

PostgreSQL初学者入门

以下所有笔记均学习自[【课程】PostgreSQL初学者入门](#)

如有兴趣了解更详细内容，请查看原网址

你也可以参考如下网站的教学内容 [PostgreSQL Tutorial](#)

Chapter 1 Introduction

The main content includes

- How to install PostgreSQL Server
- How to create a Database
- How to create a Table
- How to insert data into a Table
- How to update records in a Table
- How to get data from multiple tabbles with JOINS
- How to GROUP Data
- How to create complex queries using Subqueries
- How to truncate and drop a table
- How to implement constraints
- How to remove duplicate data

This course is based on the video, it takes almost 5 hours.

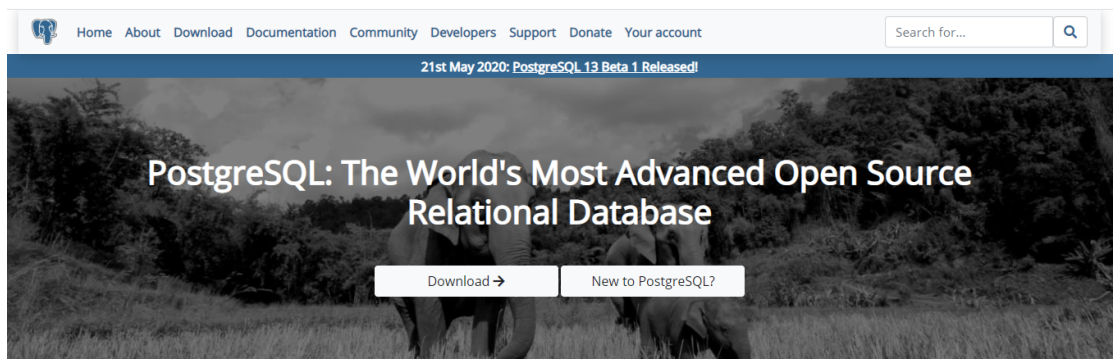
Chapter 2 PostgreSQL Installation Requirements

The minimum requirement for the PC you can check the following tables:

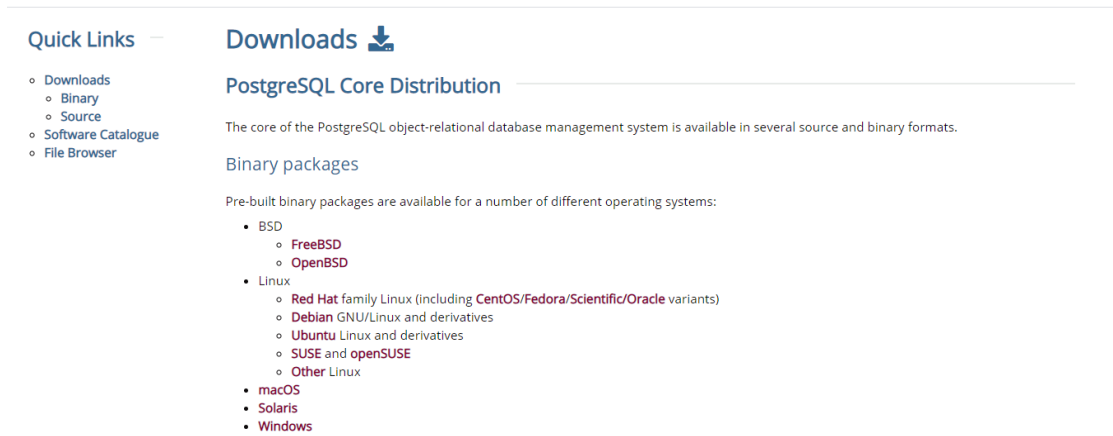
Requirements Overview		
Windows (32b & 64 bit)	Linux(32 & 64bit)	Mac OS X
Window 7,8,10	CentOS 6.X and 7	OS X Server 10.8
Windows 2008 Server	Ubuntu 14.04	OS X Server 10.9
Windows 2012	Debian 7 and 8	OSX Server 10.10
1GHZ Processor	Oracle Enterprise Linux 6 &7	
1 GB RAM	1GHZ Processor	1GHZ Processor
1GB HDD	1 GB RAM	1 GB RAM
Account with admin role needed	Super user privileges required	Super user privileges required

Chapter 3 Download PostgreSQL for windows

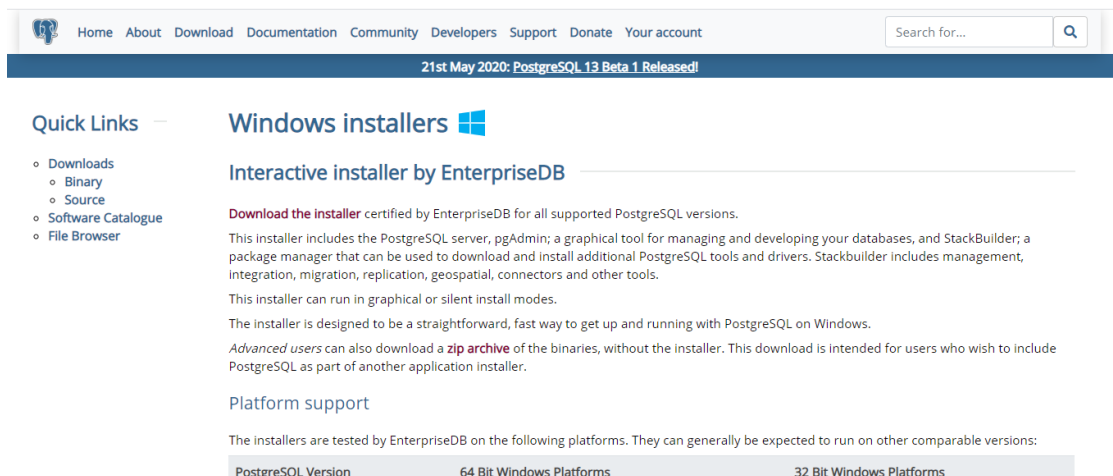
1. Go to the PostgreSQL office web site. [PGSQL OFFICE SITE](#)
2. Click the button "Download" then you will go to the download page



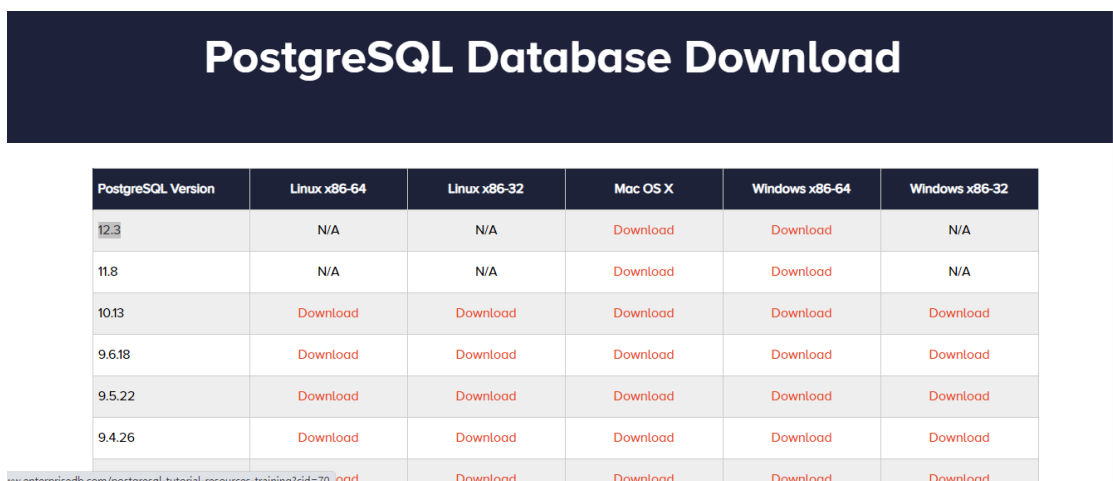
3. in the download page, you can see all kind of version for the PostgreSQL, because we are the WIn 10 , so we click the "Windows"



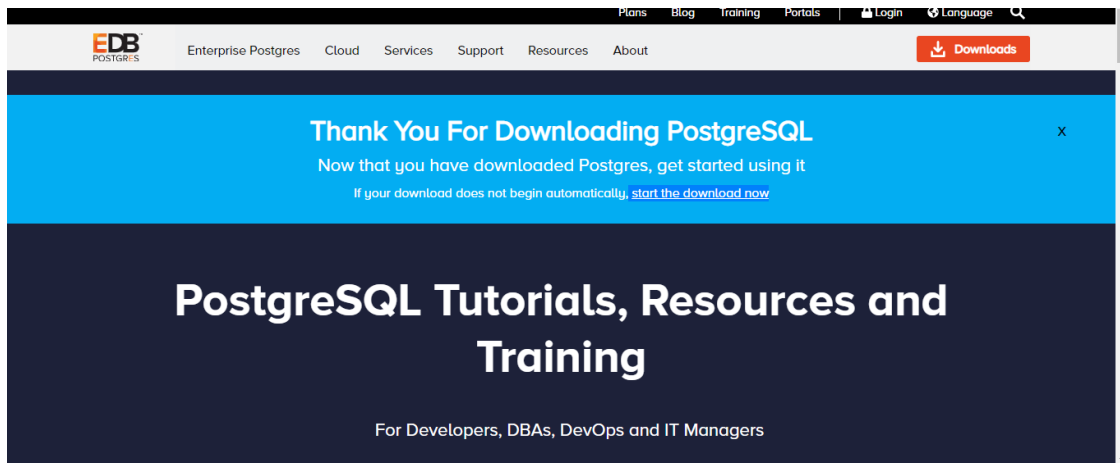
4. Then we will go the the WIn10 installer page, click "Download the installer"



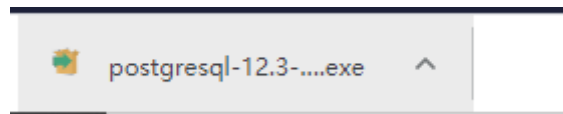
5. Then you can see many different version for the windows and other kinds of system, we choose the latest version for the windows **Windows x86-64** so it should be version **12.3** and then click "Download"



6. if the download doesn't work, please click the Hyperlinks "start the download now" then the download will start.



7. At last, we will find the download of the PostSQL is successful.



Chapter 4 Installing PostgreSQL For Windows

For installing PostgreSQL for windows, you just need to run the install package through a account with administrator rights.

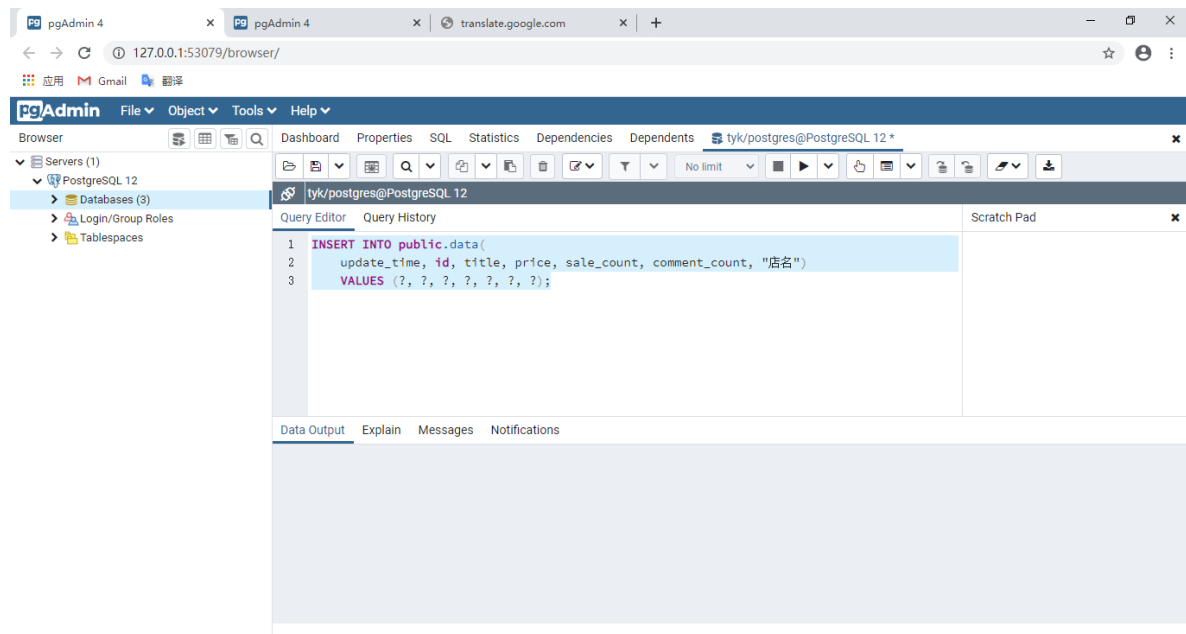
and then you only need to click click and click , all configuration are kept default.

there is a step to let you to input the account and the password, you should remember the information you type in. don't forget this, later you will use this account to logon in.

Chapter 5 Verify PostgreSQL installation for Windows

Note: because you install PostgreSQL with an account with administrator rights. so you need to run the pgAdmin in administration account.

The PostgreSQL version 12.3 server runs in a Web page, when you start the pgadmin, you will redirect to the page , if you can open the following page, which mean you install PostgreSQL successfully.



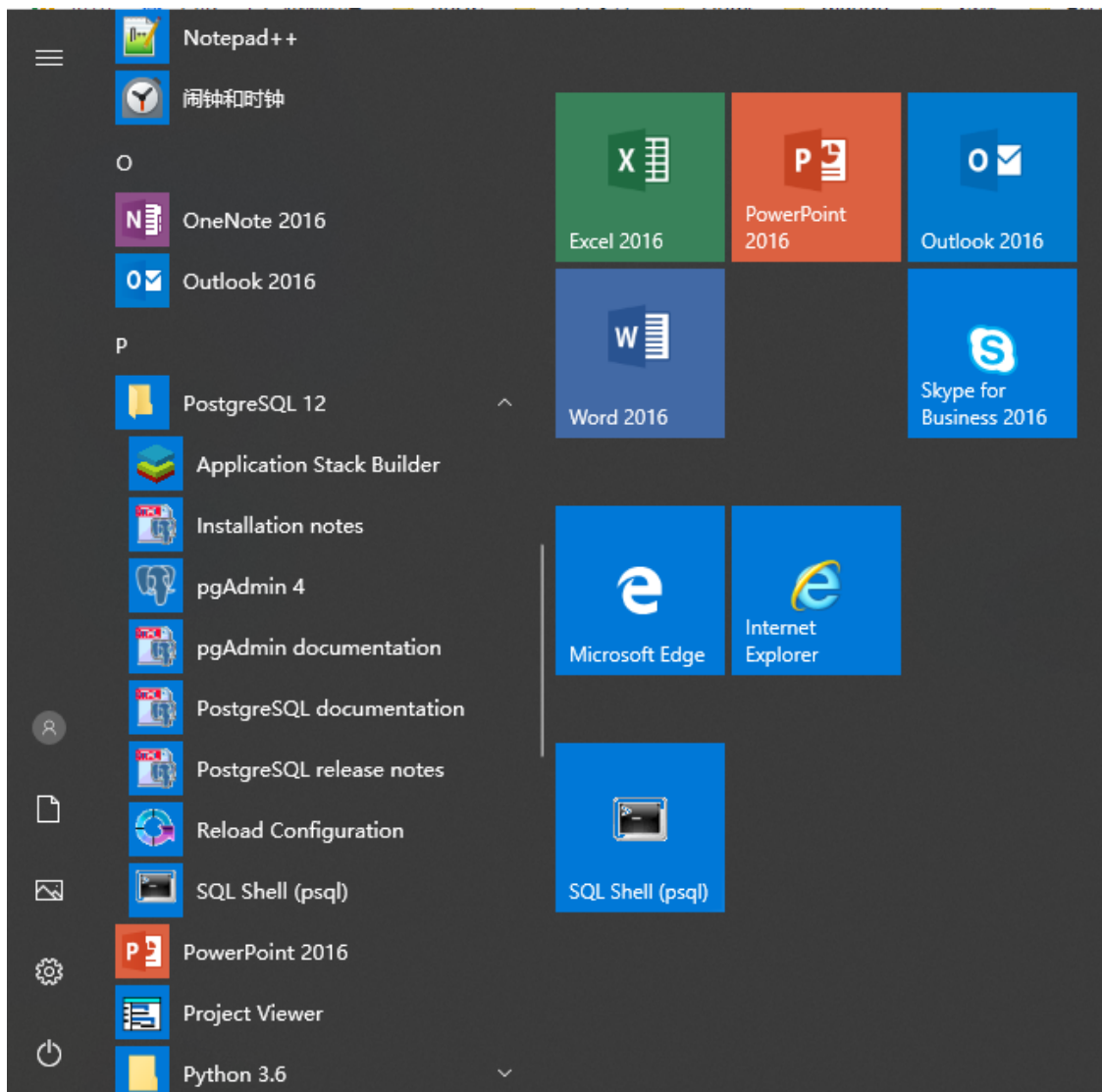
Chapter 6 Connection to a PostgreSQL Database

You can use following way to connect to a PostgreSQL database

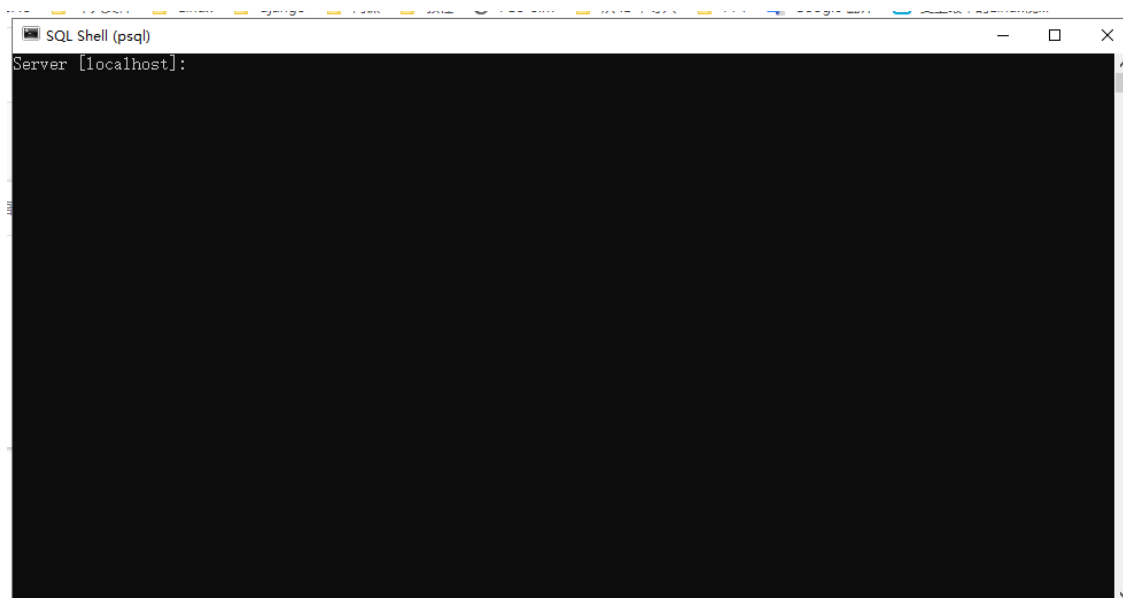
- SQL Shell (Psql)
- pgAdmin GUI (this tool was installed when PostgreSQL was installeds)

Connection with Psql

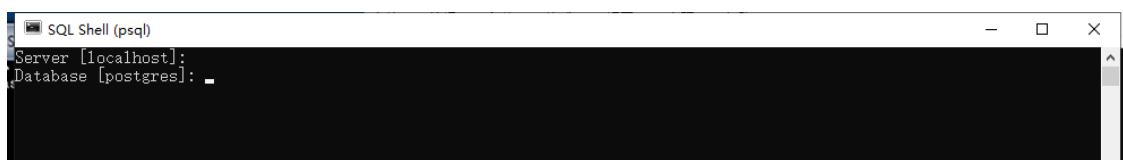
1. Open the installment folder for the PostgreSQL at the Windows Start Meau



2. Then the SQL Shell launched.



3. You need to input the server name, the default one is the localhost, if you want to select the localhost, just input "Enter"

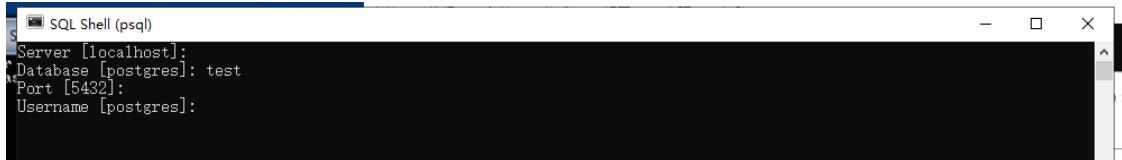


4. Then you need to input the database base, the default one is the postgres, if you want to select the localhost, jusy input "Enter", For example, i have craeted database "test", so i input the test.



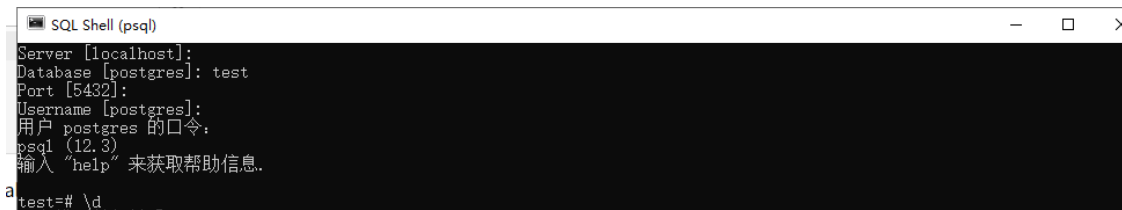
```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: test
Port [5432]:
```

5. Then you need to input the port number, keep the default.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: test
Port [5432]:
Username [postgres]:
```

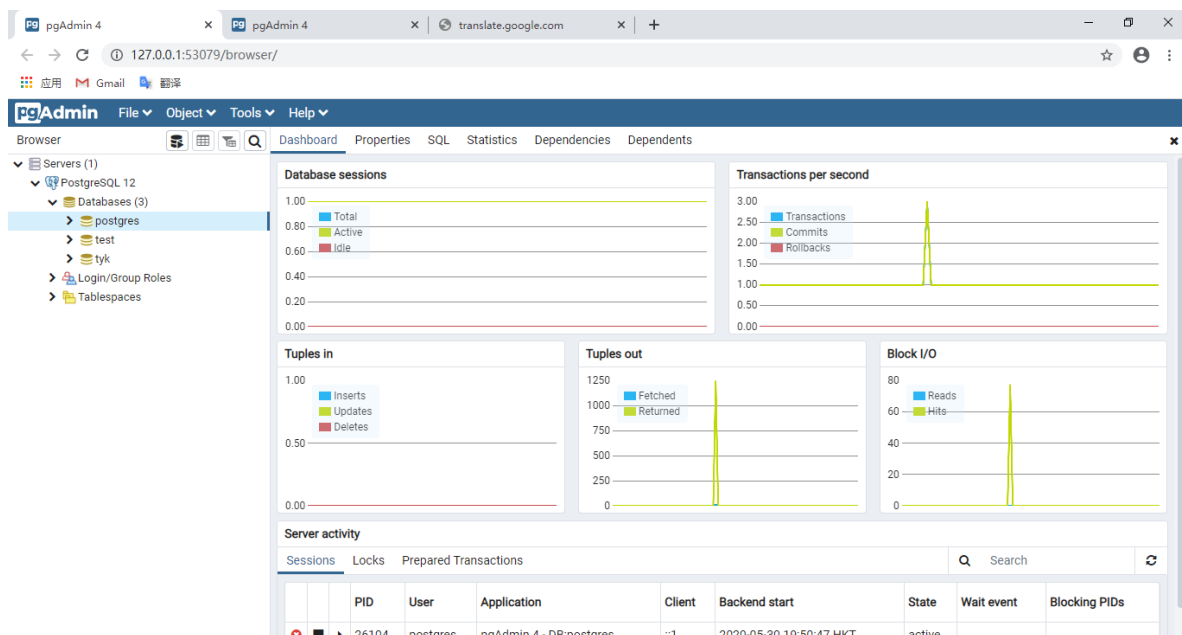
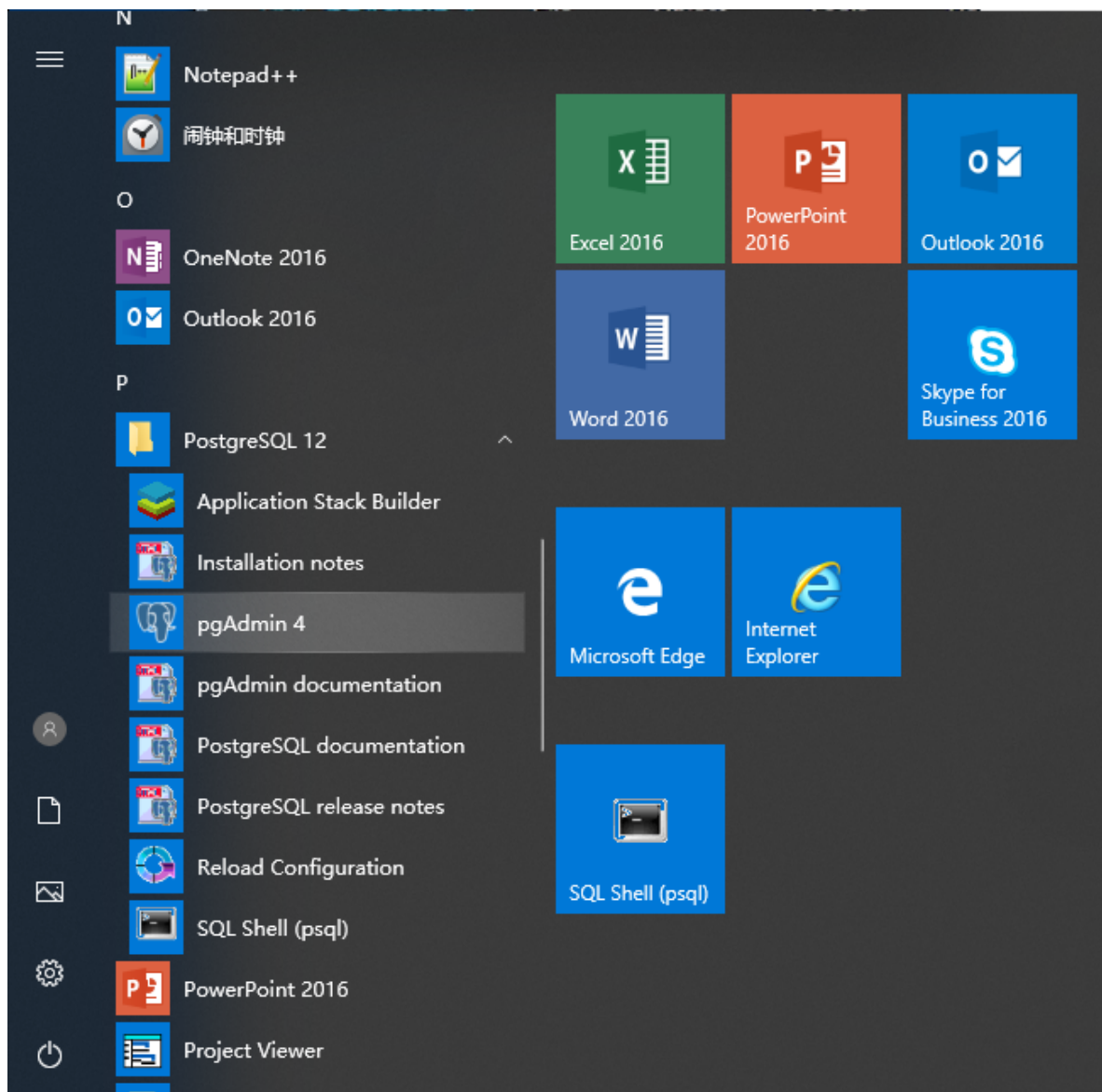
6. Then you need to input the username and the password, please use the information you input when you install the PostgreSQL. if you can see the follpwing information and enter a command mode, you succeed to connect.



```
SQL Shell (psql)
Server [localhost]:
Database [postgres]: test
Port [5432]:
Username [postgres]:
用户 postgres 的口令:
psql (12.3)
输入 "help" 来获取帮助信息.
test=# \d
```

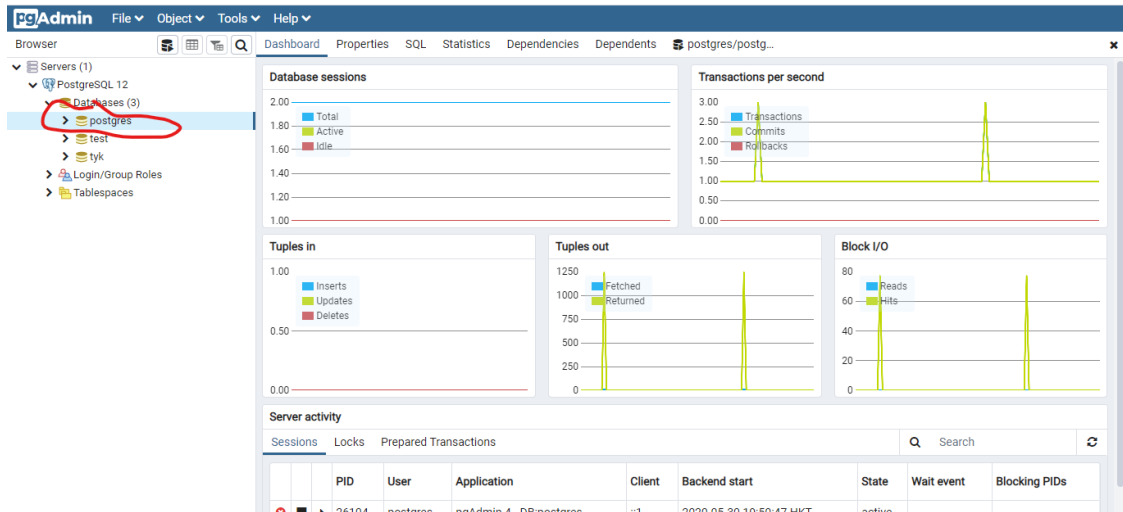
Connection with PgAdmin GUI

You should launch the PgAdmin 4 with an administrator account, and then you can the PostgreSQL Management Web Page.

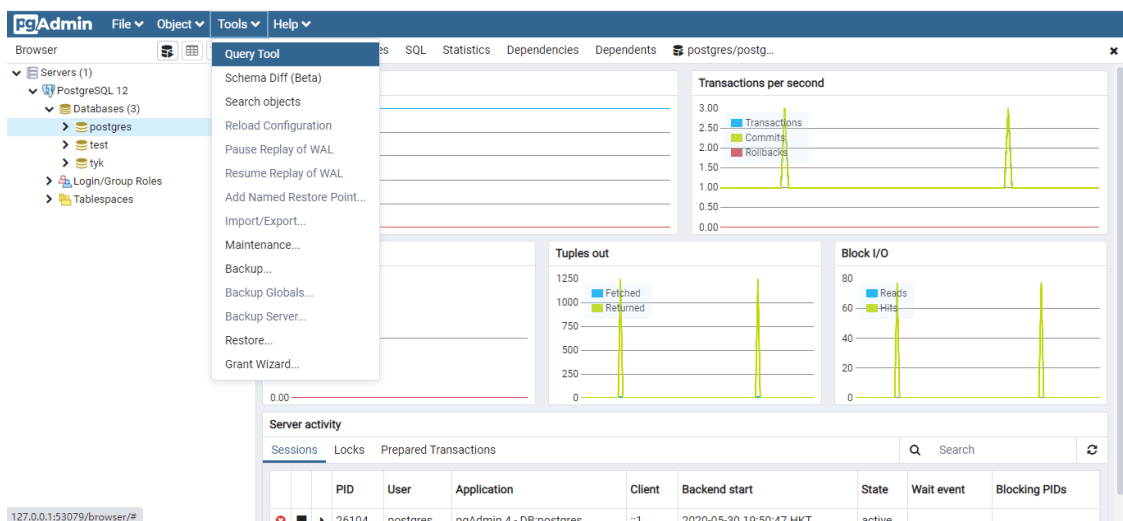


The following steps show you how to run a SQL script in the PgAdmin GUI

1. Select the database you want to query data, like input `\c databaseName` in SQL Shell, for example I choose the default postgres.

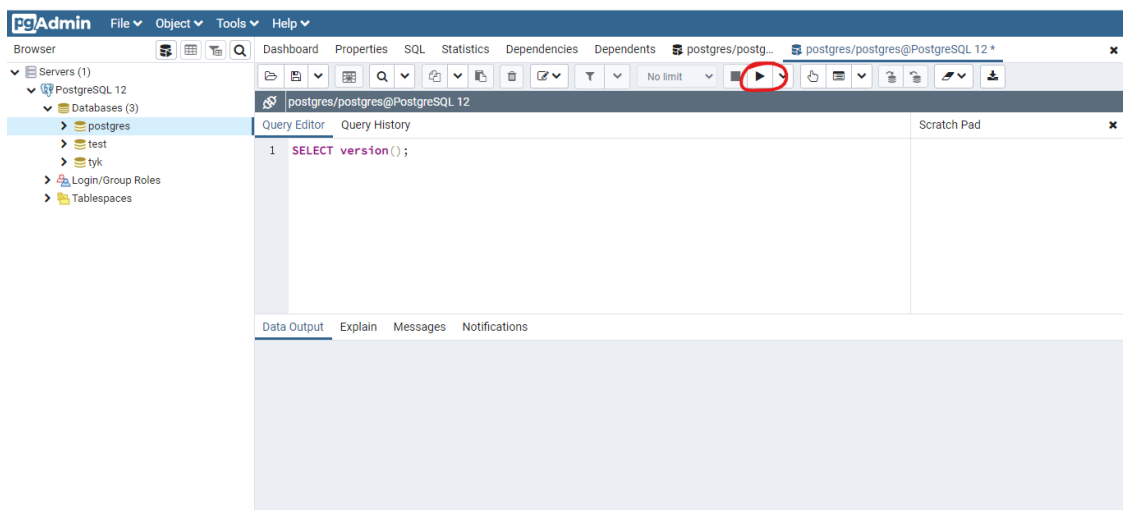


2. Then click Tools -> Query Tool

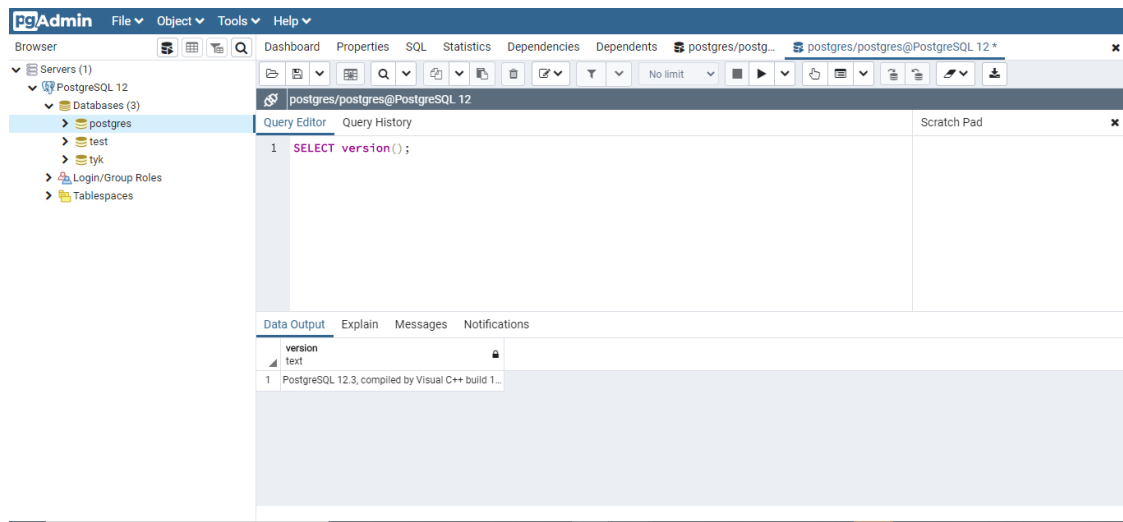


3. Then you enter the SQL script console. you can input some simple SQL script, for example I input the command to check the PostgreSQL version

```
SELECT version();
```



4. Then click the button "Run", you can see the query result in the Data Output.



Chapter 7 Download PostgreSQL Database for Mac

we skip this chapter.

Chapter 8 What is the PostgreSQL

PostgreSQL is pronounced "post-gres".

- ANSI-standard SQL
- General purpose
- Object-relational database management system (RDBMS)
- Free and Open source
- Platform independent
- Very stable and requires minimum maintenance
- Large storage
- Fast performance
- PostgreSQL is extensible : You can define your own data types, index types, plugins etc
- Active community support

Chapter 9 Who is using PostgreSQL

We just need to know that The PostgreSQL is widely used. it is enough.

Chapter 10 Loading Sample Database

DVD Rental Database

- 15 Tables
- 1 Trigger
- 7 Views
- 8 Functions
- 1 domain
- 13 Sequences

The Zip file download path [PostgreSQL Sample Database](#)

In this tutorial, we will show you how to load the **PostgreSQL sample database** into the PostgreSQL database server.

create the database

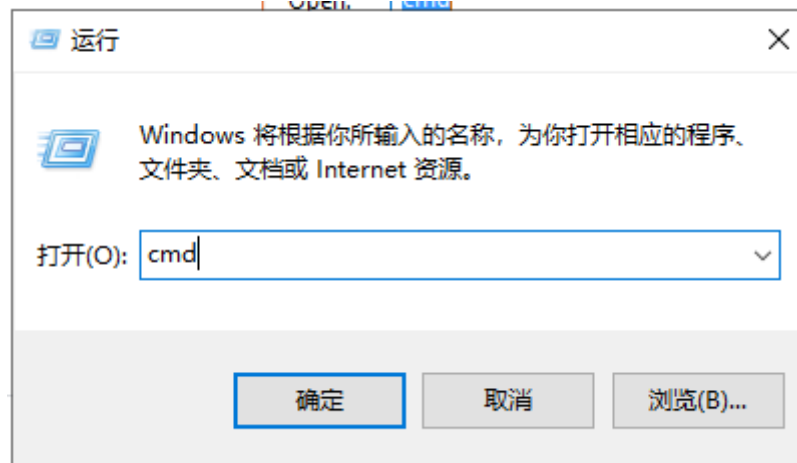
```
CREATE DATABASE dvdrental;
```

Load the DVD rental database using psql tool

Firstly First, unzip and copy the DVD rental database file to a folder e.g.,

c:\dvdrental\dvdrental.tar

Next, launch the **Command Prompt** program by using the keyboard shortcut **Windows + R**, type **cmd**, and press **Enter**:



Then, navigate the **bin** folder of the PostgreSQL installation folder:

```
C:\Program Files\PostgreSQL\12>cd bin
C:\Program Files\PostgreSQL\12\bin>dir
驱动器 C 中的卷是 Windows
卷的序列号是 A221-B7E9

C:\Program Files\PostgreSQL\12\bin 的目录
2020/05/30  19:28    <DIR>          .
2020/05/30  19:28    <DIR>          ..
2020/05/12  14:23             99,328 clusterdb.exe
2020/05/12  14:23             98,816 createdb.exe
2020/05/12  14:23            100,864 createuser.exe
2020/05/12  14:23             95,744 dropdb.exe
2020/05/12  14:23             95,744 dropuser.exe
2020/05/12  14:23            884,224 ecpg.exe
2020/05/12  14:23          21,529,088 icudt53.dll
2020/05/12  14:23          1,844,224 icuin53.dll
2020/05/12  14:23             50,688 icuioc53.dll
2020/05/12  14:23             272,896 icule53.dll
2020/05/12  14:23             54,784 iculx53.dll
2020/05/12  14:23             70,144 icuuc53.dll
```

After that, use the **pg_restore** tool to load data into the **dvdrental** database:

```
C:\Program Files\PostgreSQL\12\bin>pg_restore -U postgres -d dvdrental C:\Users\HD.huanghf\Downloads\dvdrental.tar
命令:
C:\Program Files\PostgreSQL\12\bin>
```

```
pg_restore -U postgres -d dvdrental C:\Users\HD.huanghf\Downloads\dvdrental.tar
```

In this command:

The `-U postgres` specifies the postgresuser to login to the PostgreSQL database server.

The `-d dvdrental` specifies the target database to load.

then you can check if the installment is successful

```
# 1. login the database via shell
# 2. check the database set
\l
# 3. change to the database dvdrental
\c dvdrental;
# 4. check the tables of the dvdrental
\d
```

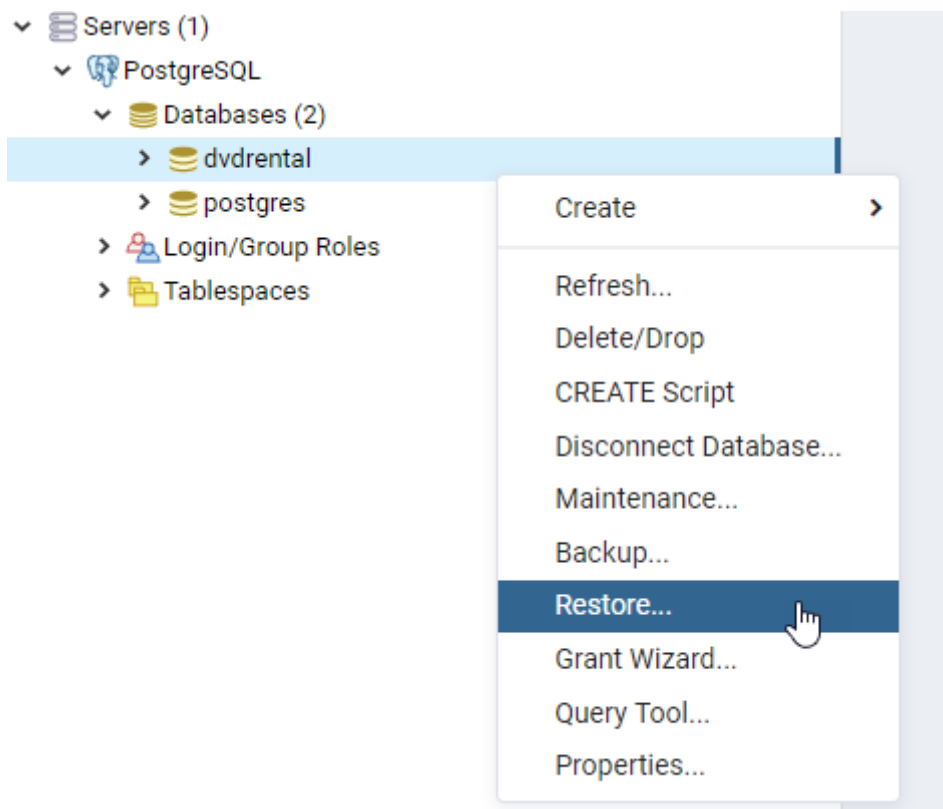
```
dvdrental=# \d
```

架构模式	名称	类型	所有者
public	actor	数据表	postgres
public	actor_actor_id_seq	序列数	postgres
public	actor_info	视图	postgres
public	address	数据表	postgres

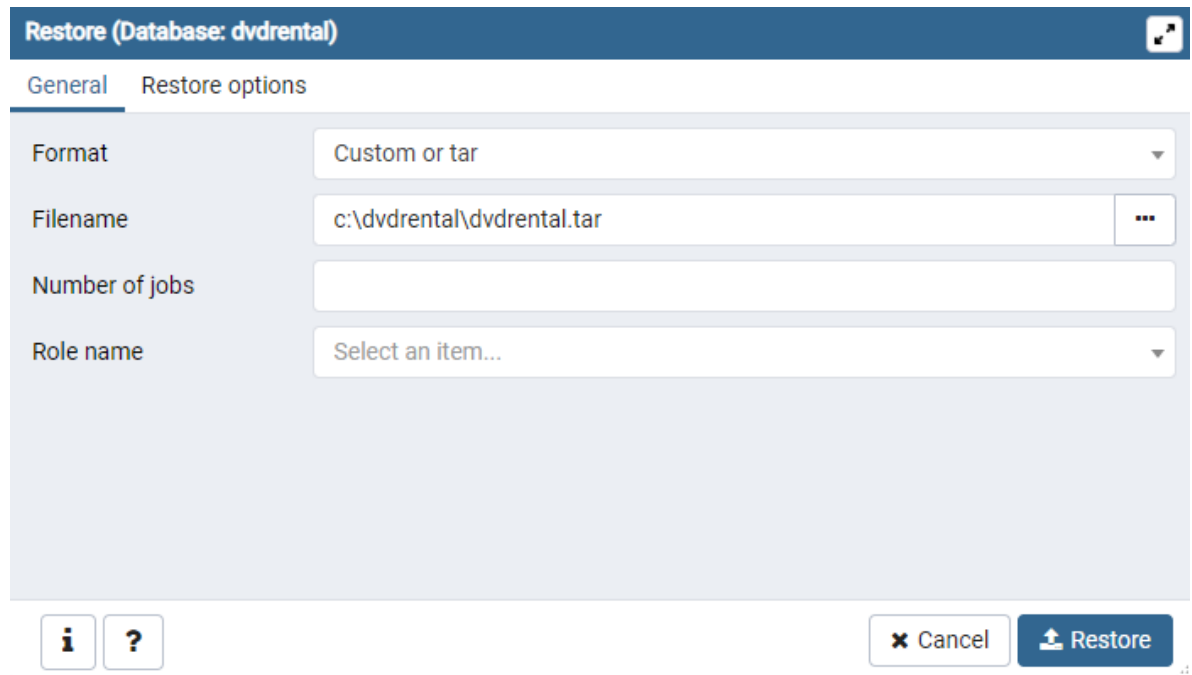
Load the DVD Rental database using the pgAdmin

First, launch the **pgAdmin** tool and connect to the PostgreSQL server.

Next, right-click on the **dvdrental** database and choose **Restore...** menu item as shown in the following picture:



Then, provide the path to database file e.g., **c:\dvdrental\dvdrental.tar** and click the **Restore** button



Restore (Database: dvdrental)

General | Restore options

Format: Custom or tar

Filename: c:\dvdrental\dvdrental.tar

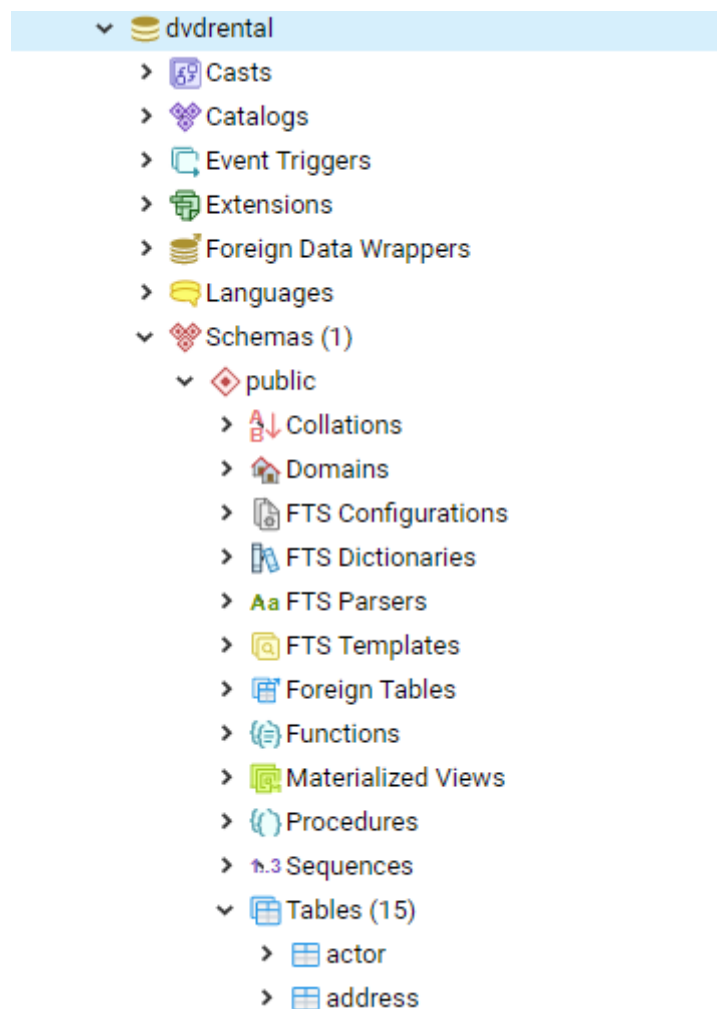
Number of jobs:

Role name: Select an item...

i ? x Cancel Restore

After that, wait for a few seconds to let the restoration process completes.

Finally, open the `dvdrental1` database from object browser panel, you will see the tables in the `public` schema and other database objects as shown in the following picture:



Chapter 11 introduction PostgreSQL Server and Database Objects

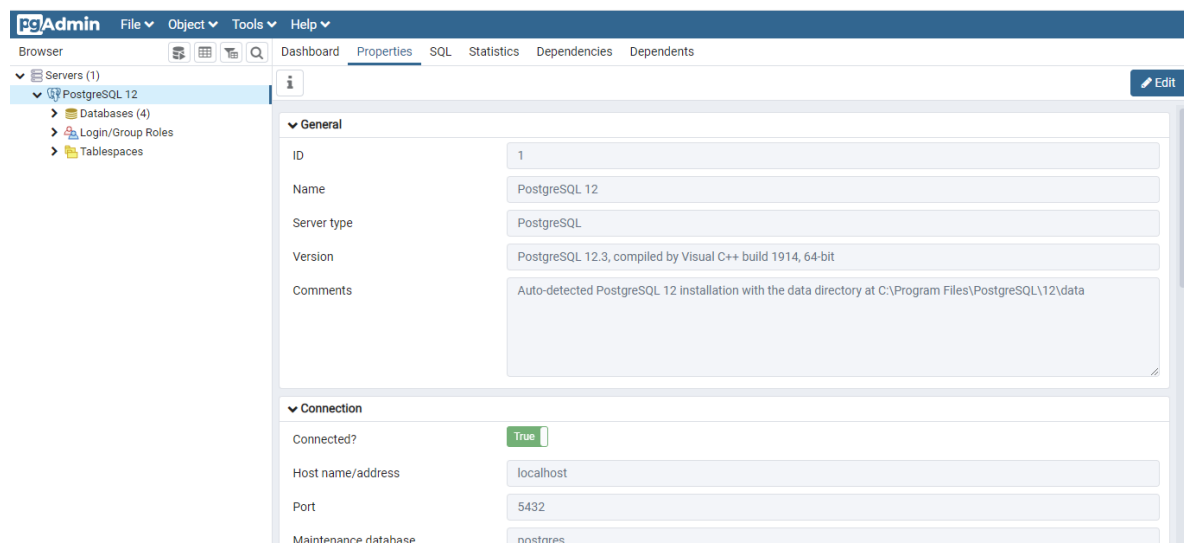
This chapter contains following contents:

- Server Service
- Database
- Table
- Schema
- Tablespace
- View
- Function
- Operator
- Cast
- Sequence
- Extension

Chapter 11 Introduction Server Service

You can install multiple server with different port and different data location in a physical machine

About the server properties, you can check the following picture,



Chapter 12 Introduction Database

The database is a container of all objects, such as tables, views, etc.

Chapter 13 Introduction Table

This chapter show how to check data from the table.

```
SELECT *  
FROM actor  
order by first_name  
limit 10;
```

Chapter 13 Introduction Schema

Schema container is the logical container of tables and other database objects

You can have multiple schemes in a database

public means everyone can access this schema.

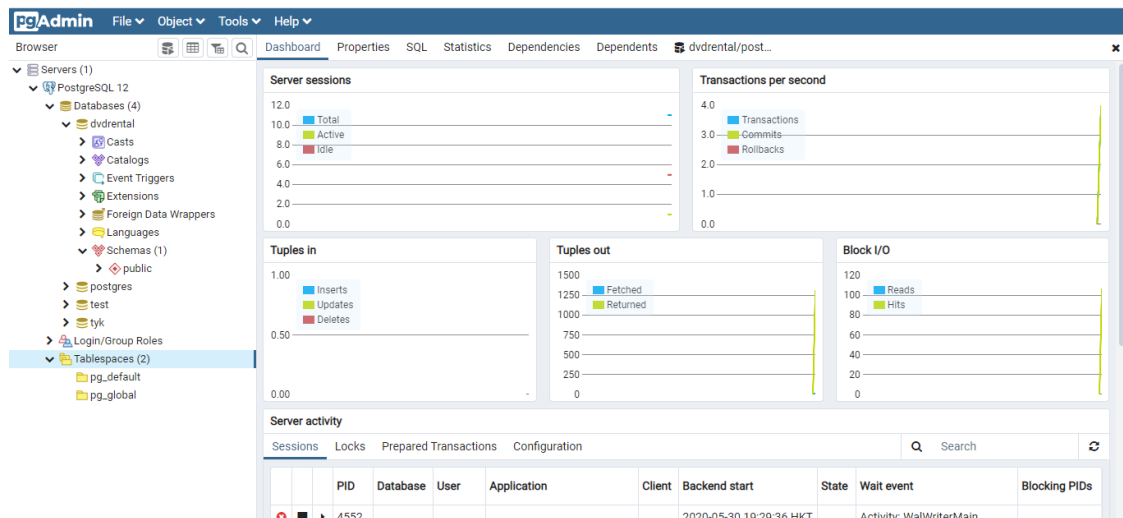
Schema can group the objects and assign the permission to the related person

Chapter 14 Introduction Tablespace

Storage location where data is stored

PostgreSQL provides two tablespaces by default

- Pg_default store user data
- Pg_global system data



Chapter 15 Introduction Views

Views are virtual tables, to finish complicate queries.

- Virtual table
- Simplifies complex queries
- Used to apply security for a set record

Chapter 16 Introduction Functions

- Block of reusable SQL code
- Return scalar value of a list of records
- return composite objects

Chapter 17 Introduction Cast and Operators

Cast

- cast: convert one data type into another data type
- Used with functions to perform conversion
- You can create custom casts to override the default

Operator

- Operator is a symbolic function

- You can define your own custom operators

Chapter 18 Introduction Sequences

- Used to manage auto-increment columns

Chapter 19 Introduction Extension

- Used to wrap other objects into a single unit;
- Casts
- Indexes
- Functions
- Etc

Chapter 23 PostgreSQL DATA Types

- Boolean
- Character
- Number
- Temporal
- Array
- Special Types

Boolean Type

A [Boolean](#) data type can hold one of three possible values: true, false or null. You use `boolean` or `bool` keyword to declare a column with the Boolean data type.

When you [insert data](#) into a Boolean column, PostgreSQL converts it to a Boolean value

- `1`, `yes`, `y`, `t`, `true` values are converted to `true`
- `0`, `no`, `false`, `f` values are converted to `false`.

When you [select data](#) from a Boolean column, PostgreSQL converts the values back e.g., `t` to `true`, `f` to `false` and `space` to `null`.

Introduction to the PostgreSQL Boolean type

PostgreSQL supports a single Boolean data type `BOOLEAN` that can have three states: TRUE, FALSE, and NULL. PostgreSQL uses one byte for storing a boolean value in the database. The `BOOLEAN` can be abbreviated as `BOOL`.

In standard SQL, a Boolean value can be TRUE, FALSE, or NULL. However, PostgreSQL is quite flexible when dealing with TRUE and FALSE values. The following table shows the valid literal values for TRUE and FALSE in PostgreSQL.

True	False
true	false
't'	'f'
'true'	'false'
'y'	'n'
'yes'	'no'
'1'	'0'

PostgreSQL Boolean examples

First, to create a new table `stock_availability`, to log which product are available.

```
CREATE TABLE stock_availability (
    product_id INT NOT NULL PRIMARY KEY,
    available BOOLEAN NOT NULL
);
```

Second, insert some sample data into the `stock_availability` table. We use various literal value for the boolean values.

```
INSERT INTO stock_availability (product_id, available)
VALUES
    (100, TRUE),
    (200, FALSE),
    (300, 't'),
    (400, '1'),
    (500, 'y'),
    (600, 'yes'),
    (700, 'no'),
    (800, '0');
```

Third, to check the products that are available, you use the following statement:

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

SELECT *
FROM stock_availability
WHERE available;
-----

SELECT *
FROM stock_availability
WHERE available = 'no';
-----

SELECT *
FROM stock_availability
WHERE NOT available;
```

Set a default value of the Boolean column

For example, the following `ALTER TABLE` statement set the default value for the `available` column in the `stock_availability` table:

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

If you insert a row without specifying the value for the `available` column, PostgreSQL uses the `FALSE` value by default.

```
INSERT INTO stock_availability (product_id)
VALUES
    (900);
SELECT
    *
FROM
    stock_availability
WHERE
    product_id = 900;
product_id | available
-----+-----
      900 | f
(1 row)
```

Similarly, if you want to set a default value for a Boolean column when you [create a table](#), you use the `DEFAULT` clause in the column definition as follows:

```
CREATE TABLE boolean_demo(
    ...
    is_ok BOOL DEFAULT 't'
);
```

Character

PostgreSQL provides three [character data types](#): `CHAR(n)`, `VARCHAR(n)`, and `TEXT`

- `CHAR(n)` is the fixed-length character with space padded. If you insert a string that is shorter than the length of the column, PostgreSQL pads spaces. If you insert a string that is longer than the length of the column, PostgreSQL will issue an error.
- `VARCHAR(n)` is the variable-length character string. With `VARCHAR(n)`, you can store up to `n` characters. PostgreSQL does not pad spaces when the stored string is shorter than the length of the column.
- `TEXT` is the variable-length character string. Theoretically, text data is a character string with unlimited length.

Introduction to the PostgreSQL character types

PostgreSQL provides three primary character types: `character(n)` or `char(n)`, `character varying(n)` or `varchar(n)`, and `text`, where `n` is a positive integer.

Character Types	Description
character varying(n), varchar(n)	variable-length with limit
character(n), char(n)	fixed-length, blank padded
text, varchar	variable unlimited length

Both `char(n)` and `varchar(n)` can store up to n characters in length. If you try to store a longer string in the column that is either `char(n)` or `varchar(n)`, PostgreSQL will issue an error.

However, one exception is that if the excess characters are all spaces, PostgreSQL will truncate the spaces to the maximum length and store the string.

If a string casts to a `char(n)` or `varchar(n)` explicitly, PostgreSQL will truncate the string to n characters before inserting into the table.

The text data type can store a string with unlimited length.

If you do not specify the n integer for the `varchar` data type, it behaves like the `text` data type. The performance of the `varchar` (without n) and `text` are the same.

The only advantage of specifying the length specifier for the `varchar` data type is that PostgreSQL will check and issue an error if you try to insert a longer string into the `varchar(n)` column.

Unlike `varchar`, The `character` or `char` without the length specifier is the same as the `character(1)` or `char(1)`.

Different from other database systems, in PostgreSQL, there is no performance difference among three character types. In most situation, you should use `text` or `varchar`, and `varchar(n)` if you want PostgreSQL to check for the length limit.

PostgreSQL character type example

Let's take a look at an example to see how the char, varchar, and text data types work.

First, we [create a new table](#) for the demonstration.

```
CREATE TABLE character_tests (
  id serial PRIMARY KEY,
  x CHAR (1),
  y VARCHAR (10),
  z TEXT
);
```

Then, we [insert a new row](#) into the `character_tests` table.

```
INSERT INTO character_tests (x, y, z)
VALUES
(
  'Yes',
  'This is a test for varchar',
  'This is a very long text for the PostgreSQL text column'
);
```

PostgreSQL issued an error:

```
ERROR:  value too long for type character(1)
```

This is because the data type of the `x` column is `char(1)` and we tried to insert a string with three characters into this column. Let's fix it.

```
INSERT INTO character_tests (x, y, z)
VALUES
(
    'Y',
    'This is a test for varchar',
    'This is a very long text for the PostgreSQL text column'
);
```

Now we got a different error.

```
ERROR:  value too long for type character varying(10)
```

This is because we tried to insert a string with more than 10 characters into the `y` column with the `varchar(10)` data type.

The following statement inserts a new row into the `character_tests` table successfully.

```
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	x	y	z
1	Y	varchar(n)	This is a very long text for the PostgreSQL text column

(1 row)

Now you should know how to choose the right character data type for your database table. Most of the time, you should choose `text` or `varchar` without length specifier

Number

Integers

There are three kinds of integers in PostgreSQL:

- Small integer (`SMALLINT`) is 2-byte signed integer that has a range from -32,768 to 32,767.
- Integer (`INT`) is a 4-byte integer that has a range from -2,147,483,648 to 2,147,483,647.
- [Serial](#) is the same as integer except that PostgreSQL will automatically generate and populate values into the `SERIAL` column. This is similar to `AUTO_INCREMENT` column in MySQL or `AUTOINCREMENT` column in SQLite.

To store the whole numbers in PostgreSQL, you use one of the following integer types:

`SMALLINT`, `INTEGER`, and `BIGINT`.

The following table illustrates the specification of each integer type:

Name	Storage Size	Min	Max
<code>SMALLINT</code>	2 bytes	-32,768	+32,767
<code>INTEGER</code>	4 bytes	-2,147,483,648	+2,147,483,647
<code>BIGINT</code>	8 bytes	-9,223,372,036,854,775,808	+9,223,372,036,854,775,807

If you try to store a value outside of the permitted range, PostgreSQL will issue an error.

Unlike [MySQL integer](#), PostgreSQL does not provide unsigned integer types.

SMALLINT

The `SMALLINT` requires 2 bytes storage size which can store any integer numbers that is in the range of (-32,767, 32,767).

You can use the `SMALLINT` type for storing something like ages of people, the number of pages of a book, etc.

The following statement [creates a table](#) named `books`:

```
CREATE TABLE books (
    book_id SERIAL PRIMARY KEY,
    title VARCHAR (255) NOT NULL,
    pages SMALLINT NOT NULL CHECK (pages > 0)
);
```

In this example, the `pages` column is a `SMALLINT` column. Because the number of pages of a book must be positive, we added a [CHECK](#) constraint to enforce this rule.

INTEGER

The `INTEGER` is the most common choice between integer types because it offers the best balance between storage size, range, and performance.

The `INTEGER` type requires 4 bytes storage size that can store numbers in the range of (-2,147,483,648, 2,147,483,647).

You can use the `INTEGER` type for a column that stores quite big whole numbers like the population of a city or even country as the following example:

```
CREATE TABLE cities (
    city_id serial PRIMARY KEY,
    city_name VARCHAR (255) NOT NULL,
    population INT NOT NULL CHECK (population >= 0)
);
```

Notice that `INT` is the synonym of `INTEGER`.

BIGINT

In case you want to store the whole numbers that are out of the range of the `INTEGER` type, you can use the `BIGINT` type.

The `BIGINT` type requires 8 bytes storage size that can store any number in the range of (-9,223,372,036,854,775,808,+9,223,372,036,854,775,807).

Using `BIGINT` type is not only consuming a lot of storage but also decreasing the performance of the database, therefore, you should have a good reason to use it.

Floating

There three main types of floating-point numbers:

- `float(n)` is a floating-point number whose precision, at least, n, up to a maximum of 8 bytes.
- `real` or `float8` is a 4-byte floating-point number.
- `numeric` or `numeric(p,s)` is a real number with p digits with s number after the decimal point. The `numeric(p,s)` is the exact numbe

Temporal Data Type

The temporal data types allow you to store date and /or time data. PostgreSQL has five main temporal data types:

- `DATE` stores the dates only.
- `TIME` stores the time of day values.
- `TIMESTAMP` stores both date and time values.
- `TIMESTAMPTZ` is a timezone-aware timestamp data type. It is the abbreviation for [timestamp](#) with the time zone.
- `INTERVAL` stores periods of time.

The `TIMESTAMPTZ` is the PostgreSQL's extension to the SQL standard's temporal data types.

Special Data Type

box - a rectangular box

line - s set of points

point - a geometric pair of numbers

lseg - a line segment

polygon - a closed geometric

inet - an IP4 address

macaddr - a MAC address

Array Data Type

you can store an array of strings an array of integers, etc

JSON Data Type

JSON stands for JavaScript Object Notation. JSON is an open standard format that consists of key-value pairs. The main usage of JSON is to transport data between a server and web application. Unlike other formats, JSON is human-readable text.

PostgreSQL supports native JSON data type since version 9.2. It provides many functions and operators for manipulating JSON data.

Let's get started by [creating a new table](#) for practicing with JSON data type.

```
CREATE TABLE orders (  
  ID serial NOT NULL PRIMARY KEY,  
  info json NOT NULL  
);
```

The `orders` table consists of two columns:

1. The `id` column is the primary key column that identifies the order.
2. The `info` column stores the data in the form of JSON.

Insert JSON data

To insert data into a JSON column, you have to ensure that data is in a valid JSON format. The following `INSERT` statement inserts a new row into the `orders` table.

```
INSERT INTO orders (info)  
VALUES  
(  
  '{ "customer": "John Doe", "items": {"product": "Beer","qty": 6}}');
```

It means `John Doe` bought `6` bottle of `beers`.

Let's insert multiple rows at the same time.

```
INSERT INTO orders (info)  
VALUES  
(  
  '{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}'  
),  
(  
  '{ "customer": "Josh William", "items": {"product": "Toy Car","qty":  
1}}'  
),  
(  
  '{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty":  
2}}'  
);
```

Querying JSON data

To query JSON data, you use the `SELECT` statement, which is similar to querying other native data types:

```
SELECT  
  info  
FROM  
  orders;
```

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

PostgreSQL returns a result set in the form of JSON.

PostgreSQL provides two native operators `->` and `->>` to help you query JSON data.

- The operator `->` returns JSON object field by key.
- The operator `->>` returns JSON object field by text.

The following query uses the operator `->` to get all customers in form of JSON:

```
SELECT
  info -> 'customer' AS customer
FROM
  orders;
```

customer
▶ "John Doe"
"Lily Bush"
"Josh William"
"Mary Clark"

And the following query uses operator `->>` to get all customers in form of text:

```
SELECT
  info ->> 'customer' AS customer
FROM
  orders;
```

customer
▶ John Doe
Lily Bush
Josh William
Mary Clark

Because `->` operator returns a JSON object, you can chain it with the operator `->>` to retrieve a specific node. For example, the following statement returns all products sold:

```
SELECT
  info -> 'items' ->> 'product' as product
FROM
  orders
ORDER BY
  product;
```

product
▶ Beer
Diaper
Toy Car
Toy Train

First `info -> 'items'` returns items as JSON objects. And then `info->'items'-->'product'` returns all products as text.

Use JSON operator in WHERE clause

We can use the JSON operators in `WHERE` clause to filter the returning rows. For example, to find out who bought `Diaper`, we use the following query:

```
SELECT
  info ->> 'customer' AS customer
FROM
  orders
WHERE
  info -> 'items' ->> 'product' = 'Diaper'
```

customer
► Lily Bush

To find out who bought two products at a time, we use the following query:

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

customer	product
► Mary Clark	Toy Train

Notice that we used the [type cast](#) to convert the `qty` field into `INTEGER` type and compare it with two.

Apply aggregate functions to JSON data

We can apply [aggregate functions](#) such as [MIN](#), [MAX](#), [AVERAGE](#), [SUM](#), etc., to JSON data. For example, the following statement returns minimum quantity, maximum quantity, average quantity and the total quantity of products sold.

```
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

```

	min	max	sum	avg
▶	1	24	33	8.25

PostgreSQL JSON functions

PostgreSQL provides us with some functions to help you process JSON data.

json_each function

The `json_each()` function allows us to expand the outermost JSON object into a set of key-value pairs. See the following statement:

```

SELECT
    json_each (info)
FROM
orders;

```

json_each
▶ (customer, ""John Doe"")
(items, {"product": "Beer", "qty": 6})
(customer, ""Lily Bush"")
(items, {"product": "Diaper", "qty": 24})
(customer, ""Josh William"")
(items, {"product": "Toy Car", "qty": 1})
(customer, ""Mary Clark"")
(items, {"product": "Toy Train", "qty": 2})

If you want to get a set of key-value pairs as text, you use the `json_each_text()` function instead.

json_object_keys function

To get a set of keys in the outermost JSON object, you use the `json_object_keys()` function. The following query returns all keys of the nested `items` object in the `info` column

```

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

```

json_object_keys
▶ product
qty
product
qty
product
qty
product
qty

json_typeof function

The `json_typeof()` function returns type of the outermost JSON value as a string. It can be `number`, `boolean`, `null`, `object`, `array`, and `string`.

The following query return the data type of the items:

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

json_typeof
▶ object
object
object
object

The following query returns the data type of the qty field of the nested items JSON object.

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

json_typeof
▶ number
number
number
number

There are more [PostgreSQL JSON functions](#) if you want to dig deeper.

Chapter 24 PostgreSQL UNIQUE constraints

- Ensure values in a column or group of columns are unique in a table

```
CREATE TABLE person(  
    id serial PRIMARY KEY,  
    first_name VARCHAR(50),  
    last_name VARCHAR(50),  
    email VARCHAR(50) UNIQUE  
);  
  
INSERT INTO public.person(  
    first_name, last_name, email)  
VALUES ('haifneg', 'huang', 'qhuanghfeng@a1.com');
```

Chapter 25 PostgreSQL WHERE

Filtering Rows Returned From SELECT Query

Syntax

```
SELECT column1, column2  
FROM table_NAME  
WHERE conditions;
```

Chapter 26 PostgreSQL Retrieve all Data from Table

```
SELECT *  
FROM tableName;
```

Chapter 27 PostgreSQL Query Data from specific column

```
SELECT column1, column2  
FROM tableName;
```

Chapter 28 PostgreSQL Remove Duplicate Records

```
SELECT DISTINCT column1, column2  
FROM tableName  
ORDER BY  
column1 ASC  
column2 DESC;
```

Chapter 29 PostgreSQL Sorting Data returned by SELECT Statement

```
SELECT DISTINCT column1, column2
FROM tableName
ORDER BY
column1 ASC
column2 DESC;
```

Chapter 30 PostgreSQL Order By

- ORDER BY clause sorts the rows returned from SELECT statement
- Sort the rows in ascending or descending order Default is ASC

```
SELECT DISTINCT column1, column2
FROM tableName
ORDER BY
column1 ASC
column2 DESC;
```

Chapter 31 PostgreSQL GROUP BY clause

To divide rows returned from a SELECT Statement into groups

You can apply aggregate function e.g. SUM and COUNT

```
SELECT column_1 aggregate_function (column_2)
FROM tableName;
GROUP BY column_1;
```

Aggregate Functions

- Used to perform calculations on data
- They return a single value calculated from values in a column

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

Chapter 32 PostgreSQL HAVING clause

- Used to in conjunction with GROUP BY clause
- Used filter group rows that do not satisfy a specified condition

Syntax

```
SELECT column_1 aggregate_function (column_2)
FROM tableName;
GROUP BY column_1
HAVING condition;
```

```
SELECT customer_id,SUM(amount)
FROM payment
GROUP BY CUSTOMER_ID
HAVING SUM(amount) > 100
ORDER BY SUM(amount) DESC;
```

Chapter 33 PostgreSQL TRUNCATE and DROP

TRUNCATE delete all data

DROP delete the table from the database

```
TRUNCATE TABLE table_name;
DROP TABLE table_name;
```

Chapter 34 PostgreSQL CRUD

CREATE READ UPDATE DELETE INSERT

Chapter 35 PostgreSQL CREATE DATABASE on PgAdmin

With the GUI tool, it is so easy to create a database;

Chapter 36 PostgreSQL CREATE a Table on PgAdmin

choose the database you want to create the table

choose Schemas-> Tables-> Right click -> Create -> Table...

then you can input the table name you want and the table fields

Chapter 37 PostgreSQL CREATE a Table with commands

Syntax

```
CREATE TABLE table_name(
    column1_name datatype column_constraint,
    column2_name datatype column_constraint,
);

CREATE TABLE Berries(
    berry_id INT PRIMARY KEY,
    berry_name VARCHAR(255) NOT NULL
);
```

Chapter 39 PostgreSQL Subquery

- Allows you to construct complex queries
- Query nested inside another query such as SELECT
- To construct a subquery the second query is placed in brackets
- The Where clause is used as expression in the subquery

Syntax

```
SELECT "column_name1"
FROM "table_name1"
WHERE "column_name2" [Comparison Operator]
( SELECT "column_name3"
FROM "table_name2"
WHERE "condition");
```

```
SELECT film_id, title, rental_rate
FROM film
WHERE rental_rate > (
SELECT AVG(rental_rate)
FROM film);
```

Chapter 40 PostgreSQL Update

Syntax

```
UPDATE table_name
SET column1 = value1,
    column2 = value2
WHERE condition;
```

```
UPDATE Berries
SET berry_id = 2
WHERE berry_name = 'Hello';
```

Chapter 41 PostgreSQL Delete

Syntax

```
DELETE FROM table_name
WHERE condition;
```

```
DELETE FROM Berries
WHERE berry_id = 2;
```

Chapter 42 PostgreSQL INSERT

Syntax

```
INSERT INTO table_name(column1, column2)
VALUES(value1, value2);
```

```
INSERT INTO Berries
values(9,'Hello'),
(5,'Hello'),
(6,'Hello');
```

Chapter 43 PostgreSQL Comparison Operator

Comparison Operator	Description
=	Equal
<> AND !=	Not Equal
> >= < <=	Greater than, Greater than or equal
IN ()	Matches value in a list
IS NULL	NULL Value
BETWEEN	Within a range
LIKE	Pattern matching with wild cards % and _

```
SELECT *
FROM payment
WHERE amount > 9.99;
```

Chapter 44 PostgreSQL Between Operator

Match a value against a range of values

Values can be numbers, text or dates

Included in the WHERE clause

Syntax

```
SELECT column_names
FROM table_name
WHERE column_name BETWEEN value1,value2;
```

```
SELECT customer_id, payment_id, amount
FROM payment
WHERE amount between 5 and 10;
```

Chapter 45 PostgreSQL Not Between Operator

- Return values not in a range of values
- Values can be numbers, text or dates
- Include in the WHERE clause

Syntax

```
SELECT column_names
FROM table_name
WHERE column_name NOT BETWEEN value1 value2;
```

Chapter 46 PostgreSQL Not Operator

- Return records if condition specified is not True
- Included in the WHERE clause

Syntax

```
SELECT column_names
FROM table_name
WHERE NOT condition;
```

```
SELECT *
FROM inventory
WHERE NOT store_id = 1;
```

Chapter 47 PostgreSQL Like Operator

- used to match pattern in a column
- Used to query data by using a matching technique
- % wildcard allows you to match string of any length characters including zero
- _ wild card allows you to match a single character

Syntax

```
SELECT column_names
FROM table_name
WHERE column LIKE pattern;
```

```
SELECT first_name, last_name
FROM customer
WHERE first_name LIKE 'A__';
```

Chapter 48 PostgreSQL Or Operator

```
SELECT column_names
FROM table_name
WHERE condition OR condition;
```

Chapter 49 PostgreSQL And Operator

```
SELECT column_names
FROM table_name
WHERE condition AND condition;
```

Chapter 50 PostgreSQL combining And Or Operator

Adjust priority with parentheses

Chapter 51 PostgreSQL Limit Operator

- Gets a subset of row returned by a query
- Gets the number of highest of lowest items in a table
- OFFSET skip the first m rows

```
SELECT *  
FROM table_NAME  
LIMIT n;
```

```
SELECT *  
FROM table_NAME  
LIMIT n OFFSET m;
```

```
SELECT *  
FROM table_Name  
ORDER BY column1  
LIMIT n;
```

Chapter 52 PostgreSQL IN Operator

- Specify or check against multiple values in a WHERE clause

```
SELECT column_names  
FROM table_Name  
WHERE column_name IN (value1, value2);  
WHERE column_name IN (SELECT);
```

Chapter 53 PostgreSQL UNION Operator

- Used to combine result sets of multiple queries into a single result
- Removes all duplicate rows
- Both queries must return same numbers of columns
- The corresponding columns in the queries must have compatible data types

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

Chapter 54 PostgreSQL UNION ALL Operator

- Used to combine result sets of multiple queries into a single result
- Does not remove duplicate rows
- Both queries must return sme number of columns

- The corresponding columns in the queries must have compatible data types

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

Chapter 55 PostgreSQL INTERSECT Operator

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

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

Chapter 56 PostgreSQL EXCEPT Operator

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

```
SELECT column1,column2
FROM table1
WHERE conditionA
EXCEPT
SELECT column1, column2
FROM table2
WHERE conditionB;
```

Chapter 57 PostgreSQL JOINS

- Used to retrieve data from multiple tables
- Joins is performed when two or more tables are joined in a SQL Statement
- Tables participating in the join should have related columns between them

Types of Joins

- INNER JOIN
- LEFT JOIN
- FULL OUTER JOIN
- CROSS JOIN
- NATURAL JOIN

Chapter 58 PostgreSQL INNER JOIN

- Used to retrieve data from multiple tables using INNER JOIN clause

- Returns records that have matching values from columns in both tables

```
SELECT column_names
FROM tableA
INNER JOIN tableB
on tableA.column_name = tableB.column_name;
```

Chapter 59 PostgreSQL Left JOIN

- Used to retrieve data from multiple table using LEFT JOIN clause
- Returns records from the left table and the matched records from right tables

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

Chapter 60 PostgreSQL Full Outer Join

- Used to retrieve data from multiple tables using FULL Outer JOIN clause
- Returns records from the left table and the right tables
- If rows in joined table do not match the full outer joins sets NULL values
- For the matching rows a single row is included in the result from both table
- Note FULL OUTER JOIN can potentially return very large results-sets

```
SELECT column_names(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name;
```

Chapter 61 PostgreSQL Cross Join

- Used to retrieve data from multiple tables using CROSS JOIN clause
- Returns records from joined tables
- Also called CARTESIAN JOIN
- Produce the cartesian product of rows from combined joined tables
- Does not require the tables to have matching columns in the JOIN Clause

```
SELECT *
FROM table1
CROSS JOIN table2;
```

Chapter 62 PostgreSQL Natural Join

- Used to retrieve data from multiple tables using clause NATURAL JOIN
- Creates an implicit join based on matching column names in the joined tables

- Natural join can be a INNER JOIN, LEFT JOIN, RIGHT JOIN
- PostgreSQL will use the INNER JOIN by default

```
SELECT *  
FROM table1  
NATURAL JOIN table2;
```