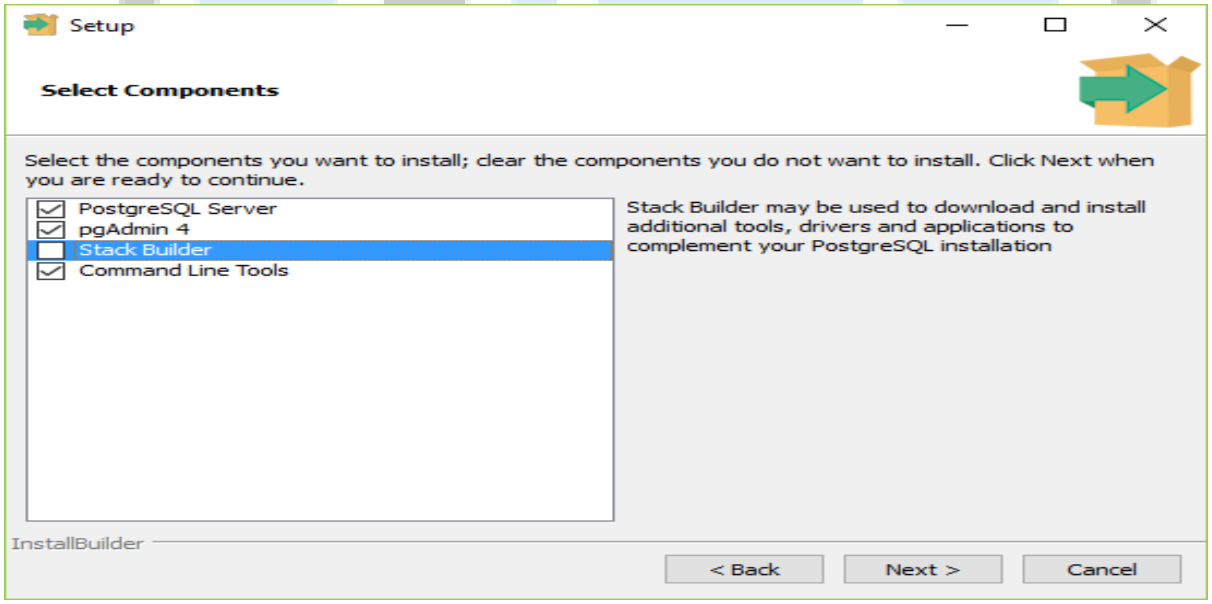
Step 2. Click the Next button



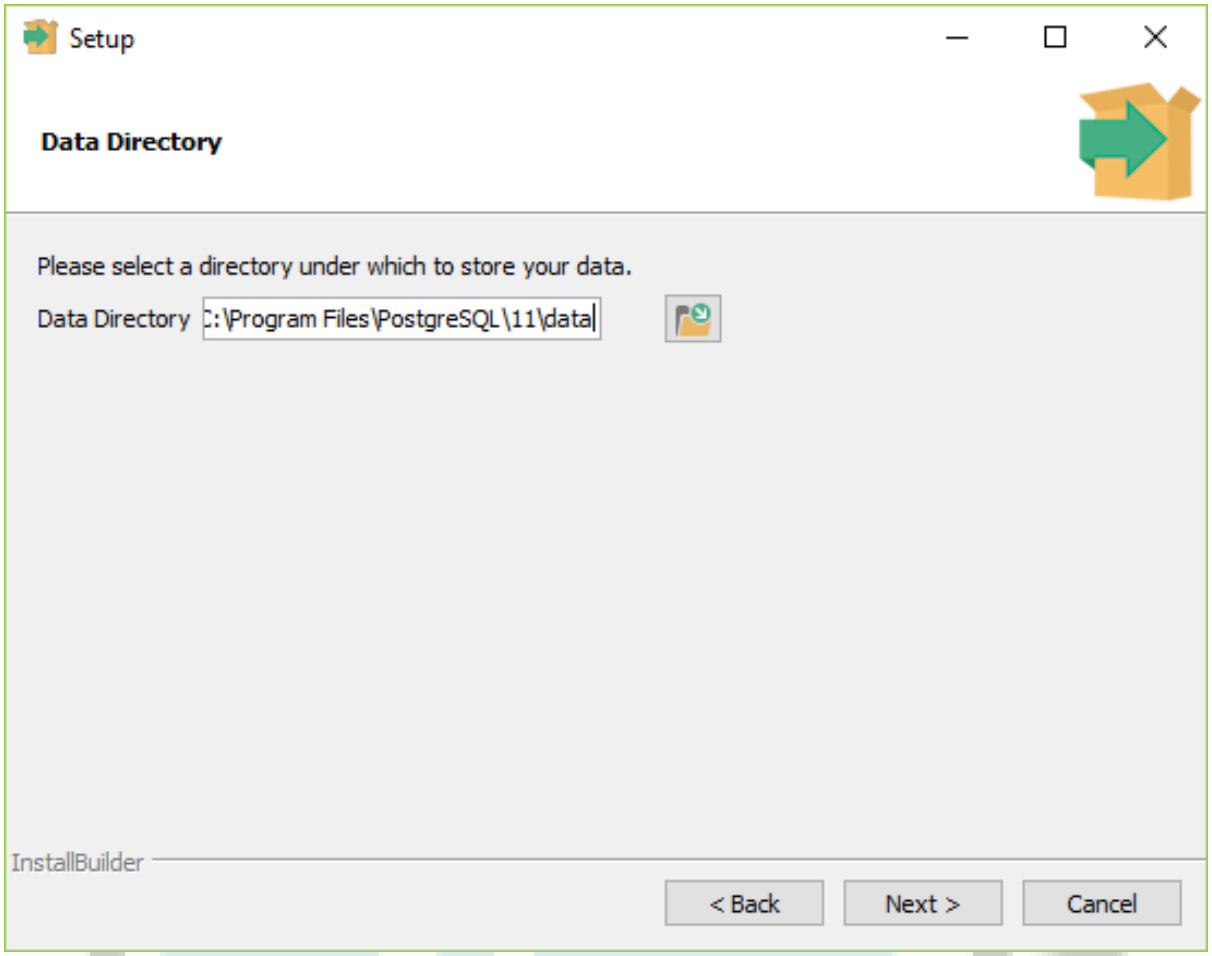
Step 3. Specify installation folder, choose your own or keep the default folder suggested by PostgreSQL installer and click the Next button



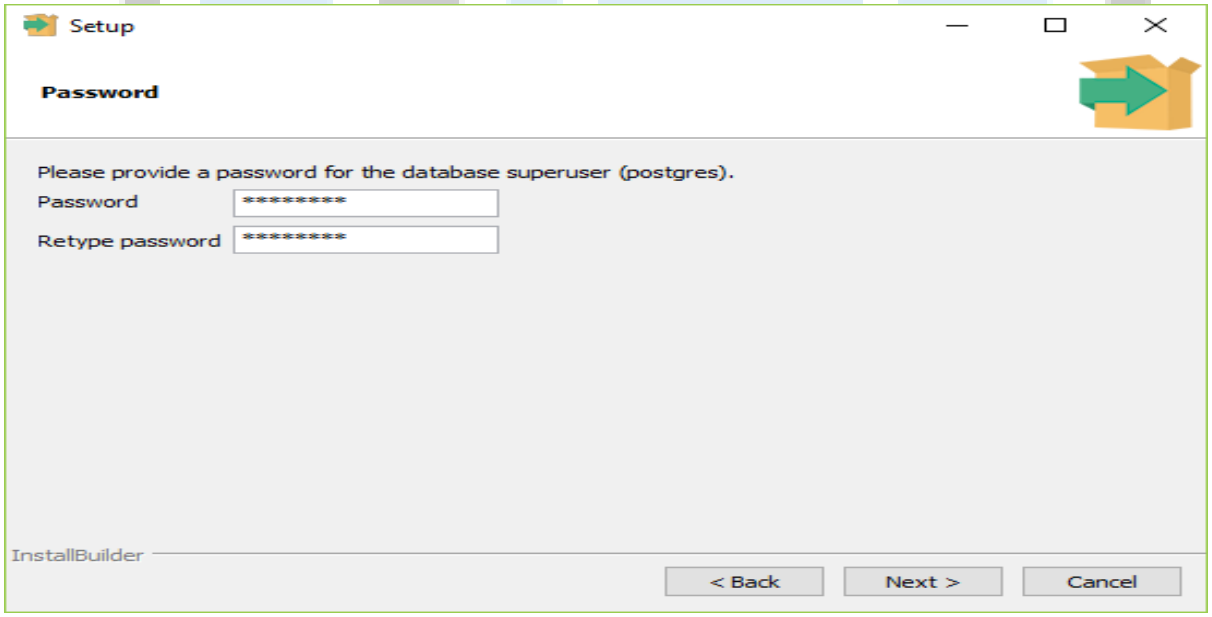
Step 4. Select components to install and click the Next button



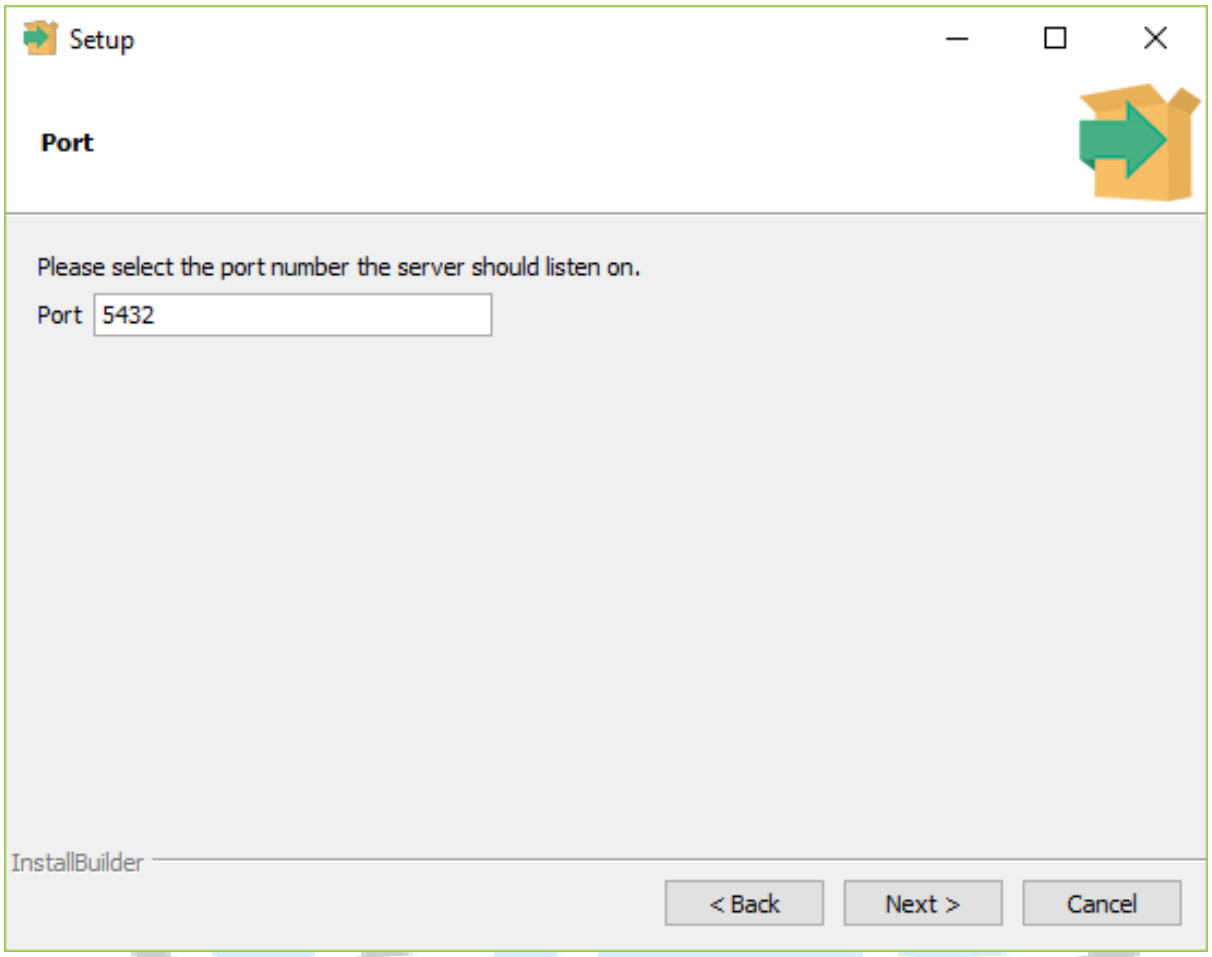
Step 5. Select the database directory to store the data. Just leave it by default or choose your own and click the Next button.



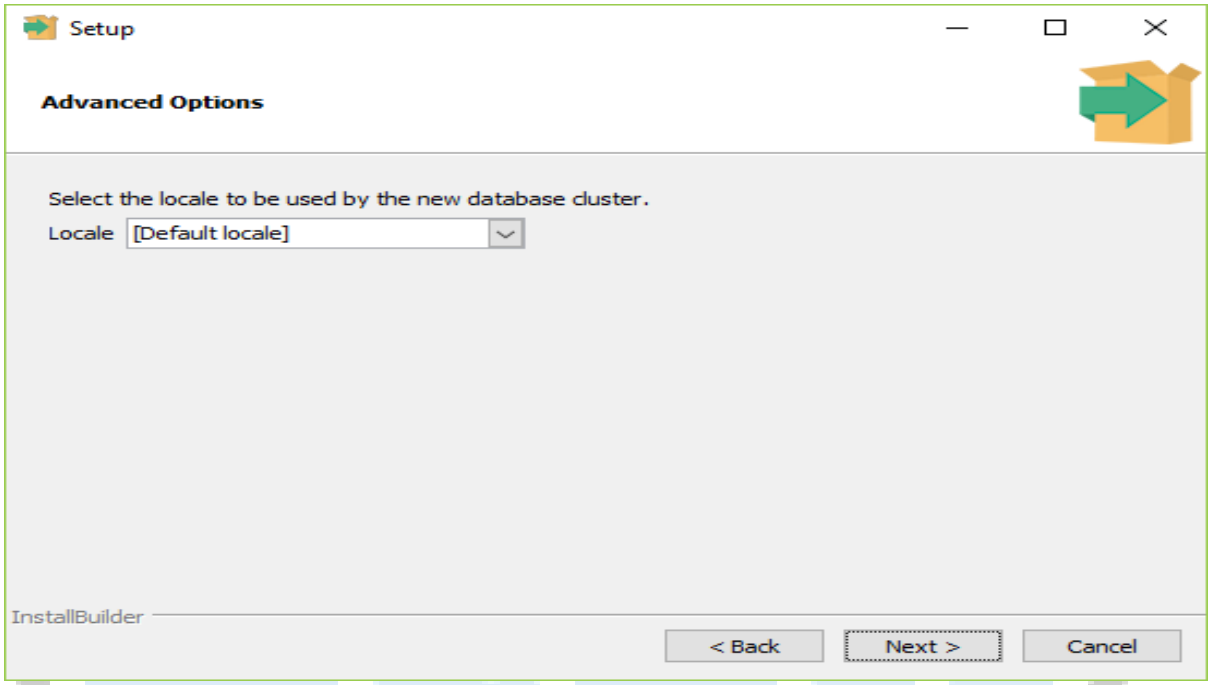
Step 6. Enter the password for the database superuser (postgres)



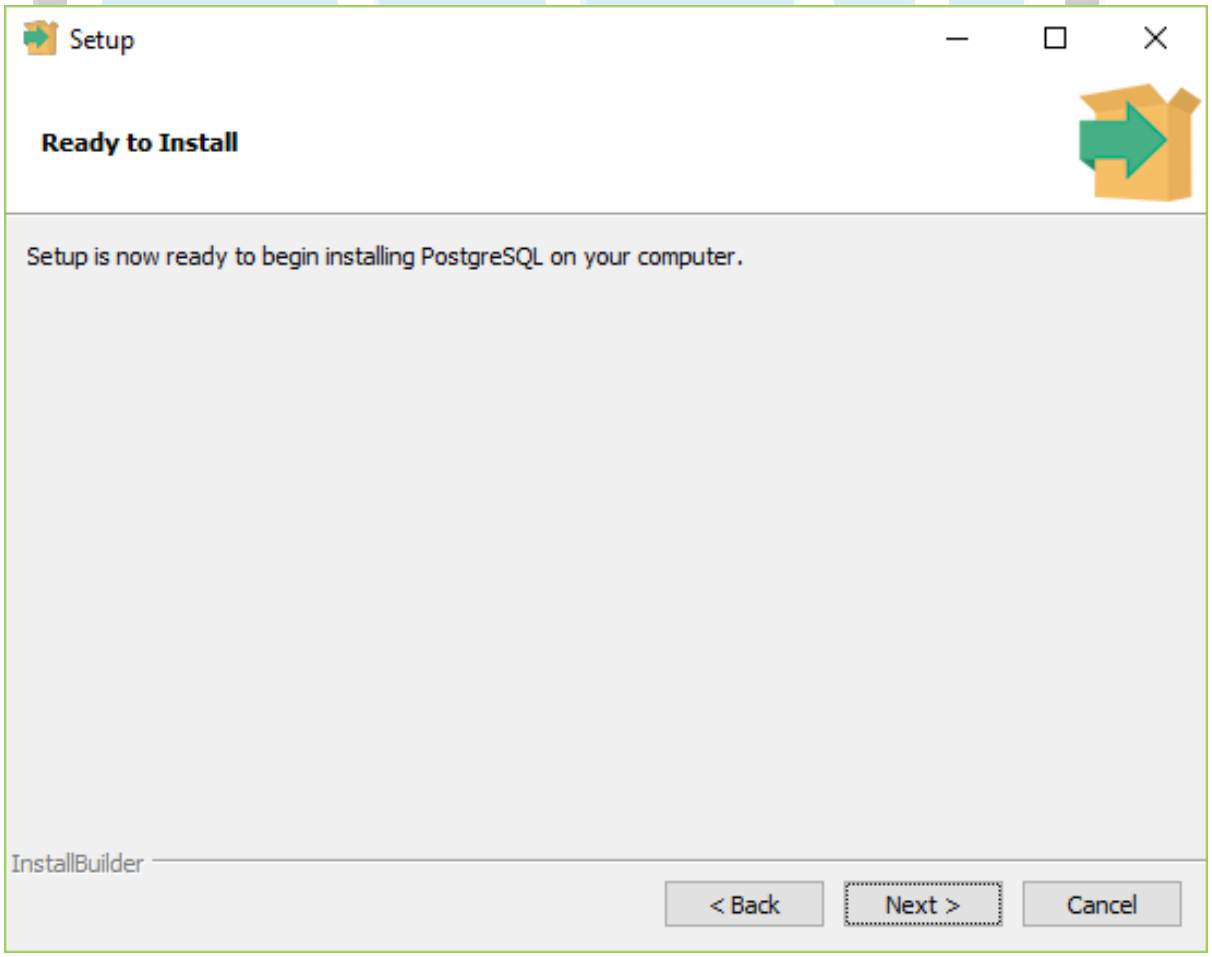
Step 7. Enter the port for PostgreSQL. Make sure that no other applications are using this port. Leave it as default if you are unsure.



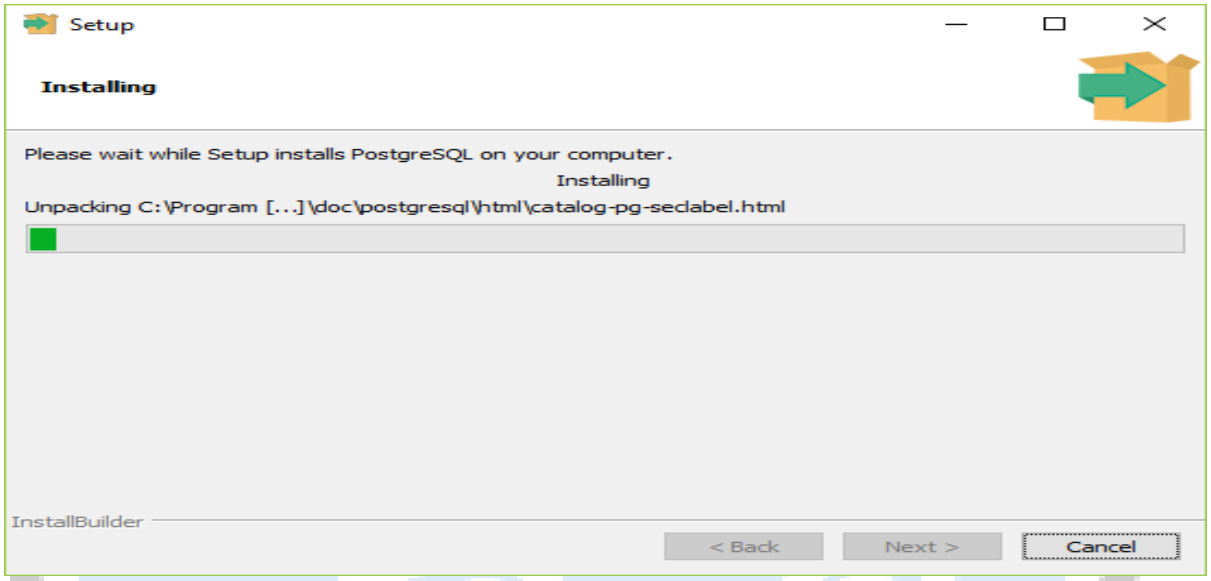
Step 8. Choose the default locale used by the database and click the Next button.



Step 9. Ready to install PostgreSQL. Click the **Next** button to start installing.



The installation may take a few minutes to complete.



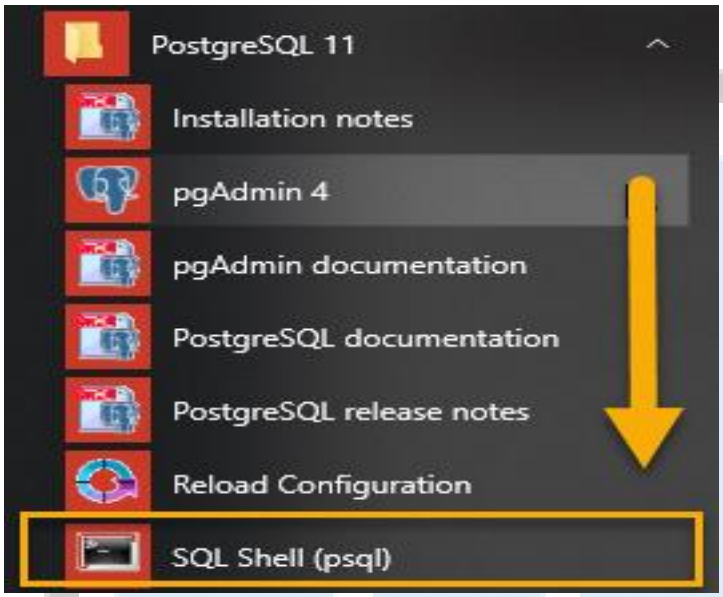
Step 10. Click the **Finish** button to complete the PostgreSQL installation.

Verify the Installation

There are several ways to verify the installation. You can try to connect to the PostgreSQL database server from any client application e.g., psql and pgAdmin.

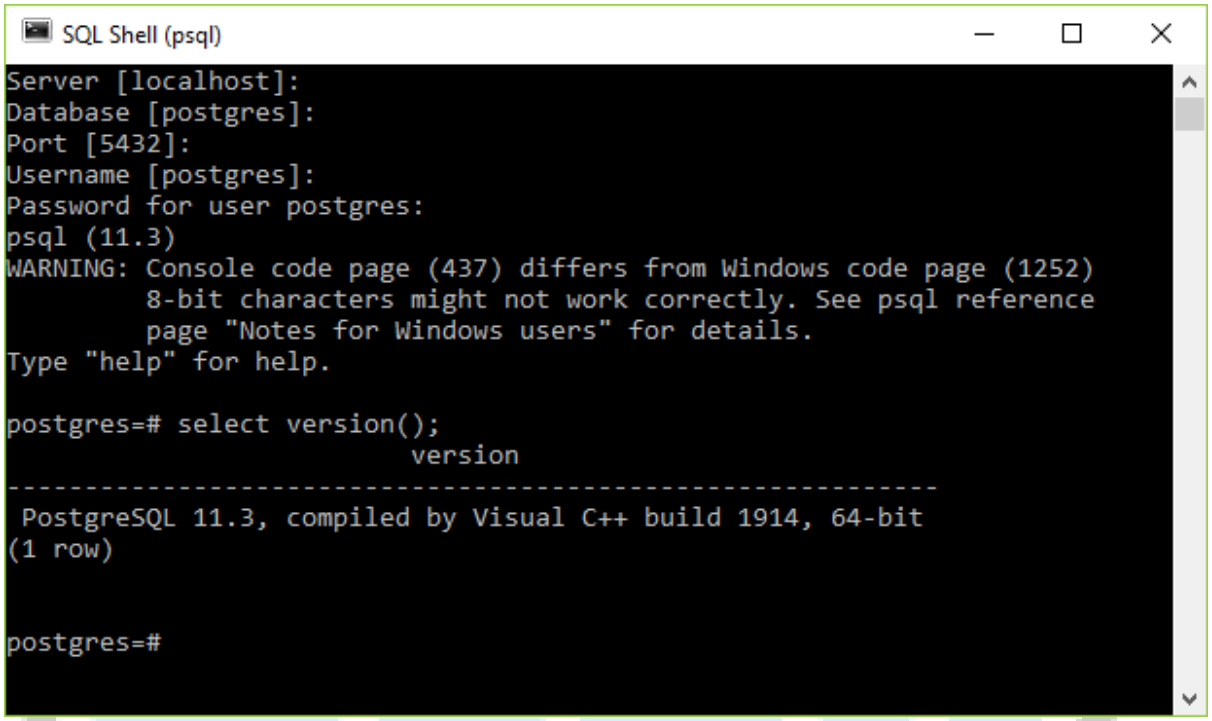
The quick way to verify the installation is through the psql program.

First, click the psql icon to launch it. The psql window command line will display.



Second, enter all the necessary information such as the server, database, port, username, and password. To accept the default, you can press **Enter**. Note that you should provide the password that you entered during installing the PostgreSQL.

Third, issue the command SELECT version(); you will see the result as follows:



Congratulation! you've successfully installed PostgreSQL database server on your local system. Let's learn various ways to connect to PostgreSQL database server.

3.Connect To a PostgreSQL Database Server

Summary: in this tutorial, you will learn how to **connect to the PostgreSQL Database Server** via an interactive terminal program called **psql** and via the **pgAdmin** application.

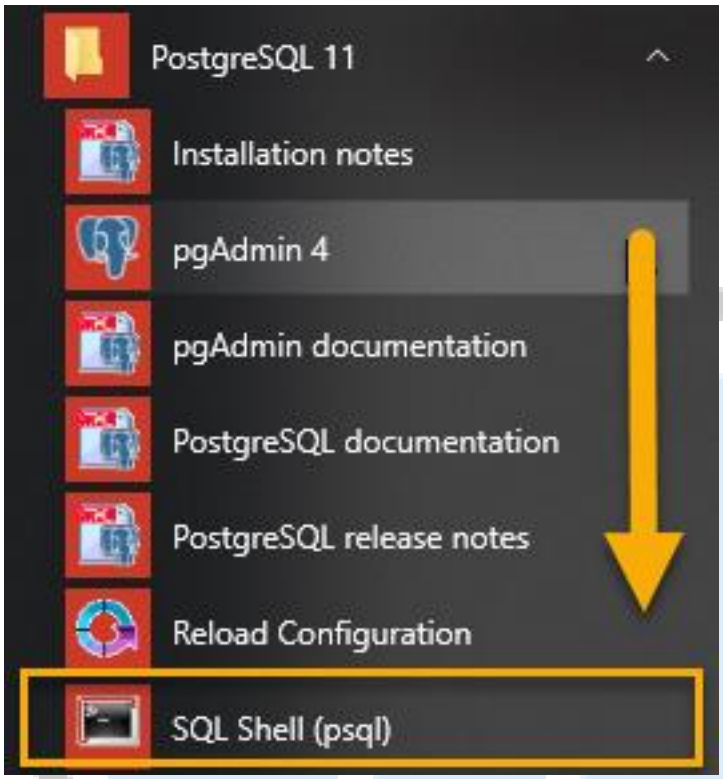
When you installed the PostgreSQL database server, the PostgreSQL installer also installed some useful tools for working with the PostgreSQL database server. You can connect to the PostgreSQL database server via the psql interactive program or pgAdmin tool.

Connect to PostgreSQL database server using psql

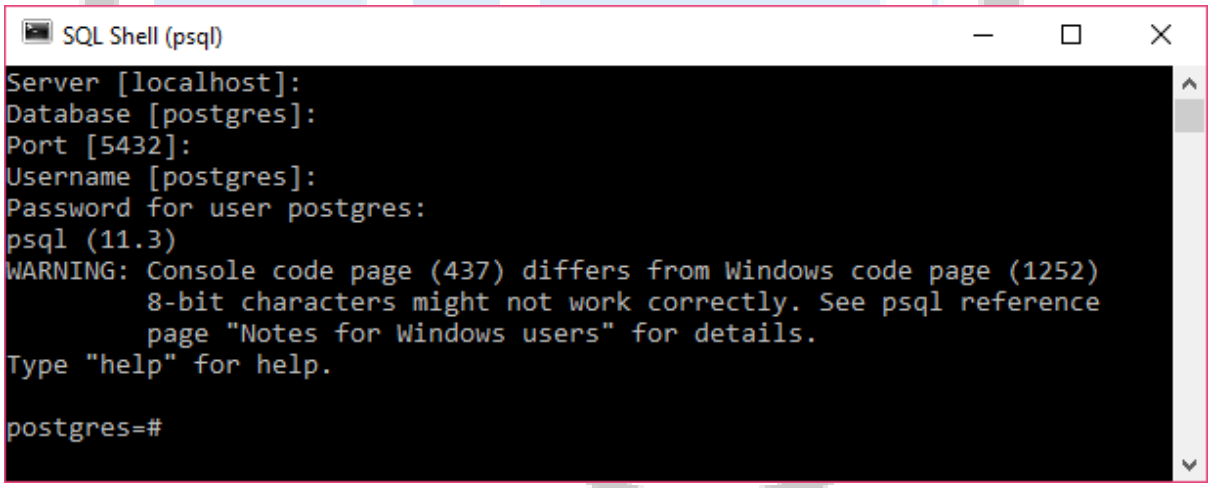
psql is an interactive terminal program provided by PostgreSQL. It allows you to interact with the PostgreSQL database server such as executing SQL statements and managing database objects.

The following steps show you how to connect to the PostgreSQL database server via the *psql* program:

First, launch **psql** program and connect to the PostgreSQL Database Server using the **postgres** user by clicking the psql icon as shown below:

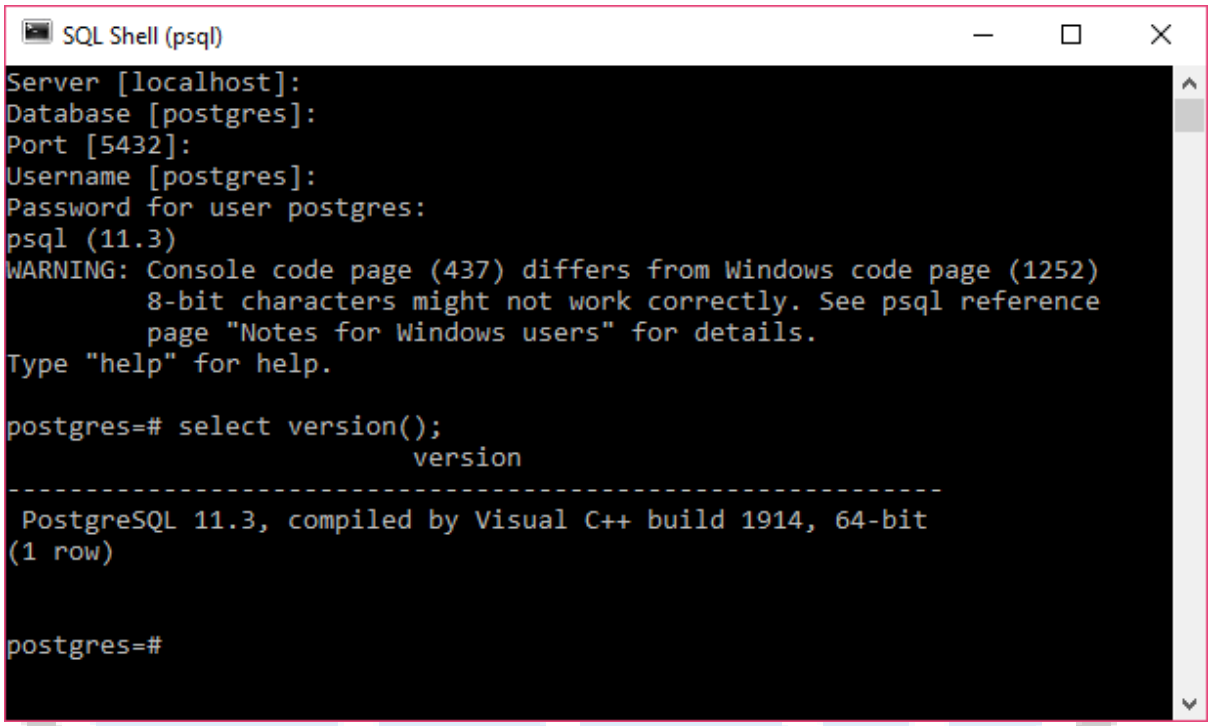


Second, enter the necessary information such as Server, Database, Port, Username, and Password. Press Enter to accept the default. However, you need to enter the password that you set up during the installation.



Third, interact with the PostgreSQL Database Server by issuing an SQL statement. You can try the following statement to test it out:

```
1 SELECT version();
```



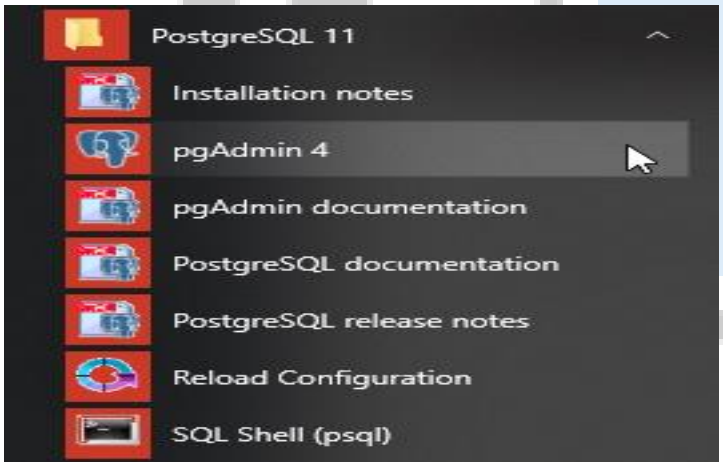
Please don't forget to end the command with the semicolon (;). After pressing **Enter**, psql will return the current PostgreSQL version that you have in the system.

Connect to PostgreSQL database server using pgAdmin

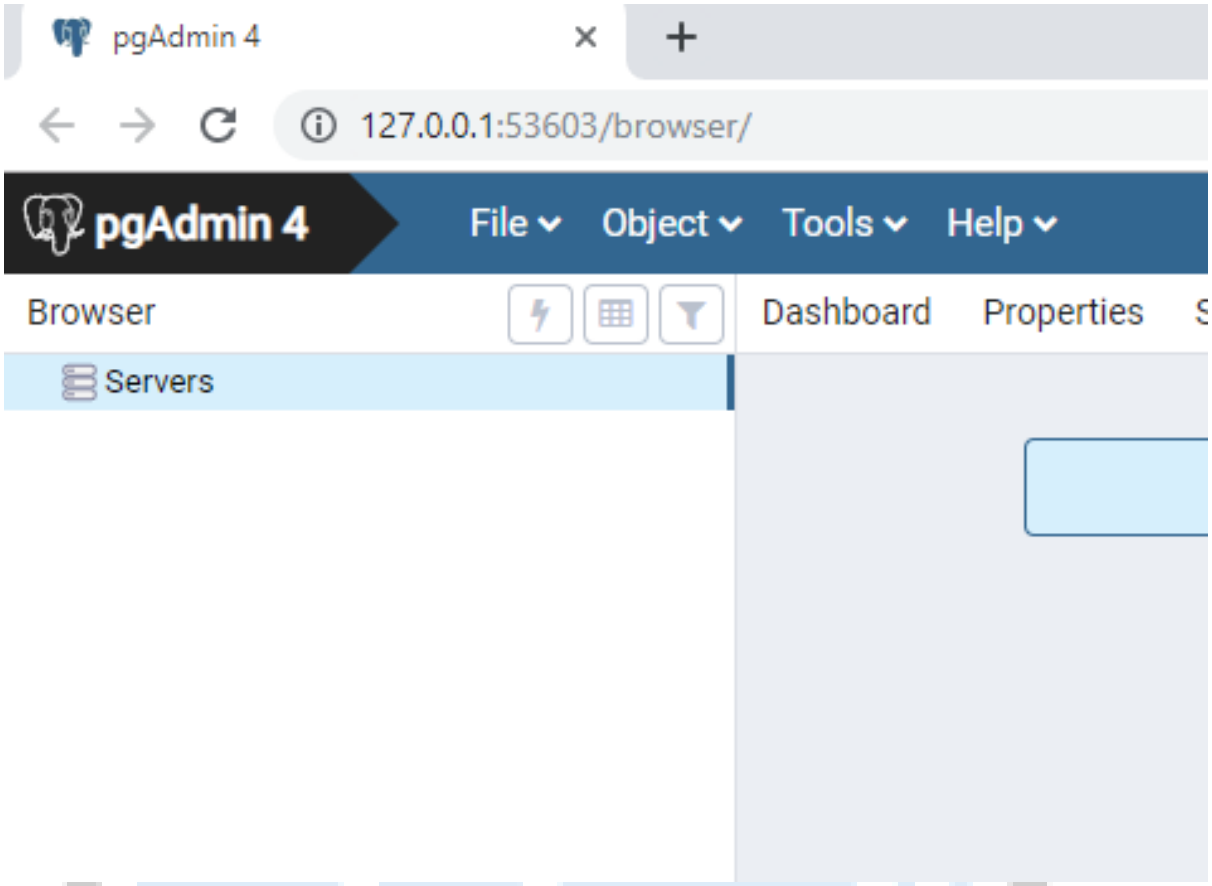
The second way to connect to a database is using pgAdmin application. By using the pgAdmin application, you can interact with the PostgreSQL database server via an intuitive user interface.

The following illustrates how to connect to a database using pgAdmin GUI application:

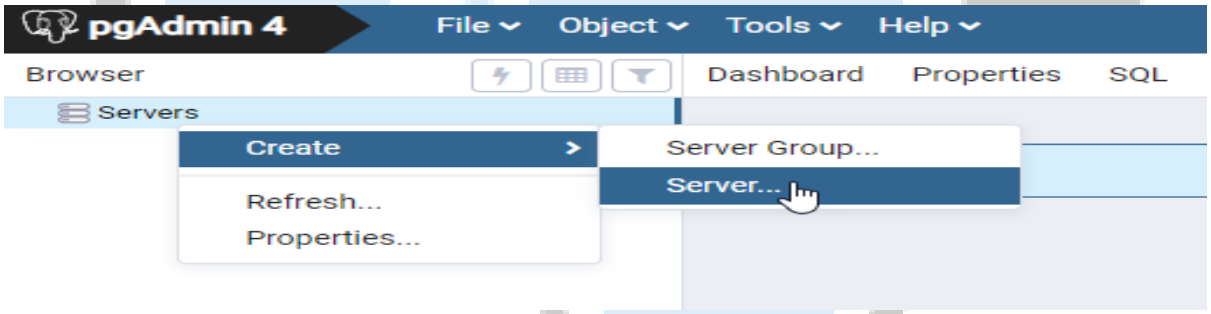
First, launch the pgAdmin application.



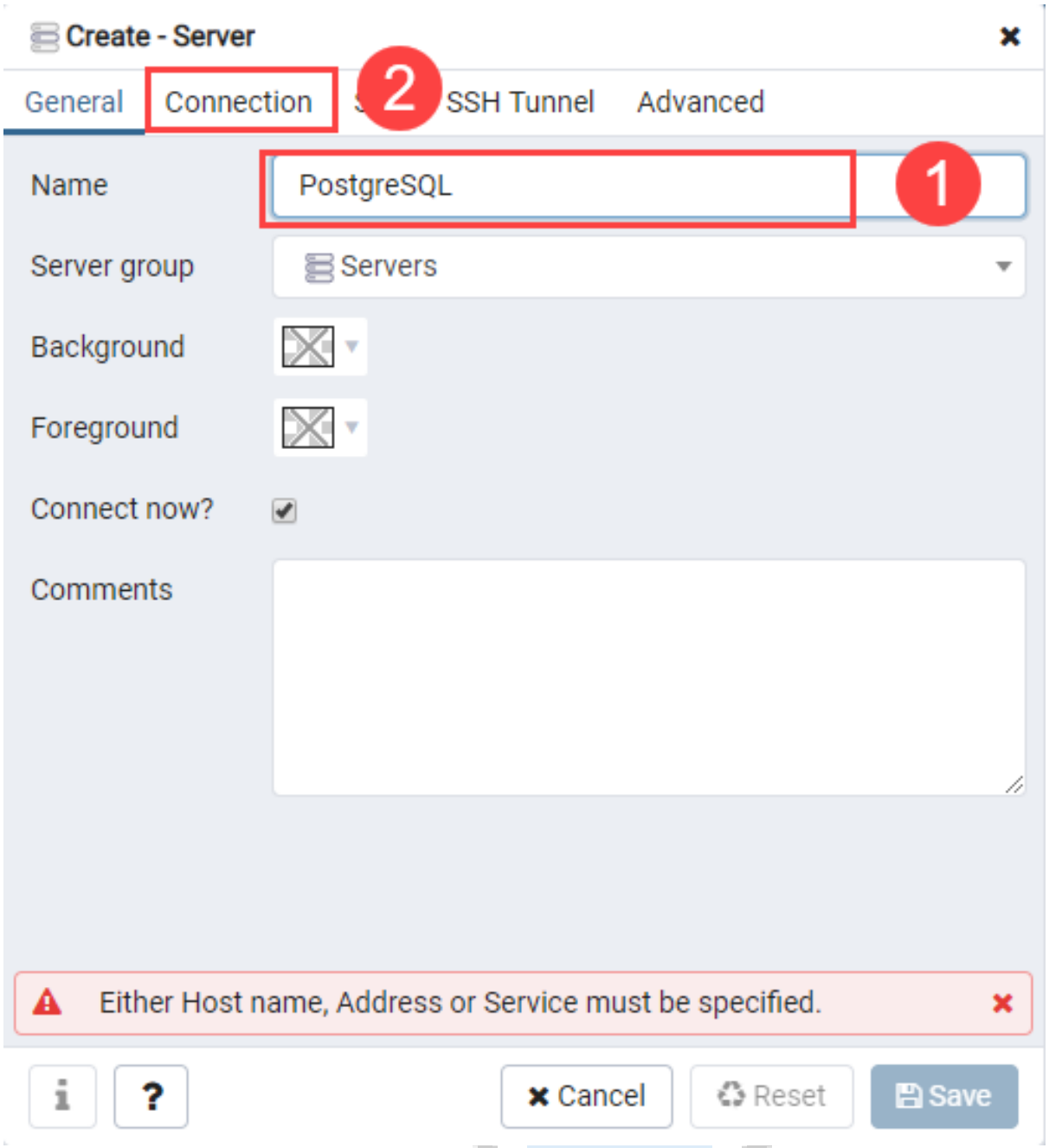
The pgAdmin application version 4 will launch on the web browser as shown in the following picture:



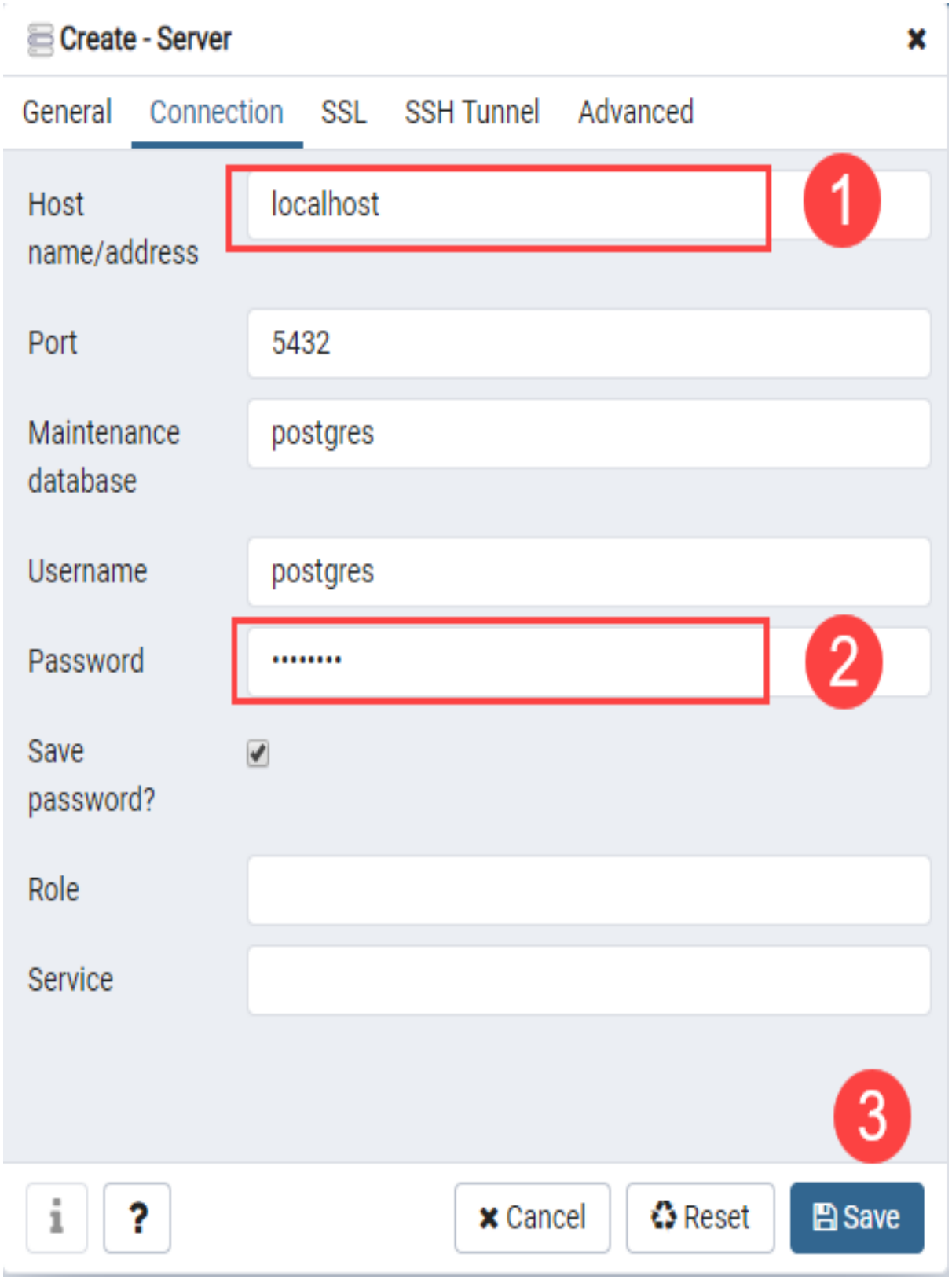
Second, right-click the Servers node and select **Create > Server…** menu to create a server



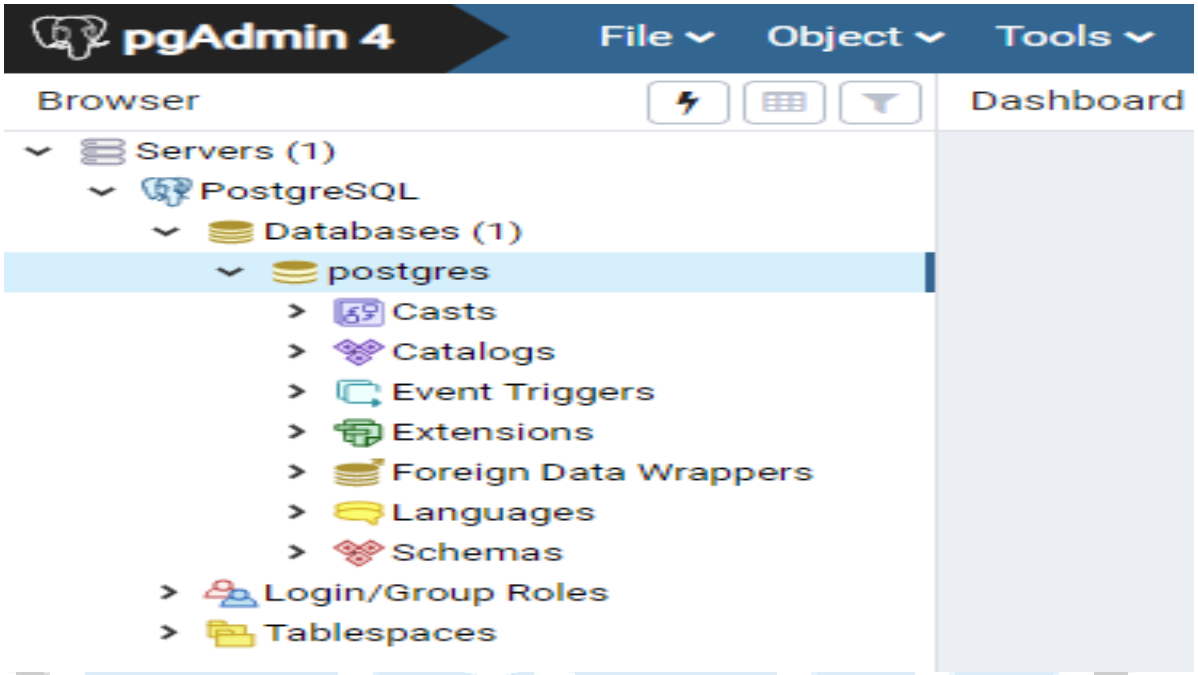
Third, enter the server name e.g., **PostgreSQL** and click the **Connection** tab:



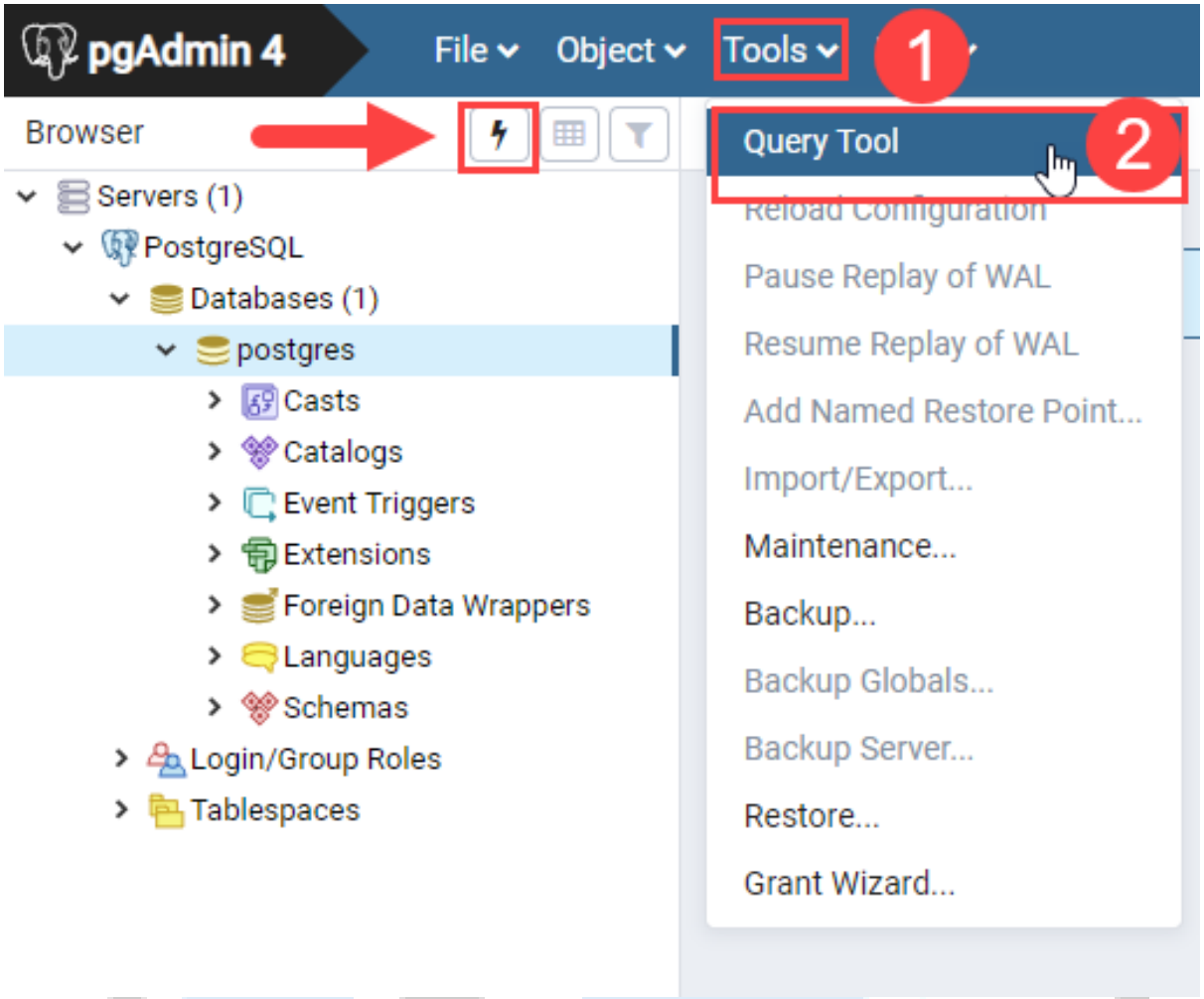
Fourth, enter the host and password for the **postgres** user and click the **Save** button:



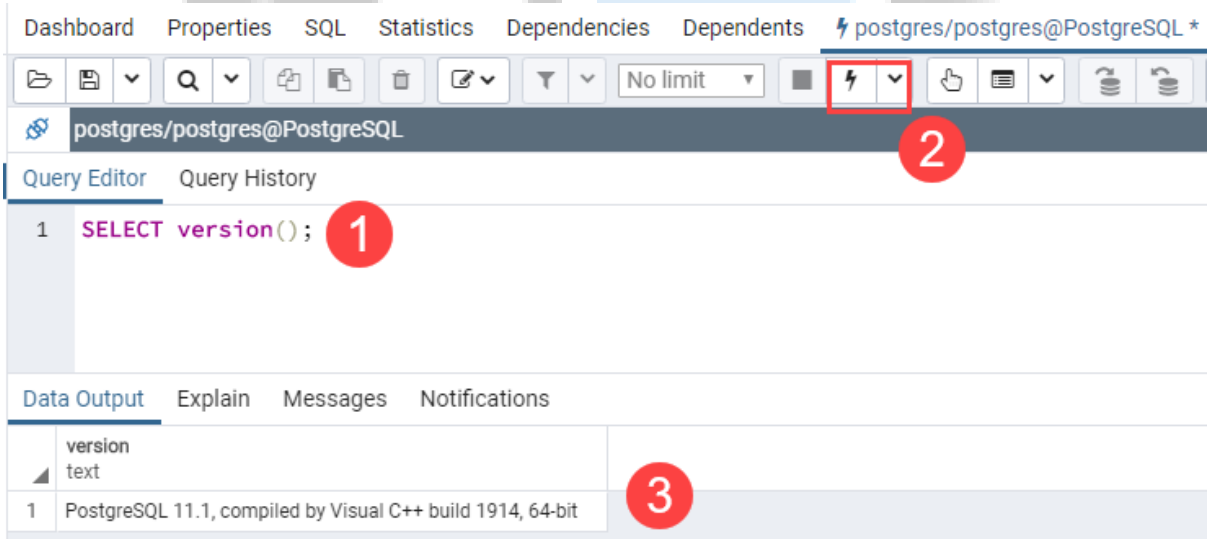
Fifth, click on the Servers node to expand the server. By default, PostgreSQL has a database named postgres as shown below:



Sixth, open the query tool by choosing the menu item **Tool > Query Tool** or click the lightning icon.



Seventh, enter the query in the **Query Editor**, click the **Execute** button, you will see the result of the query displaying in the **Data Output** tab:



Connect to PostgreSQL database from other applications

Any application that supports ODBC or JDBC can connect to the PostgreSQL database server. In addition, if you develop an application that uses an appropriate driver, the application can connect to the PostgreSQL database server as well.

- Connect to PostgreSQL from PHP
- Connect to PostgreSQL from Python
- Connect to PostgreSQL from Java
-

4.PostgreSQL Server and Database Objects

you are going to get familiar with the most common **server and database objects** provided by PostgreSQL. It is important to understand those objects and their functionality so you do not miss out on the cool features that you may wish to have in the system.

After installing PostgreSQL, loading sample database and connecting to the database server using *pgAdmin*GUI application, you will see that PostgreSQL provides many server and database objects. To leverage the features of each object that PostgreSQL provides effectively, you should have a good understanding of what each object is and how to use it effectively.

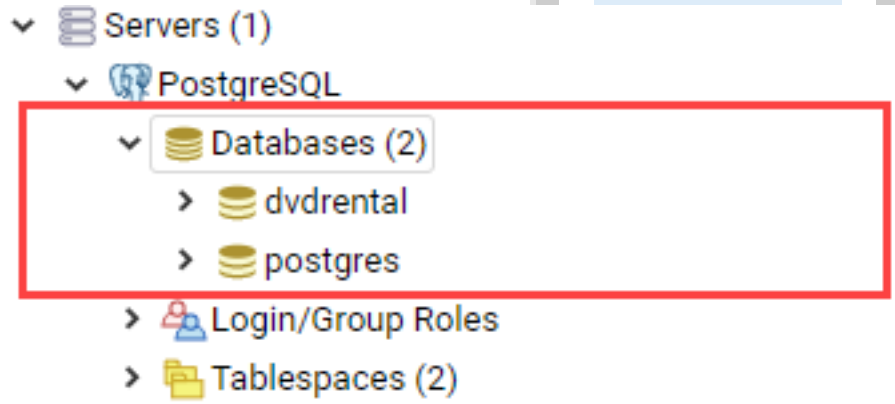
Let's get familiar with these PostgreSQL server and database objects.

Server service

When you install a PostgreSQL instance, you will have a corresponding PostgreSQL server service. The PostgreSQL server service is also known as the PostgreSQL server. You can install multiple PostgreSQL servers on a physical server using different ports and having different locations to store data.

Databases

A database is a container of other objects such as tables, views, functions, and indexes. You can create as many databases as you want inside a PostgreSQL server.



PostgreSQL Server and Database Objects

you are going to get familiar with the most common **server and database objects**provided by PostgreSQL. It is important to understand those objects and their functionality so you do not miss out on the cool features that you may wish to have in the system.

Let's get familiar with these PostgreSQL server and database objects.

Server service

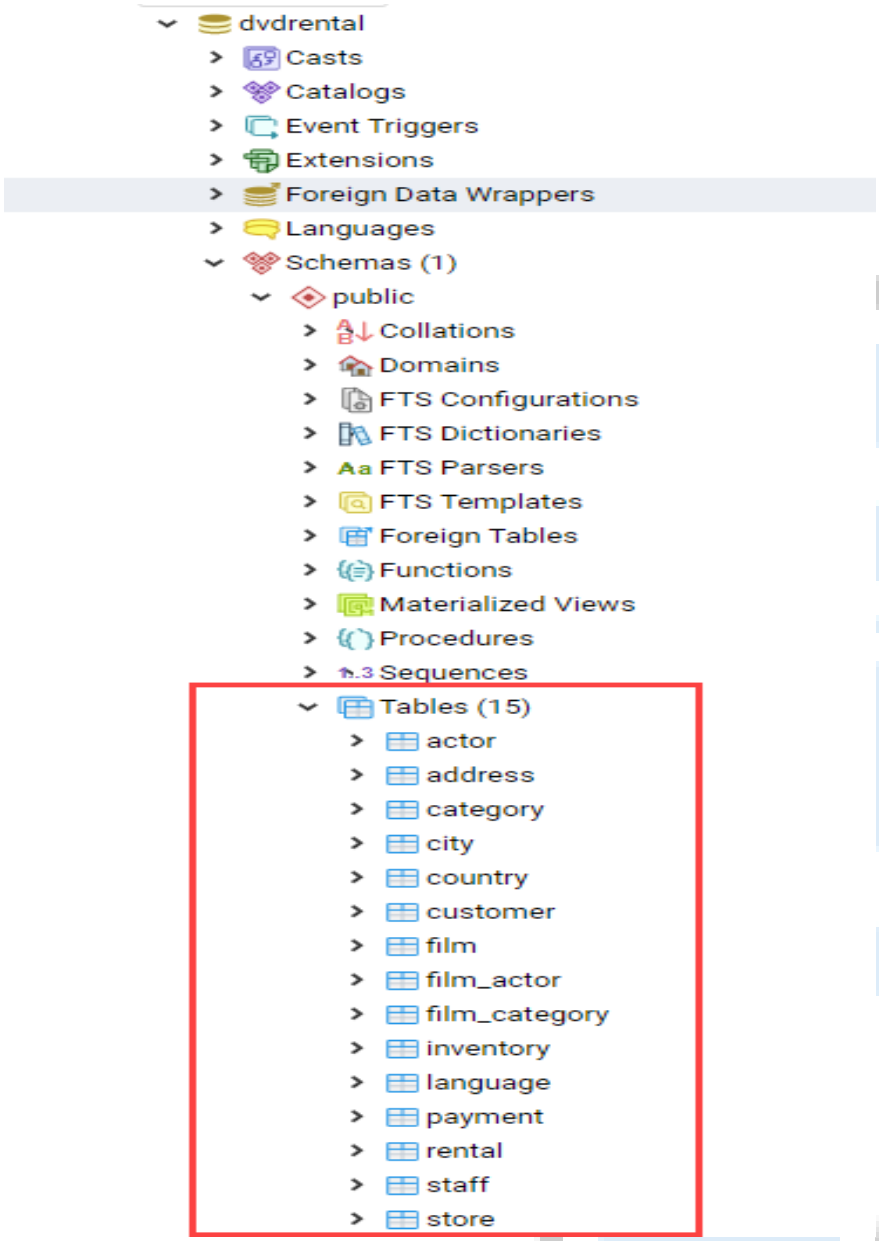
When you install a PostgreSQL instance, you will have a corresponding PostgreSQL server service. The PostgreSQL server service is also known as the PostgreSQL server. You can install multiple PostgreSQL servers on a physical server using different ports and having different locations to store data.

Databases

A database is a container of other objects such as tables, views, functions, and indexes. You can create as many databases as you want inside a PostgreSQL server.

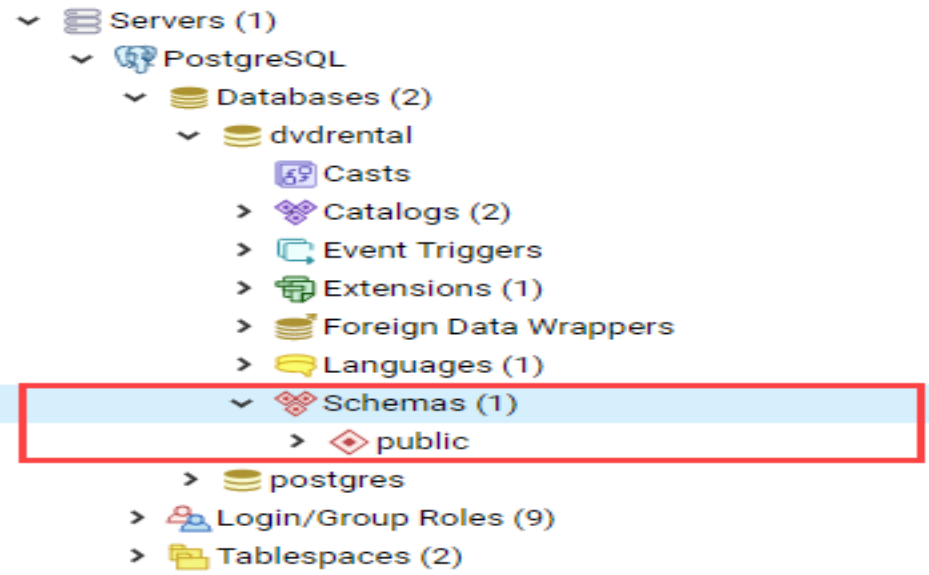
Tables

Tables store data. A table belongs to a database and each database has multiple tables. A special feature of PostgreSQL is table inheritance, meaning that a table (child table) can inherit from another table (parent table) so when you query data from the child table, the data from the parent table is also showing up.



Schemas

A schemalogical container of tables and other objects inside a database. Each PostgreSQL database may have multiple schemas. It is important to note that schema is a part of the ANSI-SQL standard.



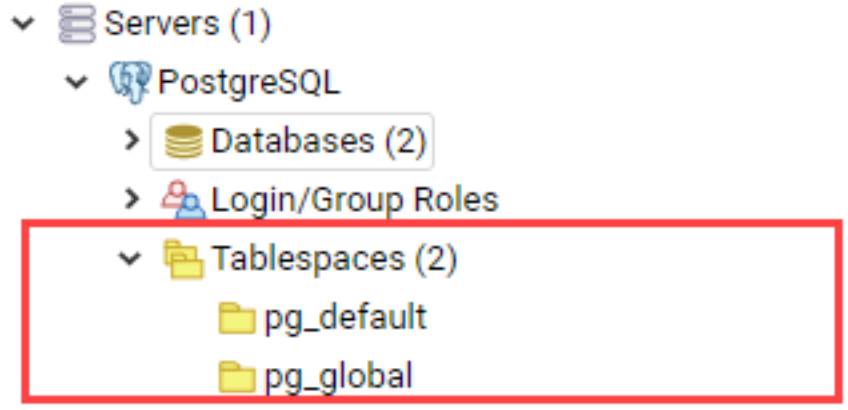
Tablespaces

Tablespaces are where PostgreSQL stores the data. Tablespaces allow you to move your data to different physical locations across drivers easily by using simple commands.

By default, PostgreSQL provides you with two tablespaces:

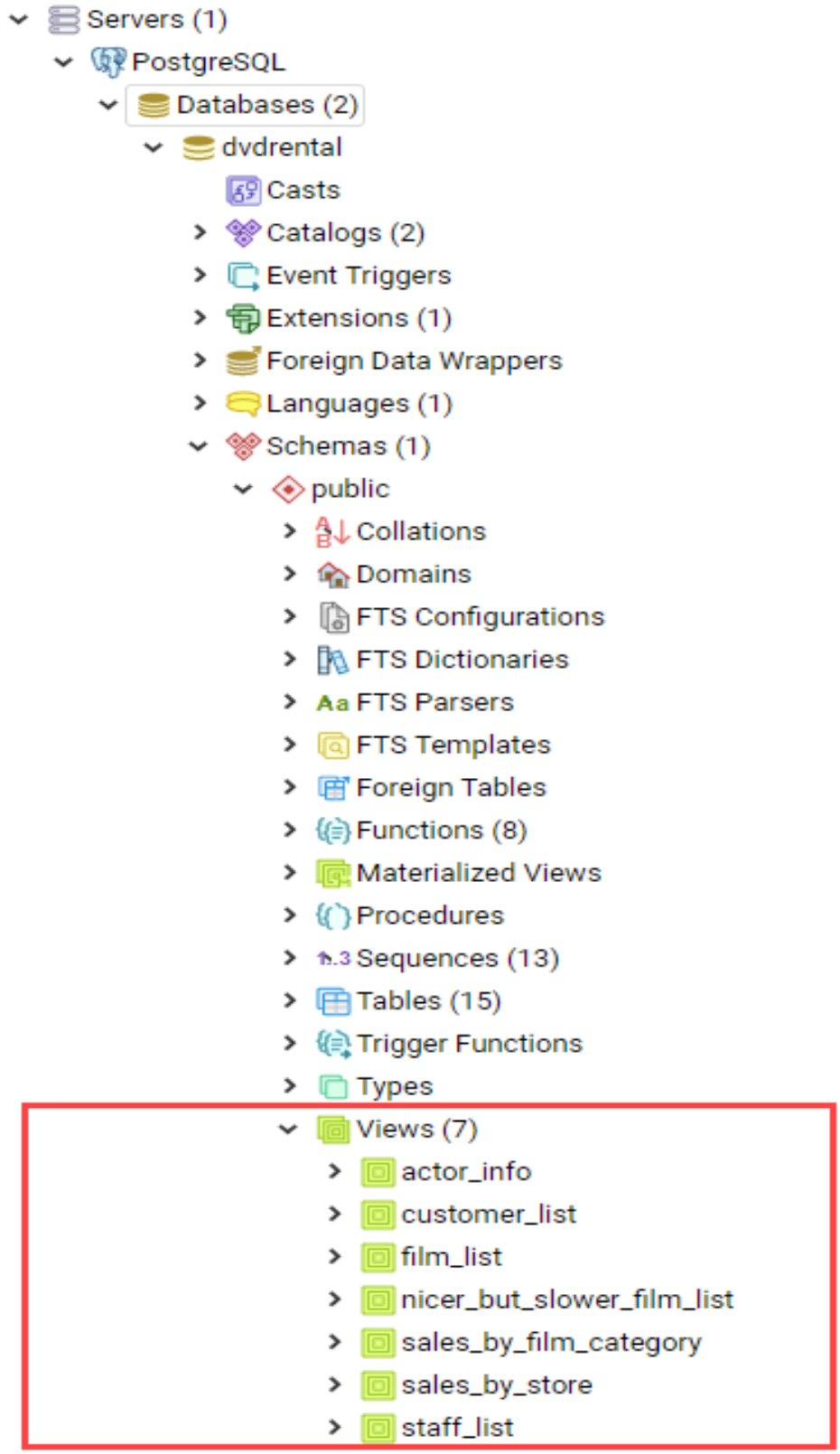
1. The pg_default is for storing user data.
2. The pg_global is for storing system data.

The following picture shows the default tablespaces:



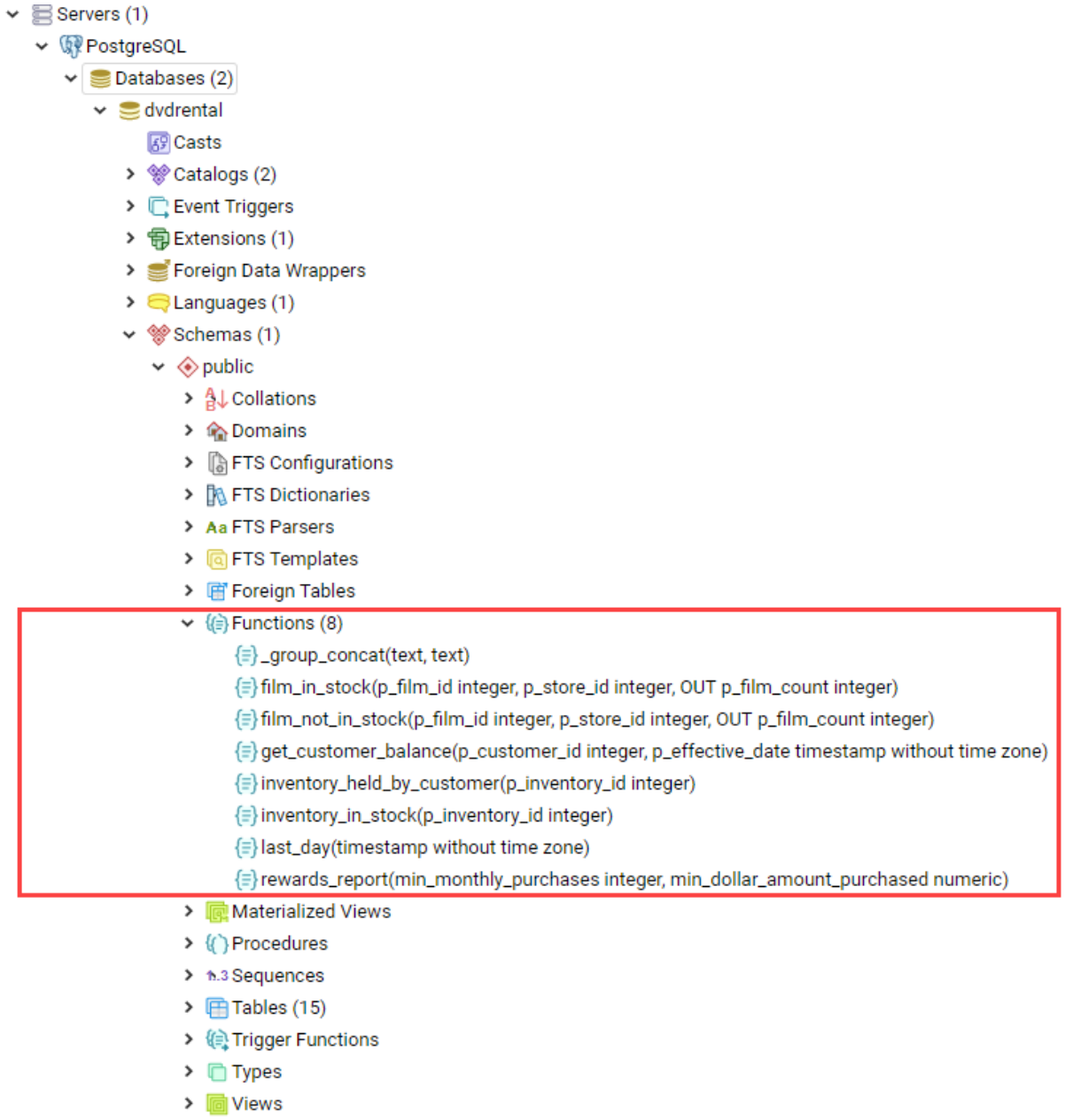
Views

Views are named query stored in the database. Besides the read-only views, PostgreSQL support Updated views.



Functions

A function is a reusable block of SQL code that returns a scalar value of a set of rows.



Operators

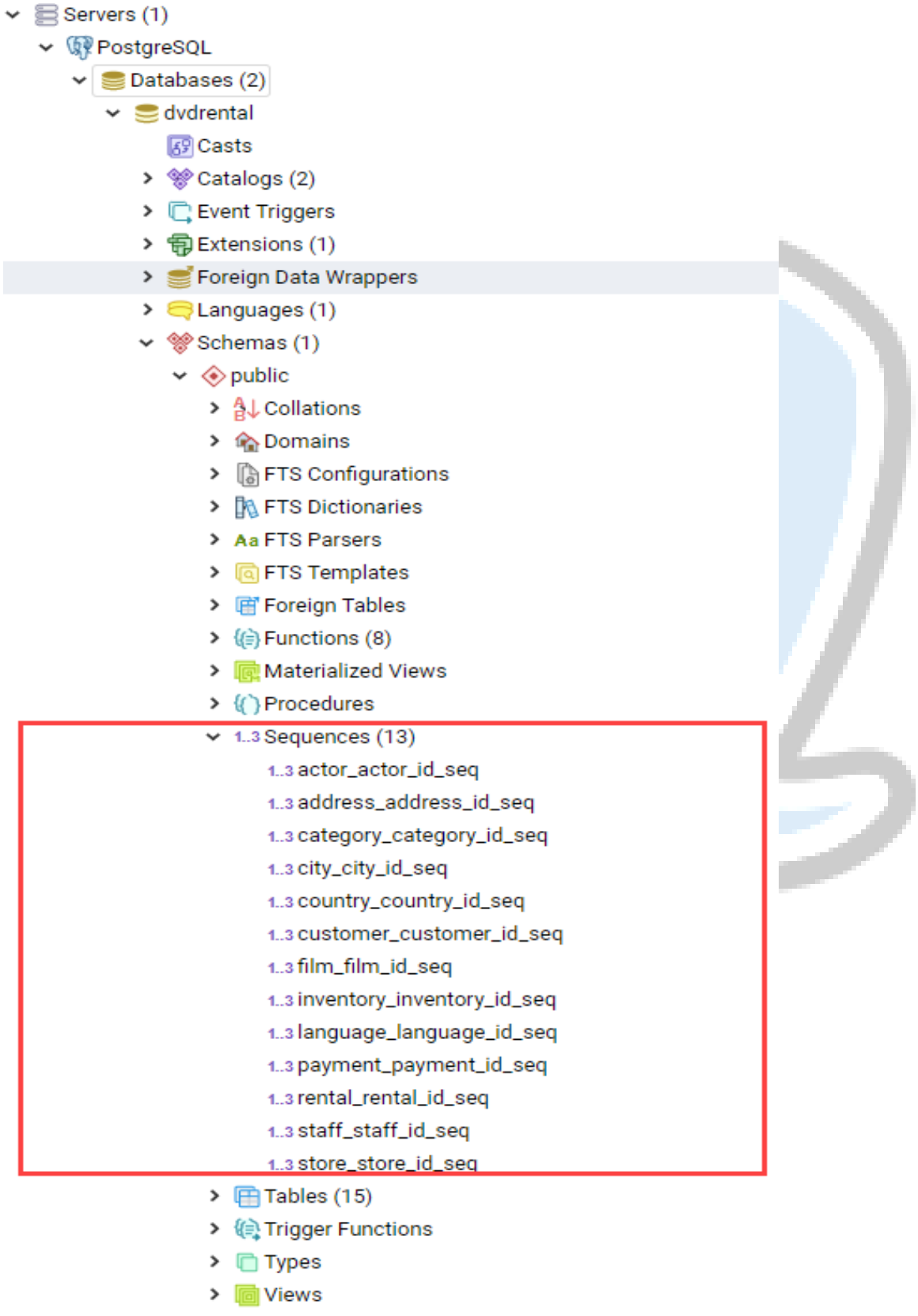
Operators are symbolic functions. PostgreSQL allows you to define custom operators.

Casts

Casts enable you to convert one data type into another data type. Casts backed by functions to perform the conversion. You can also create your casts to override the default casting provided by PostgreSQL.

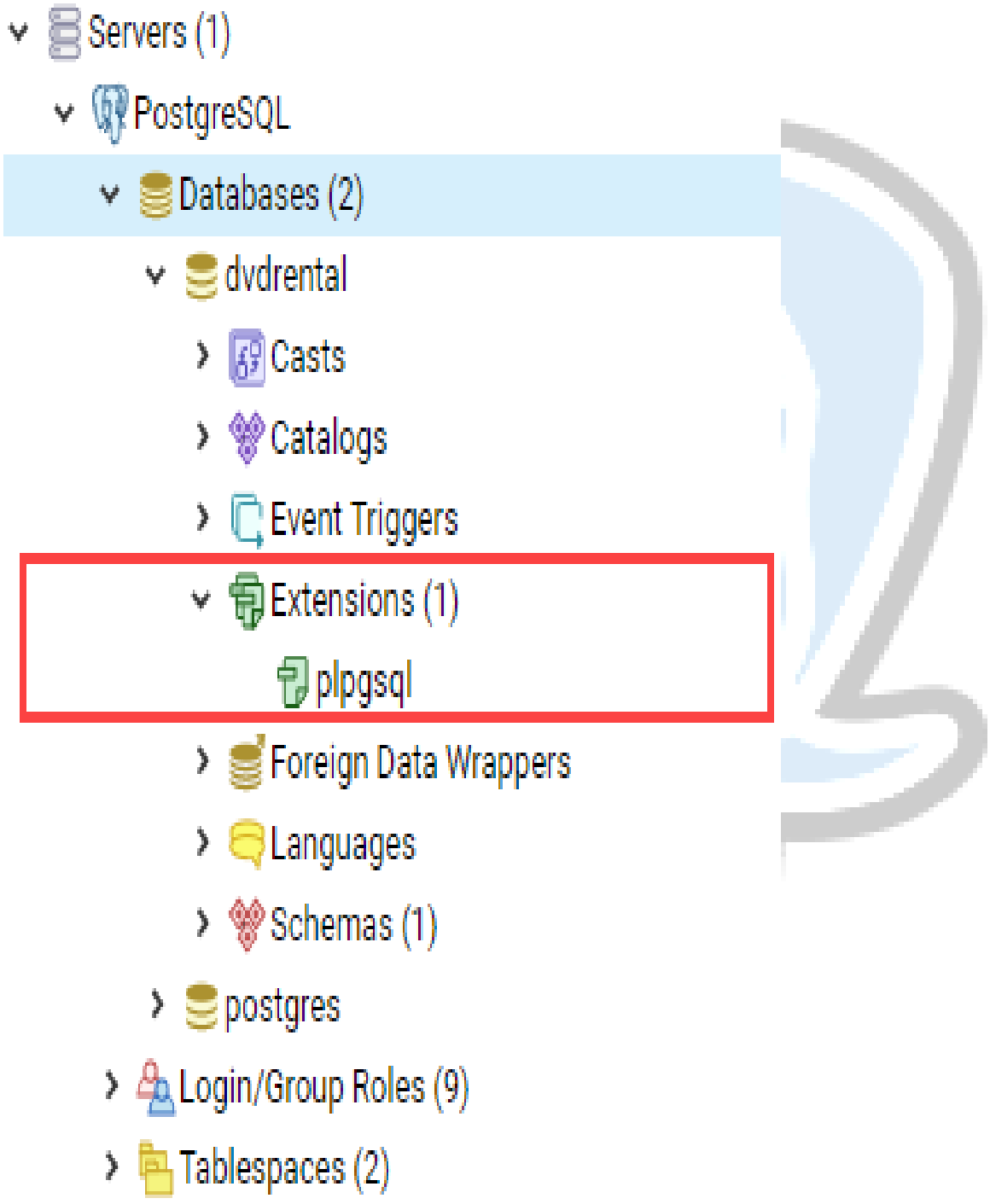
Sequence

Sequences are used to manage auto-increment columns that defined in a table as a serial


Servers (1)
PostgreSQL
Databases (2)
dvdrental
Casts
Catalogs (2)
Event Triggers
Extensions (1)
Foreign Data Wrappers
Languages (1)
Schemas (1)
public
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions (8)
Materialized Views
Procedures
Sequences (13)
actor_actor_id_seq
address_address_id_seq
category_category_id_seq
city_city_id_seq
country_country_id_seq
customer_customer_id_seq
film_film_id_seq
inventory_inventory_id_seq
language_language_id_seq
payment_payment_id_seq
rental_rental_id_seq
staff_staff_id_seq
store_store_id_seq
Tables (15)
Trigger Functions
Types
Views

Extension

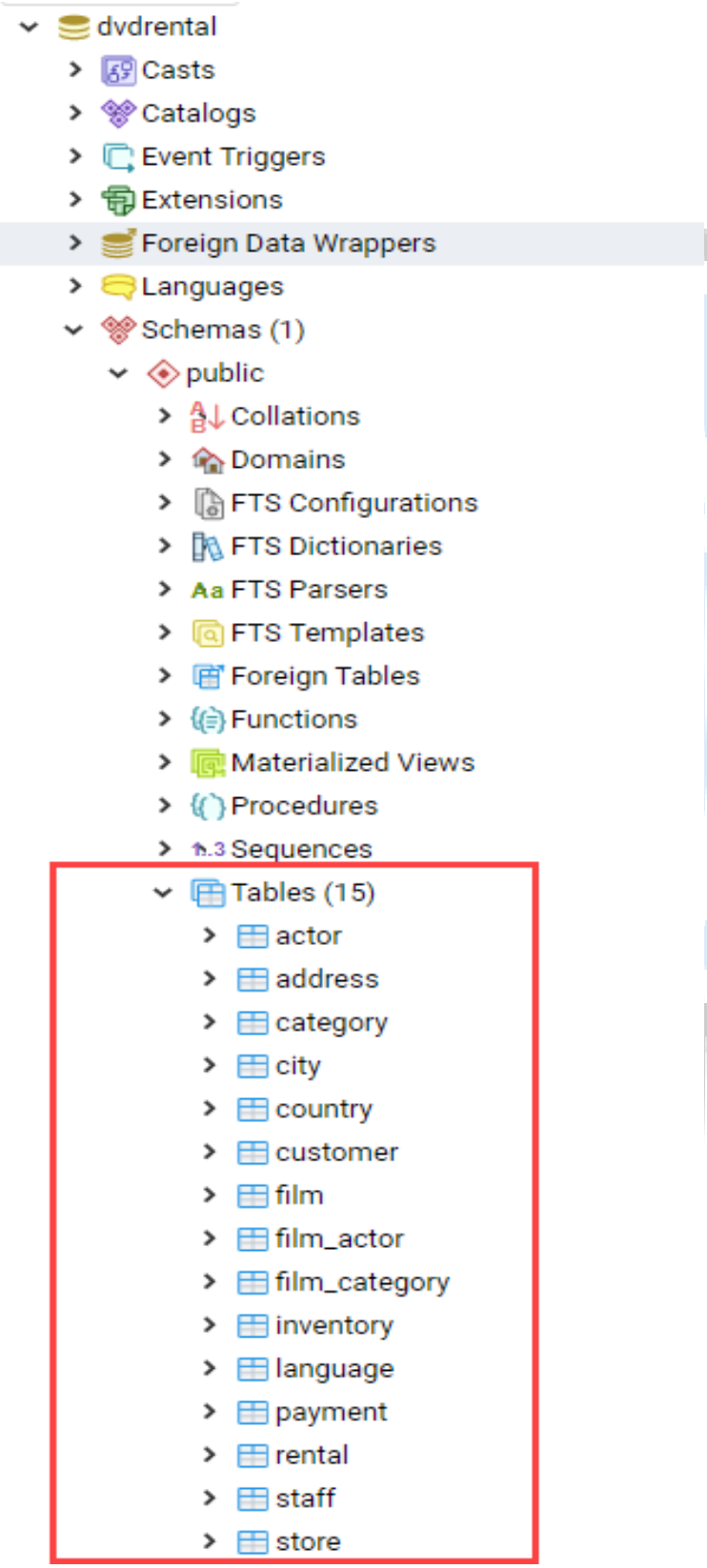
PostgreSQL introduced extension concept since version 9.1 to wrap other objects including types, casts, indexes, functions, etc., into a single unit. The purpose of extensions is to make it easier to maintain.



Tables store data. A table belongs to a database and each database has multiple tables.

A special feature of PostgreSQL is table inheritance, meaning that a table (child table) can inherit from another table (parent table) so when you query data from the child table, the data from

the parent table is also showing



up.

2.Customer Table with different Data in PostgreSQL

Query Editor Query History

```
CREATE  TABLE CUSTOMER(
customer_id int primary key not null,
store_id int,
first_name text,
last_name text,
email text,
address_id char(50),
activebool bool,
create_date date,
last_update timestamp,
active bool
)

```

Data Output Explain Messages Notifications

Query Editor Query History

```
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(524,1,'Jared','Ely','jared.ely@sakilacustomer.org',530,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(1,1,'Mary','Smith','mary.smith@sakilacustomer.org',5,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(2,1,'Patricia','Johnson','patricia.johnson@sakilacustomer.org',6,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(3,1,'Linda','Williams','linda.williams@sakilacustomer.org',7,'t','2006-02-14','2013-05-26 14:49:45.738','1');
```

Data Output Explain Messages Notifications

Query Editor Query History

```
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(4,2,'Barbara','Jones','barbara.jones@sakilacustomer.org',8,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(5,1,'Elizabeth','Brown','elizabeth.brown@sakilacustomer.org',9,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(6,2,'Jennifer','Davis','jennifer.davis@sakilacustomer.org',10,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(7,1,'Maria','Miller','maria.miller@sakilacustomer.org',11,'t','2006-02-14','2013-05-26 14:49:45.738','1');
```

Data Output Explain Messages Notifications

Query Editor Query History

```
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(8,2,'Susan','Wilson','susan.wilson@sakilacustomer.org',12,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool, create_date,last_update,active)
values
(9,2,'Margaret','Moore','margaret.moore@sakilacustomer.org',13,'t','2006-02-14','2013-05-26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool,create_date,last_update,active)
values
(10,1,'Dorothy','Taylor','dorothy.taylor@salikacustomer.org',14,'t','2006-02-14','2013-05--26 14:49:45.738','1');
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool,create_date,last_update,active)
values
(11,2,'Lisa','Anderson','lisa.anderson@salikacustomer.org',15,'t','2006-02-14','2013-05--26 14:49:45.738','1');
```

Data Output Explain Messages Notifications

Query Editor Query History

```
insert into CUSTOMER
(customer_id,store_id,first_name,last_name,email,address_id,activebool,create_date,last_update,active)
values
(12,1,'Nancy','Thomas','nancy.thomas@salikacustomer.org',16,'t','2006-02-14','2013-05--26 14:49:45.738','1');


select * from CUSTOMER
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer	first_name text	last_name text	email text	address_id character (50)	activebool boolean	create_date date	last_update timestamp without time zone	active boolean
1	524	1	Jared	Ely	jared.ely...	530	... true	2006-02-14	2013-05-26 14:49:45.738	true
2	1	1	Mary	Smith	mary.sm...	5	... true	2006-02-14	2013-05-26 14:49:45.738	true
3	2	1	Patricia	Johnson	patricia.j...	6	... true	2006-02-14	2013-05-26 14:49:45.738	true
4	3	1	Linda	Williams	linda.will...	7	... true	2006-02-14	2013-05-26 14:49:45.738	true
5	4	2	Barbara	Jones	barbara.j...	8	... true	2006-02-14	2013-05-26 14:49:45.738	true
6	5	1	Elizabeth	Brown	elizabeth...	9	... true	2006-02-14	2013-05-26 14:49:45.738	true
7	6	2	Jennifer	Davis	jennifer....	10	... true	2006-02-14	2013-05-26 14:49:45.738	true
8	7	1	Maria	Miller	maria.mi...	11	... true	2006-02-14	2013-05-26 14:49:45.738	true
9	8	2	Susan	Wilson	susan.wi...	12	... true	2006-02-14	2013-05-26 14:49:45.738	true
10	9	2	Margaret	Moore	margaret...	13	... true	2006-02-14	2013-05-26 14:49:45.738	true
11	10	1	Dorothy	Taylor	dorothy.t...	14	... true	2006-02-14	2013-05-26 14:49:45.738	true
12	11	2	Lisa	Anderson	lisa.ande...	15	... true	2006-02-14	2013-05-26 14:49:45.738	true

Query Editor Query History

```
Select * from CUSTOMER
LIMIT 5;
```

Data Output Explain Messages Notifications

customer_id [PK] integer	store_id integer	first_name text	last_name text	email text	address_id character (50)	activebool boolean	create_date date	last_update timestamp without time zone	active boolean
1	524	1	Jared	Ely	jared.ely...	530	... true	2006-02-14	2013-05-26 14:49:45.738	true
2	1	1	Mary	Smith	mary.sm...	5	... true	2006-02-14	2013-05-26 14:49:45.738	true
3	2	1	Patricia	Johnson	patricia.j...	6	... true	2006-02-14	2013-05-26 14:49:45.738	true
4	3	1	Linda	Williams	linda.will...	7	... true	2006-02-14	2013-05-26 14:49:45.738	true
5	4	2	Barbara	Jones	barbara.j...	8	... true	2006-02-14	2013-05-26 14:49:45.738	true

Query Editor Query History

```
SELECT * FROM CUSTOMER
LIMIT 5 OFFSET 1000;
```

Data Output Explain Messages Notifications

customer_id [PK] integer	store_id integer	first_name text	last_name text	email text	address_id character (50)	activebool boolean	create_date date	last_update timestamp without time zone	active boolean

Successfully run. Total query runtim

Query Editor Query History

```
SELECT Customer_id,email,last_update
FROM CUSTOMER
ORDER BY Customer_id
   LIMIT 5;
```

Data Output Explain Messages Notifications

customer_id [PK] integer	email text	last_update timestamp without time zone
1	1	mary.smith@sakilacustomer.org	2013-05-26 14:49:45.738
2	2	patricia.johnson@sakilacustomer.org	2013-05-26 14:49:45.738
3	3	linda.williams@sakilacustomer.org	2013-05-26 14:49:45.738
4	4	barbara.jones@sakilacustomer.org	2013-05-26 14:49:45.738
5	5	elizabeth.brown@sakilacustomer.org	2013-05-26 14:49:45.738

Query Editor Query History

```
SELECT Customer_id,email,last_update
FROM CUSTOMER
ORDER BY Customer_id
LIMIT 5 OFFSET 3;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	email text	last_update timestamp without time zone
1	4	barbara.jones@sakilacustomer.org	2013-05-26 14:49:45.738
2	5	elizabeth.brown@sakilacustomer.org	2013-05-26 14:49:45.738
3	6	jennifer.davis@sakilacustomer.org	2013-05-26 14:49:45.738
4	7	maria.miller@sakilacustomer.org	2013-05-26 14:49:45.738
5	8	susan.wilson@sakilacustomer.org	2013-05-26 14:49:45.738

Query Editor Query History

```
SELECT Customer_id,Store_id,email
FROM customer
ORDER BY store_id
LIMIT 10;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer	email text
1	524	1	jared.ely@sakilacustomer.org
2	1	1	mary.smith@sakilacustomer.org
3	2	1	patricia.johnson@sakilacustomer.org
4	3	1	linda.williams@sakilacustomer.org
5	5	1	elizabeth.brown@sakilacustomer.org
6	7	1	maria.miller@sakilacustomer.org
7	10	1	dorothy.taylor@salikacustomer.org
8	12	1	nancy.thomas@salikacustomer.org
9	8	2	susan.wilson@sakilacustomer.org
10	4	2	barbara.jones@sakilacustomer.org

Query Editor Query History

```
SELECT customer_id,store_id
FROM customer
ORDER BY Store_id
FETCH  FIRST 1 ROW ONLY;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer
1	524	1

Query Editor Query History

```
SELECT customer_id,store_id
FROM customer
ORDER BY Store_id
FETCH FIRST 5 ROW ONLY;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer
1	524	1
2	2	1
3	3	1
4	1	1
5	5	1

Query Editor Query History

```
FROM customer
ORDER BY Store_id
FETCH FIRST 5 ROW ONLY;

SELECT customer_id,store_id
FROM customer
ORDER BY Store_id
OFFSET 5 ROWS
FETCH FIRST 5 ROW ONLY;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer
1	7	1
2	10	1
3	12	1
4	8	2
5	4	2

Query Editor Query History

```
SELECT customer_id,store_id,create_date
FROM customer
WHERE customer_id IN (1, 2)
ORDER BY create_date DESC;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer	create_date date
1	1	1	2006-02-14
2	2	1	2006-02-14

Query Editor Query History

```
SELECT customer_id,store_id,create_date
FROM customer
WHERE customer_id = 1 OR customer_id = 2
ORDER BY create_date DESC;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer	create_date date
1	1	1	2006-02-14
2	2	1	2006-02-14

Query Editor Query History

```
SELECT customer_id,store_id,create_date
FROM customer
WHERE customer_id NOT IN (1, 2)
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer	create_date date
1	524	1	2006-02-14
2	3	1	2006-02-14
3	4	2	2006-02-14
4	5	1	2006-02-14
5	6	2	2006-02-14
6	7	1	2006-02-14
7	8	2	2006-02-14
8	9	2	2006-02-14
9	10	1	2006-02-14
10	11	2	2006-02-14
11	12	1	2006-02-14

Successfully run. Total query runtime: 121 msec. 11 rows affected.

Query Editor Query History

```
SELECT customer_id,store_id,create_date FROM customer
WHERE customer_id <> 1 OR customer_id <> 2
ORDER BY create_date DESC;
```

Data Output Explain Messages Notifications

	customer_id [PK] integer	store_id integer	create_date date
1	524	1	2006-02-14
2	1	1	2006-02-14
3	2	1	2006-02-14
4	3	1	2006-02-14
5	4	2	2006-02-14
6	5	1	2006-02-14
7	6	2	2006-02-14
8	7	1	2006-02-14
9	8	2	2006-02-14
10	9	2	2006-02-14
11	10	1	2006-02-14
12	11	2	2006-02-14
13	12	1	2006-02-14

Successfully run. Total query runtime: 137 msec. 13 rows affected.

Query Editor Query History

```
SELECT customer_id FROM customer
WHERE CAST (create_date AS DATE) = '2006-02-14';
```

Data Output Explain Messages Notifications

	customer_id [PK] integer
1	524
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	10
12	11
13	12

Query Editor Query History

```
SELECT first_name,last_name FROM customer
WHERE customer_id IN (SELECT customer_id FROM customer WHERE CAST (create_date AS DATE) = '2006-02-14');
```

Data Output Explain Messages Notifications

	first_name text	last_name text
1	Jared	Ely
2	Mary	Smith
3	Patricia	Johnson
4	Linda	Williams
5	Barbara	Jones
6	Elizabeth	Brown
7	Jennifer	Davis
8	Maria	Miller
9	Susan	Wilson
10	Margaret	Moore
11	Dorothy	Taylor
12	Lisa	Anderson
13	Nancy	Thomas

3.Different types of fetching data in PostgreSQL

Query Editor | Query History

```
SELECT customer_id,store_id,create_date
FROM customer
WHERE store_id BETWEEN 0 AND 1;
```

Data Output | Explain | Messages | Notifications

	customer_id [PK] integer	store_id integer	create_date date
1	524	1	2006-02-14
2	1	1	2006-02-14
3	2	1	2006-02-14
4	3	1	2006-02-14
5	5	1	2006-02-14
6	7	1	2006-02-14
7	10	1	2006-02-14
8	12	1	2006-02-14

Query Editor | Query History

```
SELECT first_name,last_name
FROM customer
WHERE first_name LIKE 'Jen%';
```

Data Output | Explain | Messages | Notifications

	first_name text	last_name text
1	Jennifer	Davis

Query Editor | Query History

```
SELECT first_name,last_name
FROM customer
WHERE first_name LIKE '%er%' ;
```

Data Output | Explain | Messages | Notifications

	first_name text	last_name text
1	Jennifer	Davis

Query Editor | Query History

```
SELECT first_name,last_name
FROM customer
WHERE first_name LIKE '_her%';
```

Data Output | Explain | Messages | Notifications

first_name text	last_name text

Query Editor | Query History

```
SELECT first_name,last_name
FROM customer
WHERE first_name NOT LIKE 'Jen%';
```

Data Output | Explain | Messages | Notifications

	first_name text	last_name text
1	Jared	Ely
2	Mary	Smith
3	Patricia	Johnson
4	Linda	Williams
5	Barbara	Jones
6	Elizabeth	Brown
7	Maria	Miller
8	Susan	Wilson
9	Margaret	Moore
10	Dorothy	Taylor
11	Lisa	Anderson
12	Nancy	Thomas

Query Editor | Query History

```
SELECT first_name,last_name
FROM customer
WHERE first_name ILIKE 'BAR%';
```

Data Output | Explain | Messages | Notifications

	first_name text	last_name text
1	Barbara	Jones

4.PostgreSQL (Boolean, Timestamp and others)

Query Editor Query History

```
CREATE TABLE stock_availability (
product_id INT NOT NULL PRIMARY KEY,
available BOOLEAN NOT NULL
);
INSERT INTO stock_availability (product_id, available)
VALUES
(100, TRUE),
(200, FALSE),
(300, 't'),
(400, '1'),
(500, 'y'),
(600, 'yes'),
(700, 'no'),
(800, '0');

SELECT * FROM stock_availability
```

Data Output Explain Messages Notifications

	product_id [PK] integer	available boolean
1	100	true
2	200	false
3	300	true
4	400	true
5	500	true
6	600	true
7	700	false
8	800	false

Query Editor Query History

```
SELECT * FROM stock_availability
WHERE available = 'yes';
```

Data Output Explain Messages Notifications

	product_id [PK] integer	available boolean
1	100	true
2	300	true
3	400	true
4	500	true
5	600	true

Query Editor Query History

```
SELECT * FROM stock_availability

WHERE available = 'no';
```

Data Output Explain Messages Notifications

	product_id [PK] integer	available boolean
1	200	false
2	700	false
3	800	false

Query Editor Query History

```
ALTER TABLE stock_availability ALTER COLUMN available
SET DEFAULT FALSE;

INSERT INTO stock_availability (product_id)
VALUES
    (900);

SELECT *FROM stock_availability
WHERE product_id = 900;
```

Data Output Explain Messages Notifications

	product_id [PK] integer	available boolean
1	900	false

Query Editor Query History

```
CREATE TABLE boolean_demo
(
is_ok BOOL DEFAULT 't'
);
select * from boolean_demo
```

Data Output Explain Messages Notifications

is_ok boolean

```
CREATE TABLE timestamp_demo
(
ts TIMESTAMP, tstz TIMESTAMPTZ
);
SET timezone = 'America/Los_Angeles'
SHOW TIMEZONE;
INSERT INTO timestamp_demo
(ts, tstz)
VALUES
(
'2016-06-22 19:10:25-07',
'2016-06-22 19:10:25-07'
);

SELECT ts FROM timestampz_demo;

SET timezone = 'America/New_York';

SELECT ts, tstz FROM timestamp_demo;

SELECT NOW();

SELECT TIMEOFDAY();

SHOW TIMEZONE;

SELECT timezone('America/New_York','2016-06-01 00:00');
```

ts timestamp without time zone	tstz timestamp with time zone
2016-06-22 19:10:25	2016-06-23 02:10:25+00

2. Introduction to the PostgreSQL character types

```
CREATE TABLE character_tests
(
id serial PRIMARY KEY,
x CHAR (1),
y VARCHAR (10),
 z TEXT
);
INSERT INTO character_tests
(x, y, z)
VALUES
(
'Y',
'varchar(n)',
'This is a very long text for the PostgreSQL text column'
);

SELECT * FROM character_tests;
```

id [PK] integer	x character (1)	y character varying (10)	z text
1	Y	varchar(n)	This is a very long text for the PostgreSQL text column

3. An Overview Of PostgreSQL NUMERIC Type

PostgreSQL NUMERIC and NaN

```
CREATE TABLE IF NOT EXISTS products (
    id serial PRIMARY KEY,
    name VARCHAR NOT NULL,
    price NUMERIC (5, 2)
);
INSERT INTO products
(NAME, price)
VALUES
('Phone',500.215),
('Tablet',500.214);
UPDATE products
SET price = 'NaN'
WHERE id = 1;

SELECT * FROM products
ORDER BY price DESC;
```

id integer	name character varying	price numeric (5,2)
1	Phone	NaN
2	Tablet	500.21

5. PostgreSQL GROUP BY, HAVING, AGGEREGATE FUNCTIONS

Query Editor | Query History

```
create table payment
(payment_id int primary key not null,
customer_id int,
staff_id int,
rental_id int,
amount int,
payment_date date
);
insert into payment(payment_id int primary key not null,
customer_id int,
staff_id int,
rental_id int,
amount int,
payment_date date)
values
( 100,10,1,1000,250,'2019/10/11' ),
( 200,0,2,2000,350,'2019/10/12' ),
( 300,30,3,3000,450,'2019/10/13' ),
( 400,40,4,4000,550,'2019/10/12' ),
( 500,50,5,5000,650,'2019/10/12' )

SELECT * FROM Payment
```

Data Output | Explain | Messages | Notifications

	payment_id [PK] integer	customer_id integer	staff_id integer	rental_id integer	amount integer	payment_date date
1	100	10	1	1000	250	2019-10-11
2	200	0	2	2000	350	2019-10-12
3	300	30	3	3000	450	2019-10-13
4	400	40	4	4000	550	2019-10-12
5	500	50	5	5000	650	2019-10-12

Query Editor | Query History

```
SELECT customer_id FROM payment
GROUP BY customer_id;
```

Data Output | Explain | Messages | Notifications

	customer_id integer
1	40
2	30
3	10
4	50
5	20

Query Editor | Query History

```
SELECT customer_id, SUM (amount) FROM payment
GROUP BY customer_id;
```

Data Output | Explain | Messages | Notifications

	customer_id integer	sum bigint
1	40	550
2	30	450
3	10	250
4	50	650
5	20	350

Query Editor Query History

```
SELECT customer_id, SUM (amount) FROM payment
GROUP BY customer_id
ORDER BY SUM (amount) DESC;
```

Data Output Explain Messages Notifications

	customer_id integer	sum bigint
1	50	650
2	40	550
3	30	450
4	20	350
5	10	250

Query Editor Query History

```
SELECT staff_id, COUNT (payment_id) FROM payment
GROUP BY staff_id;
```

Data Output Explain Messages Notifications

	staff_id integer	count bigint
1	3	1
2	5	1
3	4	1
4	2	1
5	1	1

Query Editor Query History

```
SELECT customer_id, SUM (amount)
FROM payment
GROUP BY customer_id
HAVING SUM (amount) > 200;
```

Data Output Explain Messages Notifications

	customer_id integer	sum bigint
1	40	550
2	30	450
3	10	250
4	50	650
5	20	350

Query Editor Query History

```
SELECT store_id, COUNT (customer_id)FROM customer
GROUP BY store_id
```

Data Output Explain Messages Notifications

	store_id integer	count bigint
1	2	5
2	1	8

Query Editor Query History

```
CREATE TABLE sales (
brand VARCHAR NOT NULL,
segment VARCHAR NOT NULL,
quantity INT NOT NULL,
PRIMARY KEY (brand, segment)
);
INSERT INTO sales
(brand, segment, quantity)
VALUES
('ABC', 'Premium', 100),
('ABC', 'Basic', 200),
('XYZ', 'Premium', 100),
('XYZ', 'Basic', 300);
SELECT * FROM sales
```

Data Output Explain Messages Notifications

	brand [PK] character varying	segment [PK] character varying	quantity integer
1	ABC	Premium	100
2	ABC	Basic	200
3	XYZ	Premium	100
4	XYZ	Basic	300

Query Editor Query History

```
SELECT brand,segment,SUM (quantity) FROM sales
GROUP BY brand, segment;
```

Data Output Explain Messages Notifications

	brand [PK] character varying	segment [PK] character varying	sum bigint
1	XYZ	Basic	300
2	ABC	Premium	100
3	ABC	Basic	200
4	XYZ	Premium	100

Query Editor Query History

```
SELECT brand, SUM (quantity)FROM sales
GROUP BY brand;
```

Data Output Explain Messages Notifications

	brand character varying	sum bigint
1	ABC	300
2	XYZ	400

Query Editor Query History

```
SELECT segment, SUM (quantity) FROM sales
GROUP BY segment;
```

Data Output Explain Messages Notifications

	segment character varying	sum bigint
1	Basic	500
2	Premium	200

Query Editor Query History

```
SELECT SUM (quantity) FROM sales;
```

Data Output Explain Messages Notifications

	sum bigint
1	700

Query Editor Query History

```
SELECT brand,segment, SUM (quantity)FROM sales
GROUP BY brand,segment
UNION ALL
SELECT brand,NULL,SUM (quantity)
FROM sales
GROUP BY brand
UNION ALL
SELECT NULL,segment,SUM (quantity)FROM sales
GROUP BY segment
UNION ALL
SELECT NULL, NULL, SUM (quantity)FROM sales;
```

Data Output Explain Messages Notifications

	brand character varying	segment character varying	sum bigint
1	XYZ	Basic	300
2	ABC	Premium	100
3	ABC	Basic	200
4	XYZ	Premium	100
5	ABC	[null]	300
6	XYZ	[null]	400
7	[null]	Basic	500
8	[null]	Premium	200
9	[null]	[null]	700

Query Editor Query History

```
SELECT brand,segment,SUM (quantity)FROM sales
GROUP BY
GROUPING SETS (
(brand, segment),
(brand),
(segment),
()
);
```

Data Output Explain Messages Notifications

	brand [PK] character varying	segment [PK] character varying	sum bigint
1	[null]	[null]	700
2	XYZ	Basic	300
3	ABC	Premium	100
4	ABC	Basic	200
5	XYZ	Premium	100
6	ABC	[null]	300
7	XYZ	[null]	400
8	[null]	Basic	500
9	[null]	Premium	200

Query Editor Query History

```
SELECT GROUPING(brand) grouping_brand,
GROUPING(segment) grouping_segement,brand,segment,SUM(quantity)FROM sales
GROUP BY
 GROUPING SETS
 (
(brand, segment),
(brand),
(segment),
()
)
```

Data Output Explain Messages Notifications

	grouping_brand integer	grouping_segement integer	brand [PK] character varying	segment [PK] character varying	sum bigint
1	1	1	[null]	[null]	700
2	0	0	XYZ	Basic	300
3	0	0	ABC	Premium	100
4	0	0	ABC	Basic	200
5	0	0	XYZ	Premium	100
6	0	1	ABC	[null]	300
7	0	1	XYZ	[null]	400
8	1	0	[null]	Basic	500
9	1	0	[null]	Premium	200

6.Joins in PostgreSQL

```
CREATE TABLE basket_a
(
id INT PRIMARY KEY,
fruit VARCHAR (100) NOT NULL
);
CREATE TABLE basket_b
(
id INT PRIMARY KEY,
fruit VARCHAR (100) NOT NULL
);

INSERT INTO basket_a
(id, fruit)
VALUES
(1, 'Apple'),
(2, 'Orange'),
(3, 'Banana'),
(4, 'Cucumber');

INSERT INTO basket_b
(id, fruit)
VALUES
(1, 'Orange'),
(2, 'Apple'),
(3, 'Watermelon'),
(4, 'Pear');
```

1.PostgreSQL Inner join

```
SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
INNER JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit;

SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit
WHERE b.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit
```

	id_a integer	fruit_a character varying (100)	id_b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange

2.PostgreSQL left join

```
SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit;
```

	id_a integer	fruit_a character varying (100)	id_b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]

3.PostgreSQL LEFT OUTER JOIN .

```
SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
INNER JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit;

SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit
WHERE b.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit
```

	id_a integer	fruit_a character varying (100)	id_b integer	fruit_b character varying (100)
1	3	Banana	[null]	[null]
2	4	Cucumber	[null]	[null]

4.PostgreSQL right join .

```
SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit;

SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
LEFT JOIN basket_b b ON a.fruit = b.fruit
WHERE b.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit
WHERE a.id IS NULL;
```

	id_a integer	fruit_a character varying (100)	id_b integer	fruit_b character varying (100)
1	[null]	[null]	3	Watermelon
2	[null]	[null]	4	Pear

5. PostgreSQL right OUTER join .

```
LEFT JOIN basket_b b ON a.fruit = b.fruit
WHERE b.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit
WHERE a.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
FULL OUTER JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
```

	id_a integer	fruit_a character varying (100)	id_b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]
5	[null]	[null]	3	Watermelon
6	[null]	[null]	4	Pear

6.PostgreSQL full outer join .

Query Editor Query History

```
LEFT JOIN basket_b b ON a.fruit = b.fruit
WHERE b.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit
WHERE a.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
FULL OUTER JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
```

Data Output Explain Messages Notifications

	id_a integer	fruit_a character varying (100)	id_b integer	fruit_b character varying (100)
1	1	Apple	2	Apple
2	2	Orange	1	Orange
3	3	Banana	[null]	[null]
4	4	Cucumber	[null]	[null]
5	[null]	[null]	3	Watermelon
6	[null]	[null]	4	Pear

7.PostgreSQL full outer join_only .

Query Editor Query History

```

SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
RIGHT JOIN basket_b b ON a.fruit = b.fruit
WHERE a.id IS NULL;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
FULL OUTER JOIN basket_b b ON a.fruit = b.fruit;


SELECT a.id id_a,a.fruit fruit_a,b.id id_b,b.fruit fruit_b FROM basket_a a
FULL JOIN basket_b b ON a.fruit = b.fruit
WHERE a.id IS NULL OR b.id IS NULL;

```

Data Output Explain Messages Notifications

	id_a integer	fruit_a character varying (100)	id_b integer	fruit_b character varying (100)
1	3	Banana	[null]	[null]
2	4	Cucumber	[null]	[null]
3	[null]	[null]	3	Watermelon
4	[null]	[null]	4	Pear

7.PostgreSQL (UUID, hstore, json, Array)

1.The Basics Of PostgreSQL UUID Data Type:

Query Editor Query History

```
CREATE TABLE contacts
(
contact_id uuid DEFAULT uuid_generate_v4(),
first_name VARCHAR NOT NULL,
last_name VARCHAR NOT NULL,
email VARCHAR NOT NULL,
phone VARCHAR,
PRIMARY KEY (contact_id)
);
INSERT INTO contacts
(first_name,last_name,email,phone)
VALUES
('John','Smith','john.smith@example.com','408-237-2345'),
('Jane','Smith','jane.smith@example.com','408-237-2344'),
('Alex','Smith','alex.smith@example.com','408-237-2343');
SELECT * FROM contacts
```

Data Output Explain Messages Notifications

	id [PK] integer	name character varying (100)	phones text[]
1	1	John Doe	{(408)-589-5846,(408)-589-5555}
2	2	Lily Bush	{(408)-589-5841}
3	3	William Gate	{(408)-589-5843}

2. PostgreSQL JSON :

Query Editor Query History

```
SELECT
   info ->> 'customer' AS customer,
   info -> 'items' ->> 'product' AS product FROM orders
WHERE CAST (info -> 'items' ->> 'qty' AS INTEGER) = 2

SELECT
   MIN (CAST (info -> 'items' ->> 'qty' AS INTEGER)),
   MAX (CAST (info -> 'items' ->> 'qty' AS INTEGER)),
   SUM ( CAST (info -> 'items' ->> 'qty' AS INTEGER)),
   AVG (CAST ( info -> 'items' ->> 'qty' AS INTEGER))FROM orders


SELECT json_each (info)FROM orders;

SELECT json_object_keys (info->'items')FROM orders;

SELECT json_typeof (info->'items')FROM orders;

SELECT json_typeof (info->'items'->'qty')FROM orders;
```

Data Output Explain Messages Notifications

Data Output Explain Messages Notifications

	info json
1	{"customer": "John Doe", "items": {"product": "Beer","qty": 6}}
2	{"customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}
3	{"customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}
4	{"customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}

3.PostgreSQL hstore:

```
CREATE EXTENSION hstore;
CREATE TABLE books (
    id serial primary key,
    title VARCHAR (255),
    attr hstore
);

INSERT INTO books (title, attr)
VALUES
   (
      'PostgreSQL Tutorial',
      '"paperback" => "243",
      "publisher" => "postgresqltutorial.com",
      "language"  => "English",
      "ISBN-13"   => "978-1449370000",
       "weight"    => "11.2 ounces"'
   );
INSERT INTO books (title, attr)
VALUES
   (
      'PostgreSQL Cheat Sheet',
      '
"paperback" => "5",
"publisher" => "postgresqltutorial.com",
"language"  => "English",
"ISBN-13"   => "978-1449370001",
"weight"    => "1 ounces"'
   );
```

```
SELECT attr FROM books;

SELECT attr -> 'ISBN-13' AS isbn FROM books;

SELECT attr -> 'weight' AS weight FROM books
WHERE attr -> 'ISBN-13' = '978-1449370000';

SELECT title,attr -> 'freeshipping' AS freeshipping FROM books;

UPDATE books
SET attr = attr || '"freeshipping"=>"no"' :: hstore;

UPDATE books
SET attr = delete(attr, 'freeshipping');
SELECT title,attr->'publisher' as publisher,attr FROM books WHERE attr ? 'publisher';

SELECT title FROM books WHERE attr @> '"weight"=>"11.2 ounces"' :: hstore;

SELECT title FROM books WHERE attr ?& ARRAY [ 'language', 'weight' ];

SELECT akeys (attr)FROM books;

SELECT skeys (attr)FROM books;

SELECT avals (attr)FROM books;

SELECT svals (attr)FROM books;

SELECT title,hstore_to_json (attr) json FROM books;

SELECT title,(EACH(attr) ).*FROM books;
```

Data Output | Explain | Messages | Notifications

	title character varying (255)	key text	value text
1	PostgreSQL Tutorial	weight	11.2 ounces
2	PostgreSQL Tutorial	ISBN-13	978-1449370000
3	PostgreSQL Tutorial	language	English
4	PostgreSQL Tutorial	paperback	243
5	PostgreSQL Tutorial	publisher	postgresqltutorial.com
6	PostgreSQL Cheat Sheet	weight	1 ounces
7	PostgreSQL Cheat Sheet	ISBN-13	978-1449370001
8	PostgreSQL Cheat Sheet	language	English
9	PostgreSQL Cheat Sheet	paperback	5
10	PostgreSQL Cheat Sheet	publisher	postgresqltutorial.com

4. PostgreSQL Array:

Query Editor Query History

```
CREATE TABLE contacts(
   id serial PRIMARY KEY,
   name VARCHAR (100),
   phones TEXT [ ]
);
INSERT INTO contacts
(name, phones)
VALUES
   ('John Doe',ARRAY [ '(408)-589-5846','(408)-589-5555' ]);
INSERT INTO contacts (name, phones)
VALUES
   ('Lily Bush','{"(408)-589-5841"}'),
   ('William Gate','{"(408)-589-5842","(408)-589-58423"}');
SELECT name,phones FROM contacts;
SELECT name,phones [ 1 ] FROM contacts;
SELECT name FROM contacts
WHERE phones [ 2 ] = '(408)-589-58423';
UPDATE contacts
SET phones [ 2 ] = '(408)-589-5843'
WHERE ID = 3;
SELECT id,name,phones [ 2 ]FROM contacts
WHERE id = 3;
UPDATE contacts
SET phones = '{"(408)-589-5843"}'
WHERE ID = 3;
SELECT name,phones FROM contacts
WHERE id = 3;
SELECT name,phones
FROM contacts
WHERE'(408)-589-5555' = ANY (phones);
SELECT name,unnest(phones)FROM contacts;
```

Data Output Explain Messages Notifications

Data Output Explain Messages Notifications

	name character varying (100)	unnest text
1	John Doe	(408)-589-5846
2	John Doe	(408)-589-5555
3	Lily Bush	(408)-589-5841
4	William Gate	(408)-589-5843

8.Conditional Expressions & Operators:

1.PostgreSQL COALESCE

```
CREATE TABLE items (
    ID serial PRIMARY KEY,
    product VARCHAR (100) NOT NULL,
    price NUMERIC NOT NULL,
    discount NUMERIC
);
INSERT INTO items (product, price, discount)
VALUES
    ('A', 1000 ,10),
    ('B', 1500 ,20),
    ('C', 800 ,5),
    ('D', 500, NULL);
SELECT product, (price - discount) AS net_price FROM items;

SELECT product,(price - COALESCE(discount,0)) AS net_price FROM items;

SELECT product,(price - CASE WHEN discount IS NULL THEN 0 ELSE discount END) AS net_price FROM items;
```

	product character varying (100)	net_price numeric
1	A	990
2	B	1480
3	C	795
4	D	[null]

```
CREATE TABLE items (
    ID serial PRIMARY KEY,
    product VARCHAR (100) NOT NULL,
    price NUMERIC NOT NULL,
    discount NUMERIC
);
INSERT INTO items (product, price, discount)
VALUES
    ('A', 1000 ,10),
    ('B', 1500 ,20),
    ('C', 800 ,5),
    ('D', 500, NULL);
SELECT product, (price - discount) AS net_price FROM items;

SELECT product,(price - COALESCE(discount,0)) AS net_price FROM items;

SELECT product,(price - CASE WHEN discount IS NULL THEN 0 ELSE discount END) AS net_price FROM items;
```

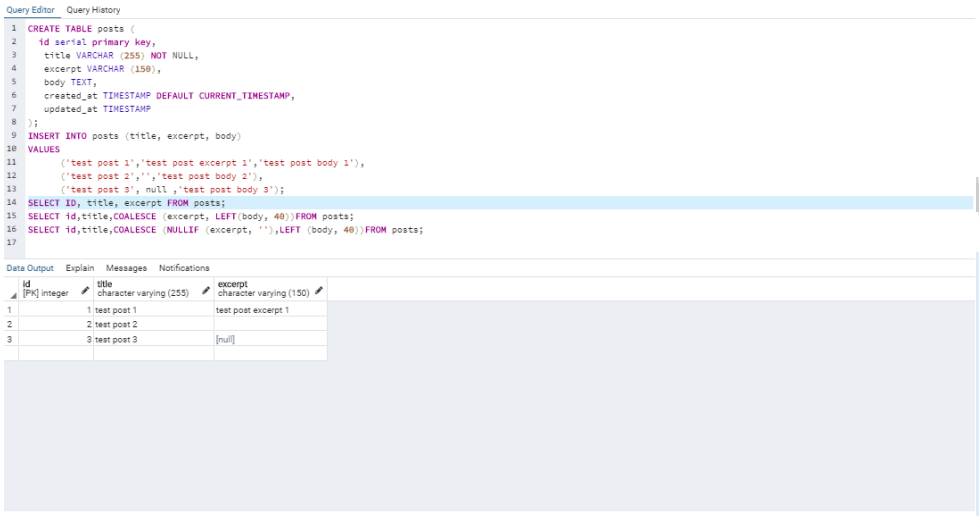
	product character varying (100)	net_price numeric
1	A	990
2	B	1480
3	C	795
4	D	500

```
CREATE TABLE items (
    ID serial PRIMARY KEY,
    product VARCHAR (100) NOT NULL,
    price NUMERIC NOT NULL,
    discount NUMERIC
);
INSERT INTO items (product, price, discount)
VALUES
    ('A', 1000 ,10),
    ('B', 1500 ,20),
    ('C', 800 ,5),
    ('D', 500, NULL);
SELECT product, (price - discount) AS net_price FROM items;

SELECT product,(price - COALESCE(discount,0)) AS net_price FROM items;

SELECT product,(price - CASE WHEN discount IS NULL THEN 0 ELSE discount END) AS net_price FROM items;
```

	product character varying (100)	net_price numeric
1	A	990
2	B	1480
3	C	795
4	D	500

2.PostgreSQL NULLIF .

PostgreSQL NULLIF function example

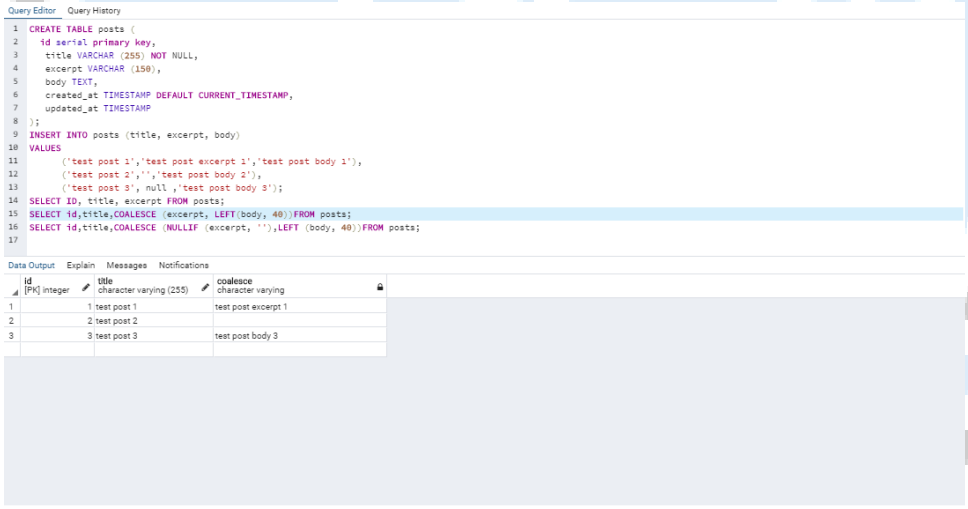


```
CREATE TABLE posts (
  id serial primary key,
   title VARCHAR (255) NOT NULL,
   excerpt VARCHAR (150),
   body TEXT,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   updated_at TIMESTAMP
);
INSERT INTO posts (title, excerpt, body)
VALUES
      ('test post 1','test post excerpt 1','test post body 1'),
      ('test post 2','','test post body 2'),
      ('test post 3', null ,'test post body 3');
SELECT ID, title, excerpt FROM posts;
SELECT id,title,COALESCE (excerpt, LEFT(body, 40))FROM posts;
SELECT id,title,COALESCE (NULLIF (excerpt, ''),LEFT (body, 40))FROM posts;
```

Query Editor | Query History

Data Output | Explain | Messages | Notifications

	id [PK] integer	title character varying (255)	excerpt character varying (150)
1	1	test post 1	test post excerpt 1
2	2	test post 2	
3	3	test post 3	[null]

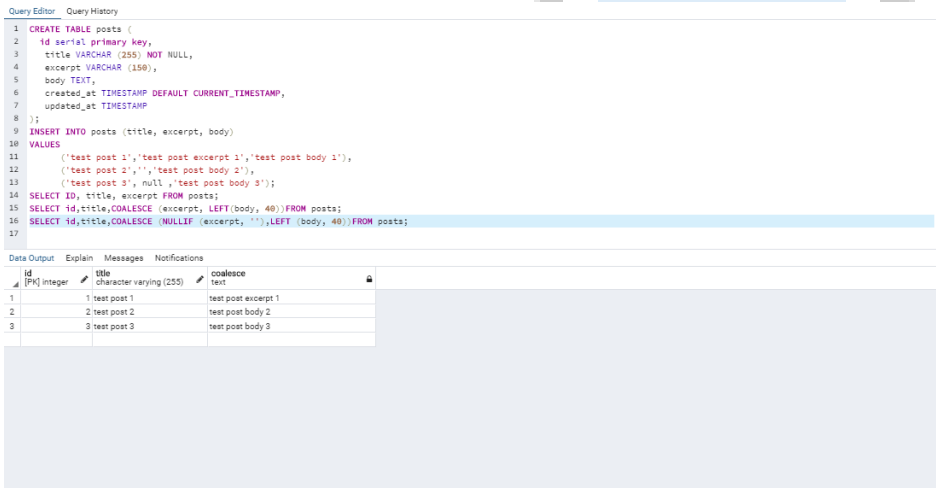


```
CREATE TABLE posts (
  id serial primary key,
   title VARCHAR (255) NOT NULL,
   excerpt VARCHAR (150),
   body TEXT,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   updated_at TIMESTAMP
);
INSERT INTO posts (title, excerpt, body)
VALUES
      ('test post 1','test post excerpt 1','test post body 1'),
      ('test post 2','','test post body 2'),
      ('test post 3', null ,'test post body 3');
SELECT ID, title, excerpt FROM posts;
SELECT id,title,COALESCE (excerpt, LEFT(body, 40))FROM posts;
SELECT id,title,COALESCE (NULLIF (excerpt, ''),LEFT (body, 40))FROM posts;
```

Query Editor | Query History

Data Output | Explain | Messages | Notifications

	id [PK] integer	title character varying (255)	coalesce character varying
1	1	test post 1	test post excerpt 1
2	2	test post 2	
3	3	test post 3	test post body 3



```
CREATE TABLE posts (
  id serial primary key,
   title VARCHAR (255) NOT NULL,
   excerpt VARCHAR (150),
   body TEXT,
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   updated_at TIMESTAMP
);
INSERT INTO posts (title, excerpt, body)
VALUES
      ('test post 1','test post excerpt 1','test post body 1'),
      ('test post 2','','test post body 2'),
      ('test post 3', null ,'test post body 3');
SELECT ID, title, excerpt FROM posts;
SELECT id,title,COALESCE (excerpt, LEFT(body, 40))FROM posts;
SELECT id,title,COALESCE (NULLIF (excerpt, ''),LEFT (body, 40))FROM posts;
```

Query Editor | Query History

Data Output | Explain | Messages | Notifications

	id [PK] integer	title character varying (255)	coalesce text
1	1	test post 1	test post excerpt 1
2	2	test post 2	test post body 2
3	3	test post 3	test post body 3

PostgreSQL type cast :: operator

```
SELECT
  '100'::INTEGER,'01-OCT-2015'::DATE;
```

int4 integer	date date
100	2015-10-01

PostgreSQL CAST examples

1) Cast a string to an integer example

```
SELECT
  '100'::INTEGER,'01-OCT-2015'::DATE;

SELECT CAST ('100' AS INTEGER);
SELECT CAST ('2015-01-01' AS DATE),CAST ('01-OCT-2015' AS DATE);

SELECT CAST ('10.2' AS DOUBLE PRECISION);

SELECT CAST('true' AS BOOLEAN),
   CAST('false' as BOOLEAN),
   CAST('T' as BOOLEAN),
   CAST('F' as BOOLEAN);

SELECT '2019-06-15 14:30:20'::timestamp;

SELECT '15 minute'::interval,
 '2 hour'::interval,
 '1 day'::interval,
 '2 week'::interval,
 '3 month'::interval;
```

int4 integer
100

2) Cast a string to a date example

```
SELECT CAST ('2015-01-01' AS DATE),CAST ('01-OCT-2015' AS DATE);

SELECT CAST ('10.2' AS DOUBLE PRECISION);

SELECT CAST('true' AS BOOLEAN),
   CAST('false' as BOOLEAN),
   CAST('T' as BOOLEAN),
   CAST('F' as BOOLEAN);

SELECT '2019-06-15 14:30:20'::timestamp;

SELECT '15 minute'::interval,
 '2 hour'::interval,
 '1 day'::interval,
 '2 week'::interval,
 '3 month'::interval;
```

date date	date date
2015-01-01	2015-10-01

3) Cast a string to a double example

Query Editor Query History

```
SELECT CAST ('2015-01-01' AS DATE),CAST ('01-OCT-2015' AS DATE);

SELECT CAST ('10.2' AS DOUBLE PRECISION);

SELECT CAST('true' AS BOOLEAN),
   CAST('false' as BOOLEAN),
   CAST('T' as BOOLEAN),
   CAST('F' as BOOLEAN);

SELECT '2019-06-15 14:30:20'::timestamp;

SELECT '15 minute'::interval,
 '2 hour'::interval,
 '1 day'::interval,
 '2 week'::interval,
 '3 month'::interval;
```

Data Output Explain Messages Notifications

	float8 double precision
1	10.2

4) Cast a string to a boolean example

Query Editor Query History

```
SELECT CAST ('2015-01-01' AS DATE),CAST ('01-OCT-2015' AS DATE);

SELECT CAST ('10.2' AS DOUBLE PRECISION);

SELECT CAST('true' AS BOOLEAN),
   CAST('false' as BOOLEAN),
   CAST('T' as BOOLEAN),
   CAST('F' as BOOLEAN);

SELECT '2019-06-15 14:30:20'::timestamp;

SELECT '15 minute'::interval,
 '2 hour'::interval,
 '1 day'::interval,
 '2 week'::interval,
 '3 month'::interval;
```

Data Output Explain Messages Notifications

	bool boolean	bool boolean	bool boolean	bool boolean
1	true	false	true	false

5) Convert a string to a timestamp example

Query Editor | Query History

```
SELECT CAST ('2015-01-01' AS DATE),CAST ('01-OCT-2015' AS DATE);

SELECT CAST ('10.2' AS DOUBLE PRECISION);

SELECT CAST('true' AS BOOLEAN),
   CAST('false' as BOOLEAN),
   CAST('T' as BOOLEAN),
   CAST('F' as BOOLEAN);

SELECT '2019-06-15 14:30:20'::timestamp;

SELECT '15 minute'::interval,
 '2 hour'::interval,
 '1 day'::interval,
 '2 week'::interval,
 '3 month'::interval;
```

Data Output | Explain | Messages | Notifications

	timestamp timestamp without time zone
1	2019-06-15 14:30:20

6) Convert a string to an interval example

Query Editor | Query History

```
SELECT CAST ('2015-01-01' AS DATE),CAST ('01-OCT-2015' AS DATE);


SELECT CAST ('10.2' AS DOUBLE PRECISION);

SELECT
   CAST('true' AS BOOLEAN),
   CAST('false' as BOOLEAN),
   CAST('T' as BOOLEAN),
   CAST('F' as BOOLEAN);

SELECT '2019-06-15 14:30:20'::timestamp;


SELECT '15 minute'::interval,
 '2 hour'::interval,
 '1 day'::interval,
 '2 week'::interval,
 '3 month'::interval;
```

Data Output | Explain | Messages | Notifications

	interval interval	interval interval	interval interval	interval interval	interval interval
1	00:15:00	02:00:00	1 day	14 days	3 mons

7) Using CAST with table data example

Query Editor Query History

```
CREATE TABLE ratings (
   ID serial PRIMARY KEY,
   rating VARCHAR (1) NOT NULL
);
INSERT INTO ratings (rating)
VALUES
   ('A'),
   ('B'),
   ('C');
INSERT INTO ratings (rating)
VALUES
   (1),
   (2),
   (3);
SELECT * FROM ratings;
SELECT id,
CASE WHEN rating~E'^\\d+$' THEN CAST (rating AS INTEGER) ELSE 0 END as rating FROM ratings;

```

Data Output Explain Messages Notifications

	id [PK] integer	rating character varying (1)
1	1	A
2	2	B
3	3	C
4	4	1
5	5	2
6	6	3

9.PostgreSQL Database Constraints :

1.PostgreSQL Primary Key

```
CREATE TABLE po_headers
(
   po_no INTEGER PRIMARY KEY,
   vendor_no INTEGER,
   description TEXT,
   shipping_address TEXT
);

CREATE TABLE po_items (
   po_no INTEGER,
   item_no INTEGER,
   product_no INTEGER,
   qty INTEGER,
   net_price NUMERIC,
   PRIMARY KEY (po_no, item_no)
);

--////////////////////////////////////////////////--
CREATE TABLE products (
   product_no INTEGER,
   description TEXT,
   product_cost NUMERIC
);
ALTER TABLE products
ADD PRIMARY KEY (product_no);
--////////////////////////////////////////////////--
```

```
--////////////////////////////////////////////--
CREATE TABLE vendors
(name VARCHAR(255));

INSERT INTO vendors
(NAME)
VALUES
   ('Microsoft'),
   ('IBM'),
   ('Apple'),
   ('Samsung');

SELECT * FROM vendors;

SELECT id,name FROM vendors;

ALTER TABLE products
DROP CONSTRAINT products_pkey;
```

name character varying (255)
1 Microsoft
2 IBM
3 Apple
4 Samsung

2. PostgreSQL CHECK Constraint

```
CREATE TABLE employees (
   id serial PRIMARY KEY,
   first_name VARCHAR (50),
   last_name VARCHAR (50),
   birth_date DATE CHECK (birth_date > '1900-01-01'),
   joined_date DATE CHECK (joined_date > birth_date),
   salary numeric CHECK(salary > 0)
);

INSERT INTO employees
(first_name,last_name,birth_date,joined_date,salary)
VALUES
('John','Doe','1972-01-01','2015-07-01',100000);

select * from employees
```

	id [PK] integer	first_name character varying (50)	last_name character varying (50)	birth_date date	joined_date date	salary numeric
1	5	John	Doe	1972-01-01	2015-07-01	100000

3. PostgreSQL UNIQUE Constraint

```
CREATE TABLE equipment (
   id serial PRIMARY KEY,
   name VARCHAR (50) NOT NULL,
   equip_id VARCHAR (16) NOT NULL
);

CREATE UNIQUE INDEX CONCURRENTLY equipment_equip_id
ON equipment (equip_id);

ALTER TABLE equipment
ADD CONSTRAINT unique_equip_id
UNIQUE USING INDEX equipment_equip_id;

SELECT datid,datname,usename, state FROM pg_stat_activity;
```

	datid oid	datname name	usename name	state text
1	[null]	[null]	postgres	[null]
2	[null]	[null]	[null]	[null]
3	13318	postgres	postgres	idle
4	13318	postgres	postgres	idle
5	13318	postgres	postgres	active
6	[null]	[null]	[null]	[null]
7	[null]	[null]	[null]	[null]
8	[null]	[null]	[null]	[null]

4.PostgreSQL Not-Null Constraint

```
CREATE TABLE production_orders (
    ID serial PRIMARY KEY,
    description VARCHAR (40) NOT NULL,
    material_id VARCHAR (16),
    qty NUMERIC,
    start_date DATE,
    finish_date DATE
);
INSERT INTO production_orders (description)
VALUES('make for infosys inc.');
UPDATE production_orders
SET qty = 1;

ALTER TABLE production_orders ALTER COLUMN qty
SET NOT NULL;
UPDATE production_orders
SET material_id = 'ABC',start_date = '2015-09-01',finish_date = '2015-09-01';
SELECT * FROM production_orders
```

id [PK] integer	description character varying (40)	material_id character varying (16)	qty numeric	start_date date	finish_date date
1	make for infosys inc.	ABC	1	2015-09-01	2015-09-01

10.PostgreSQL NTILE Function, LAG Function, LEAD Function, NTH_VALUE Function,

1.PostgreSQL NTILE Function

```
CREATE TABLE saless(
year SMALLINT CHECK(year > 0),
    group_id INT NOT NULL,
    amount DECIMAL(10,2) NOT NULL,
    PRIMARY KEY(year,group_id)
);
    INSERT INTO
    saless(year, group_id, amount)
VALUES
    (2018,1,1474),
    (2018,2,1787),
    (2018,3,1760),
    (2019,1,1915),
    (2019,2,1911),
    (2019,3,1118),
    (2020,1,1646),
    (2020,2,1975),
    (2020,3,1516);
SELECT * FROM saless;
```

year [PK] smallint	group_id [PK] integer	amount numeric (10,2)
2018	1	1474.00
2018	2	1787.00
2018	3	1760.00
2019	1	1915.00
2019	2	1911.00
2019	3	1118.00
2020	1	1646.00
2020	2	1975.00
2020	3	1516.00

2. PostgreSQL LAG Function

```
WITH cte AS (
    SELECT year,SUM(amount) amount FROM saless
    GROUP BY year
    ORDER BY year)
SELECT year,amount,LEAD(amount,1) OVER (ORDER BY year) next_year_sales FROM cte;
```

year smallint	amount numeric	next_year_sales numeric
2018	5021.00	4944.00
2019	4944.00	5137.00
2020	5137.00	[null]

3. PostgreSQL LEAD Function

```
SELECT year,amount,group_id,
LEAD(amount,1) OVER (PARTITION BY group_id ORDER BY year) next_year_sales FROM salesa;
```

year [PK] smallint	amount numeric (10,2)	group_id [PK] integer	next_year_sales numeric
2018	1474.00	1	1915.00
2019	1915.00	1	1646.00
2020	1646.00	1	[null]
2018	1787.00	2	1911.00
2019	1911.00	2	1975.00
2020	1975.00	2	[null]
2018	1760.00	3	1118.00
2019	1118.00	3	1516.00
2020	1516.00	3	[null]

4. PostgreSQL NTH_VALUE Function :

```
SELECT id,name,price,NTH_VALUE(name, 2)
OVER
( ORDER BY price DESC RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)FROM  products;
```

id integer	name character varying	price numeric (5,2)	nth_value character varying
1	Phone	NaN	Tablet
2	Tablet	500.21	Tablet

5. PostgreSQL FIRST_VALUE Function

```
SELECT
    id,
    name,
   id,
    price,
    FIRST_VALUE(name)
    OVER(
   PARTITION BY id
        ORDER BY price
        RANGE BETWEEN
            UNBOUNDED PRECEDING AND
            UNBOUNDED FOLLOWING
    ) highest_price
FROM
    products;
```

id integer	name character varying	id integer	price numeric (5,2)	highest_price character varying
1	Phone	1	NaN	Phone
2	Tablet	2	500.21	Tablet

11: Object Oriented Database (Inheritance)

Query Editor Query History

```
CREATE SEQUENCE "component_component_id_seq"  START 1;

CREATE TABLE "component" (
  "component_id" int4 DEFAULT nextval('component_component_id_seq') NOT NULL,
  "component_name" varchar(50),
  "ip" inet,
  "po" varchar(20),
  "mac" macaddr,
  "department" varchar(20),
  CONSTRAINT "component_pkey" PRIMARY KEY ("component_id")
);


CREATE TABLE "computer" (
  "os" varchar(20)
) INHERITS("component");

UPDATE component
SET department='a good department'
WHERE ip << inet '192.168.1/24';

   SELECT * FROM component;

```

Data Output Explain Messages Notifications

component_id [PK] integer	component_name character varying (50)	ip inet	po character varying (20)	mac macaddr	department character varying (20)

12: Object Oriented Querying (jOOQ) with PostgreSQL

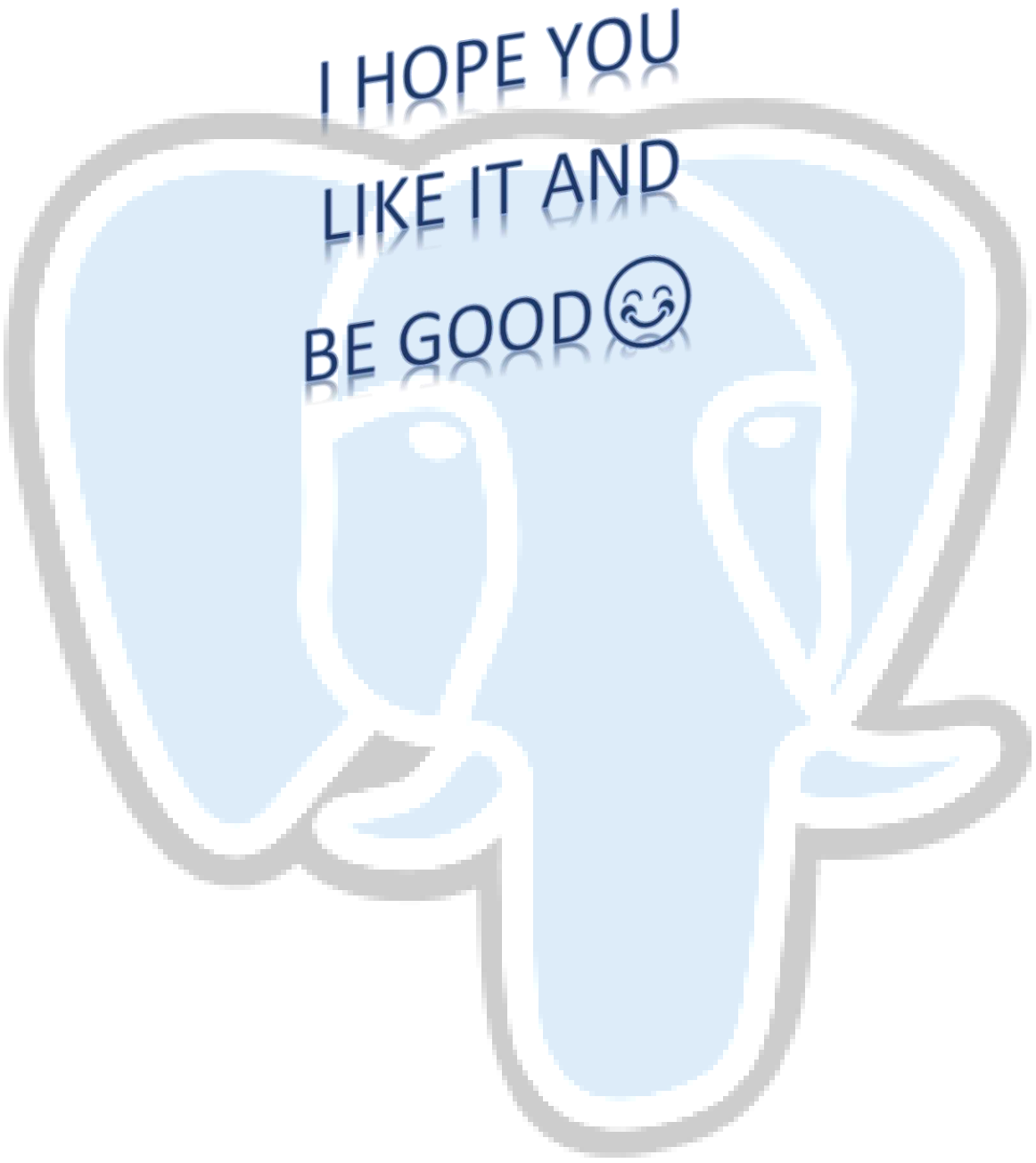
Start Page Postgresql.java

Source History

```
package postgresql;

import java.sql.*;

public class Postgresql {

    public static void main(String[] args) {
        String userName = "postgres";
        String password = "2";
        String url = "jdbc:postgresql://localhost:5432/postgres";
        try  {
            Connection conn = DriverManager.getConnection(url, userName, password);
                System.out.println("Connect OK");
        }
       catch (Exception e) {
         e.printStackTrace();
    }
}
}
```

Output - Postgresql (run)

```
run:
Connect OK
BUILD SUCCESSFUL (total time: 0 seconds)
```


I HOPE YOU
LIKE IT AND
BE GOOD