CENG 352

Database Management Systems
Spring 2024-2025
Project 1

April 13, 2025, Sunday, 23:59

1. Objectives

In this project, you are asked to write several SQL queries on a relational database using the given dataset. Your SQL server will be PostgreSQL, and you will work on your local environment. This project consists of 4 parts:

- Creating the database using the given CSV files.
- Writing proper SQL queries for certain problems.
- Testing the effect of indexing on query execution.
- Creating triggers and views.

2. Database Schema

2.1 Tables

Here are the database tables and their columns:

Country
    name
    code
    capital
    area
    population

City
- name
- country_code
- population
- elevation

Economy
- country_code
- gdp
- agriculture *percentage of services of the GDP*
- industry *percentage of services of the GDP*
- service *percentage of services of the GDP*
- inflation
- unemployment

Religion
- country_code
- name
- percentage

Spoken
- country_code
- language
- percentage

Encompasses
- country_code
- continent_name
- percentage

Continent
- name
- area

3. Tasks

3.1 Task 1 - Creating the Database (15 pts)

Using the given CSV files and considering the database schema above, you should create a file named `task1.sql` that contains SQL statements that you've used to create the tables. You need to use PostgreSQL.

3.2 Task 2 - SQL Queries (50 pts)

For this task, you should be able to write SQL queries for given problems. See the given query results to understand the structure. Create an individual SQL file for each subtask. Name the SQL files in this format: `task2_1.sql`, `task2_2.sql`, etc. Each task is worth 10 points.

3.2.1 Regulations

- For 3rd question, a country is inside a continent if encompasses table percentage of the country-continent tuple is more than all other country-continent tuples for that country.
- For 4th question, a country is inside a continent if encompasses table percentage of the country-continent tuple is more than 50%.
- For 5th question, a country can be in multiple continents so if a country of 1000 people is 40% in Europe, assume 400 people of that country contributes to the total population of Europe

3.2.2 Questions

1. Identify cities with population exceeding 50% of their country's average city population using PARTITION BY.
2. Divide countries into 4 equal groups based on their total area. Return quartile number, number of countries in each quartile, and the average area for that quartile. (*Hint:* ***NTILE*** *function*)
3. Find top 3 continents by GDP per capita. Return continent name, total GDP, avg GDP per capita, ordered by avg GDP per capita descending.
4. Write a single SQL query that computes the statistics present in a cross tabulation over continent and sectors(agriculture/industry/service). The aggregate value reported should be total gdp for each continent-sector pair. Each row should contain continent, sector, total GDP. You must solve this using **CUBE** function.
5. Use GROUPING SETS to show world/continent aggregates(Only use Encompasses and Country tables Don't use the Continent table):
   - Total area,
   - Total population

3.3 Task 3 - Index Optimization (20 pts)

Make the following queries. Run as fast as possible by adding indexes to the tables. Your goal is to find the least estimated execution time according to EXPLAIN statement in sql. To do so, you may create BTREE and HASH indexes. You may create these on any table, and with any set of columns. Write your answers in txt files: `task3_1.txt`, `task3_2.txt`, ...

For full credit, your answer to each problem must have the following four pieces of information:

1. CREATE INDEX statements showing which indexes you propose to use
2. An explanation for why you chose each of the indexes (e.g. the query is looking for a range of values, so we use BTREE index instead of a HASH index)
3. The output of EXPLAIN for the query after the indexes were added
4. Whether the indexes caused a performance increase for the query (just state if there was a speed up or not. You do not need to speculate why the index(es) did not speed up the query.)

To check the performance of each query independently, you should delete any indexes before moving on to the next problem. You can do so using DROP INDEX

Questions

Q1

```
SELECT c.name, ct.name, c.population, e.gdp, e.inflation
FROM city c
NATURAL JOIN country ct
NATURAL JOIN economy e
```

Q2

```
SELECT c.name, ct.name, c.population, e.gdp, e.inflation
FROM city c
NATURAL JOIN country ct
NATURAL JOIN economy e
WHERE c.name LIKE 'B%'
  AND ct.name = 'Turkey'
```

Q3

```
SELECT c.name, ct.name, c.population, e.gdp, e.inflation
FROM city c
NATURAL JOIN country ct
NATURAL JOIN economy e
WHERE e.gdp > 100000
```

Q4

```
SELECT c.name, ct.name, c.population, e.gdp, e.inflation
FROM city c
NATURAL JOIN country ct
NATURAL JOIN economy e
WHERE c.population > 5000000
  AND inflation = 4
ORDER BY population
```

IMPORTANT: Many databases, such as MySQL and PostgreSQL, automatically create indexes on primary keys and unique constraints and foreign keys. If a table has a foreign key constraint, an index might already exist on that column, making manual index creation redundant. You can check these existing indexes via.

```
SELECT tablename, indexname, indexdef
FROM pg_indexes
WHERE schemaname = 'public'
order by tablename;
```

3.4 Task 4 - Views and Triggers (15 pts)

3.4.1 View(5pts)

Create a view for:

- Country Name
- Biggest City(In population)
- Highest Elevation
- Density(Population/Area)

If a country does not have any cities listed in the city table, display **NULL** for both the largest city and the highest elevation.

3.4.2 Aggregated Table(5pts)

Create a Table named **country_city_count** whose columns can be seen below. And populate it. Your sql file submission should contain the create and insert statements.

- Country Name
- Country Code
- Number of Cities(inside the country)

3.4.3 Triggers(5pts)

Implement two triggers to update the table above when:

- A new city is inserted.
- A city's occupation changes (i.e., its country is updated).

4. Regulations

4.1 Submission

Submission will be done via ODTUClass. Please remember that **late submission is allowed up to 5 days for the entire semester**. You can distribute these 5 days to any project you want. You should put the answer query to each question in a separate .sql file and tar those .sql files with the following name: `e1234567_project1.tar.gz`

- `task1.sql`
- `task2_1.sql`
- `task2_2.sql`
- ...
- `task2_5.sql`
- `task3_1.txt`
- `task3_2.txt`
- `task3_3.txt`
- `task3_4.txt`
- `task4_1.sql`
- `task4_2.sql`
- `task4_3.sql`

Where you should replace ”1234567” with your own student number.

4.2 SQL Style

You should write your queries in a readable manner. Write each clause on a separate line, add indentation for a clear look. For example:

```
SELECT *
FROM orders o
WHERE o.order_id = 'test' AND
      o.testing_phase = 1
```

is easier to read than:

```
SELECT * FROM orders o WHERE o.order_id = 'test' AND o.testing_phase = 1
```

You can use online formatters/beautifiers for better indentation if you want.

4.3 Newsgroup

You must follow the ODTUClass for discussions and possible updates on a daily basis.

5. Tutorial & Guides

- You can download PostgreSQL server from **here**. Also an installation document for ubuntu users have been uploaded with this document.
- For visualization, you can use **DBeaver** or **DataGrip**. You can also use them for many other database servers. **DBeaver** guide is given below **DataGrip** guide is given in another document.
- Once you start the database server, open DBeaver and click 'New Database Connection' on the top left corner. (Figure 1)
- Choose PostgreSQL. (Figure 2)
- Enter credentials and connect to the default 'postgres' database. You will create your own database for this project later. (Figure 3)
- Open a new script and execute "create database <database_name>" by selecting the script and hitting (CTRL + ENTER). (Figure 4)
- Now that you have created another database, connect to the newly created database just like before. (Figure 5)
- Open 'Tables' to see the tables. (Figure 6) Since there are no tables, you need to create tables using your script. Run queries by selecting parts of the script and hitting CTRL + ENTER. (Figure 7)
- You are almost there. Now, you need to import the data to tables.
- To import data, right-click on the table name and click 'Import data' (Figure 8)
- Choose CSV type and .csv file that you want to import for the selected table. (Figure 9, 10)
- Choose **comma(,)** as the delimiter for the given CSV files. You can observe that in individual CSV files.
- That's it. Now you can write queries on tables.
- Note: Don't mind that the figures are old, the import behavior is exactly the same.

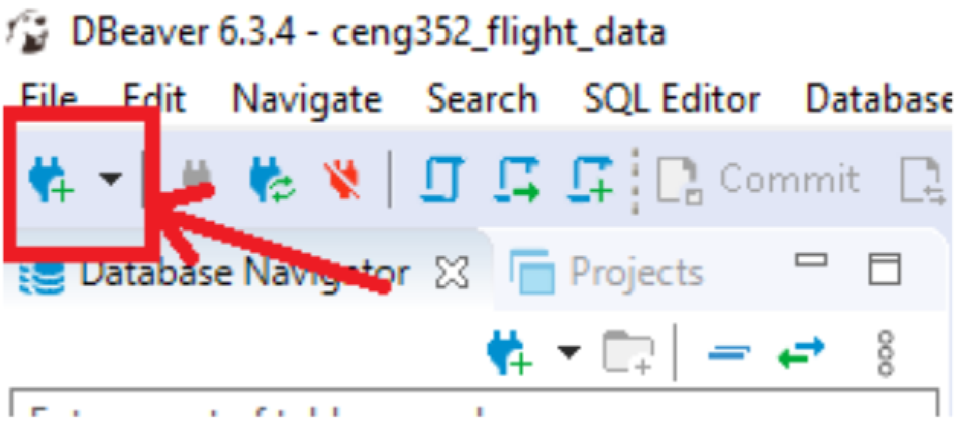


Figure 1: New database connection

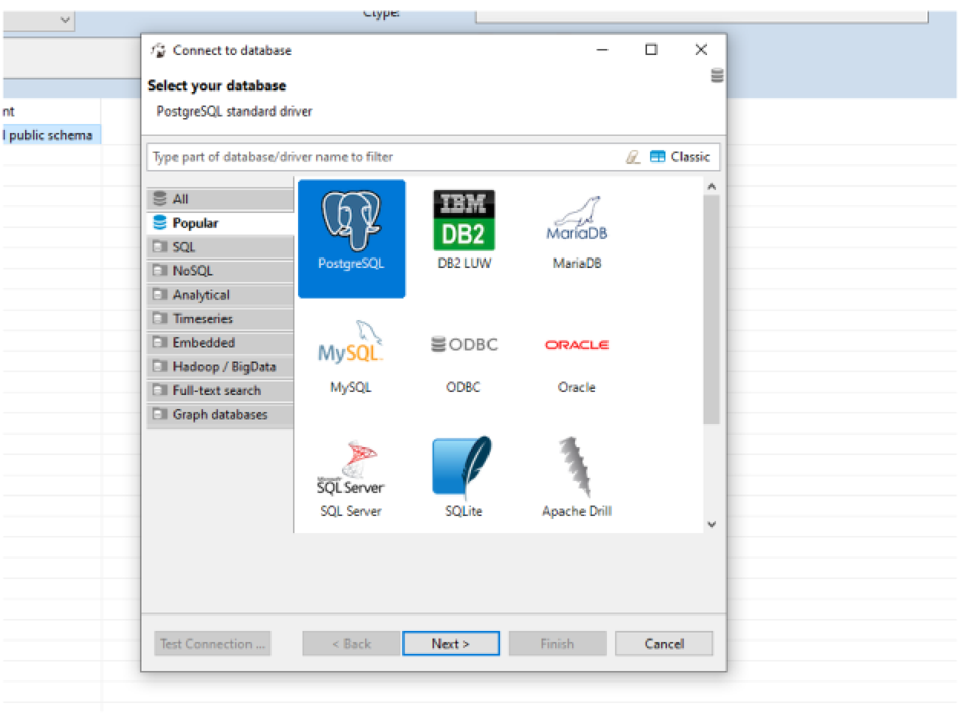
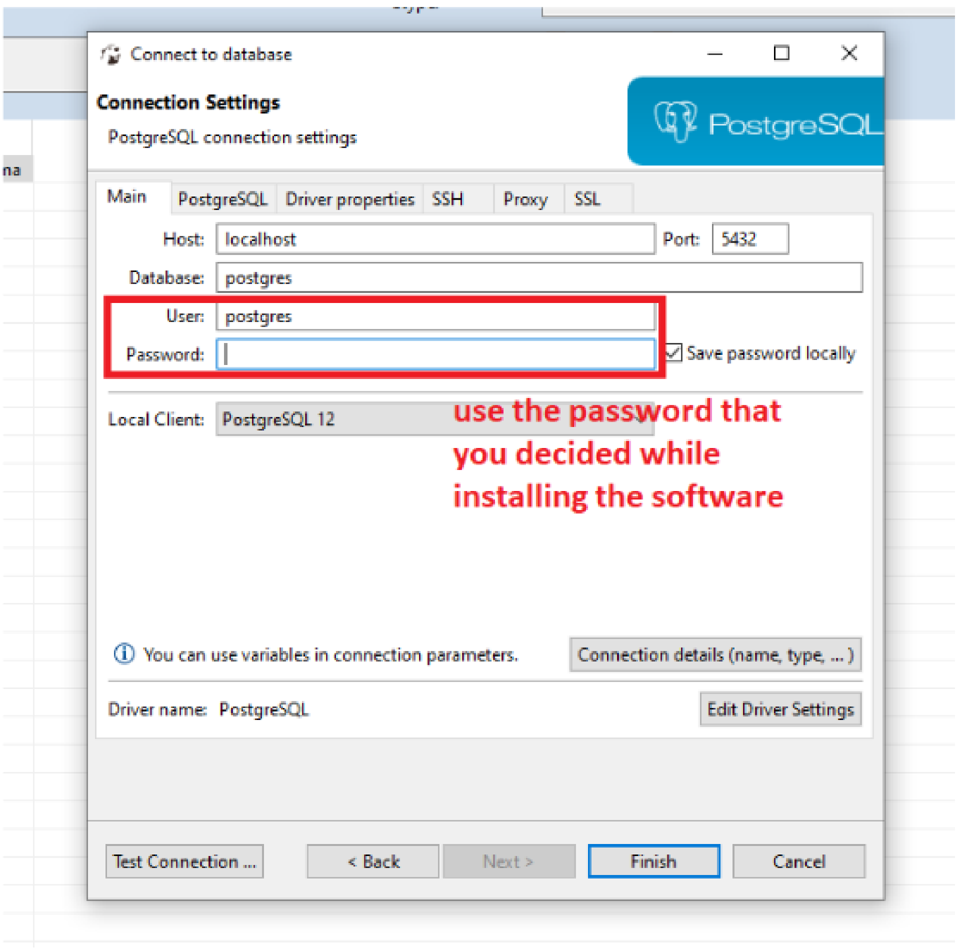


Figure 2: Database options


Connect to database

Connection Settings

PostgreSQL connection settings

Main | PostgreSQL | Driver properties | SSH | Proxy | SSL

Host: localhost Port: 5432

Database: postgres

User: postgres

Password:

☑ Save password locally

Local Client: PostgreSQL 12

use the password that you decided while installing the software

You can use variables in connection parameters.

Driver name: PostgreSQL

Figure 3: Enter credentials and connect

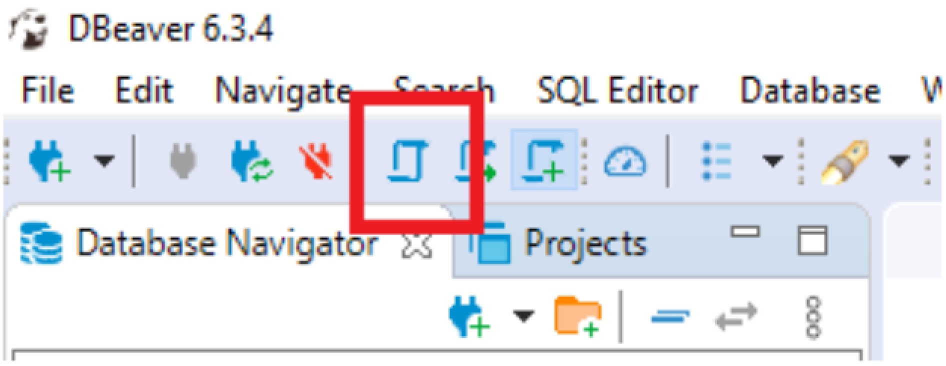
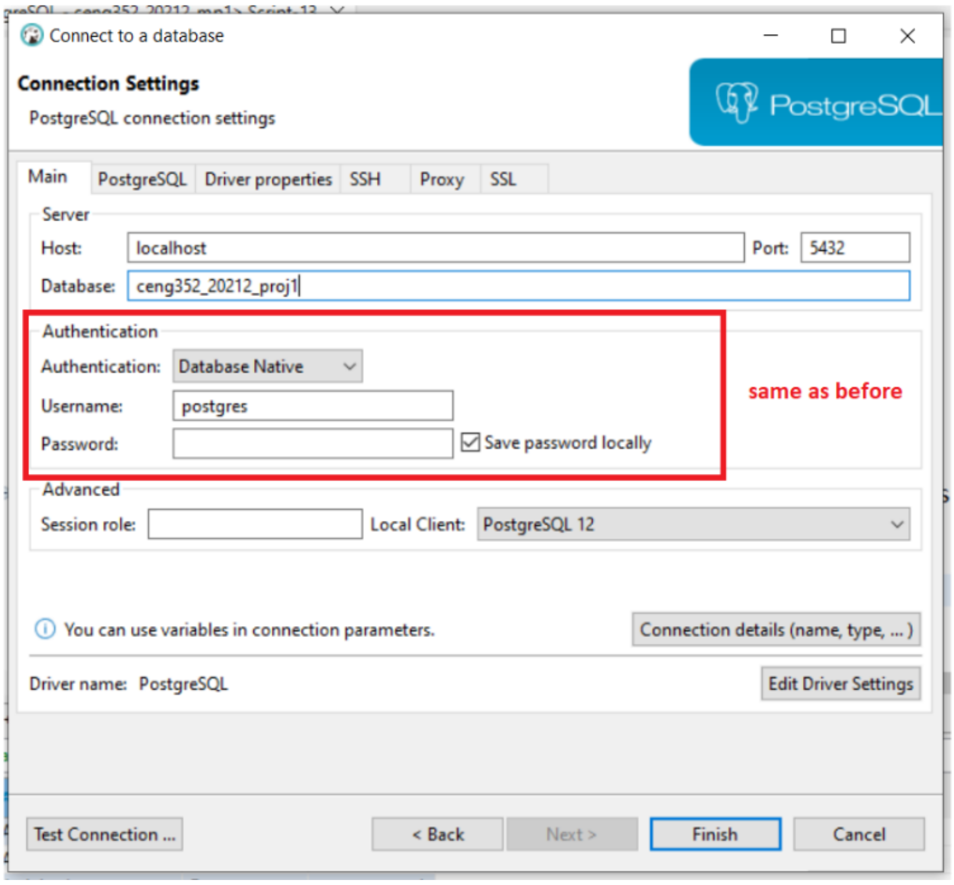


Figure 4: Open new script


Connect to a database

Connection Settings

PostgreSQL connection settings

Main | PostgreSQL | Driver properties | SSH | Proxy | SSL

Server

Host: localhost Port: 5432

Database: ceng352_20212_proj1

Authentication

Authentication: Database Native

Username: postgres

Password:

☑ Save password locally

same as before

Advanced

Session role: Local Client: PostgreSQL 12

You can use variables in connection parameters.

Driver name: PostgreSQL

Figure 5: Connect to new database

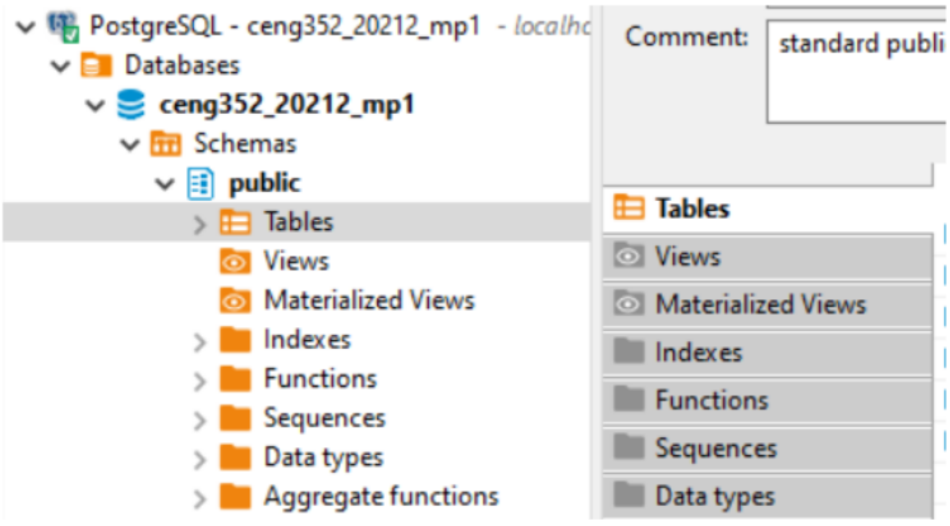
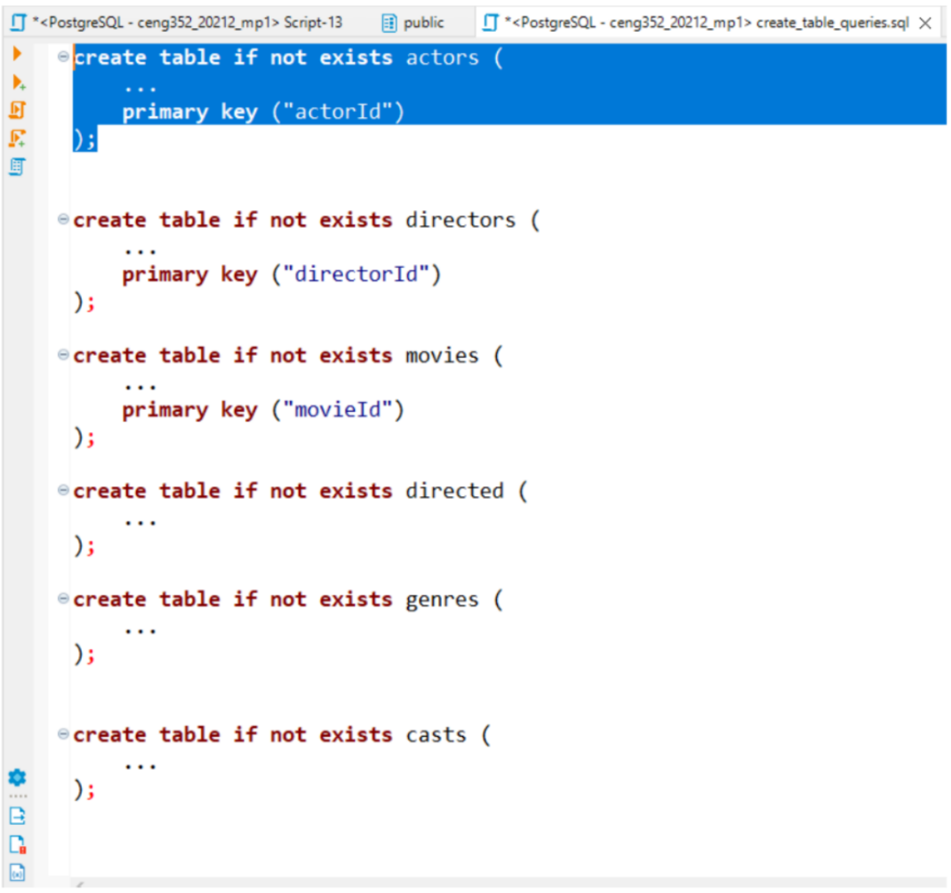
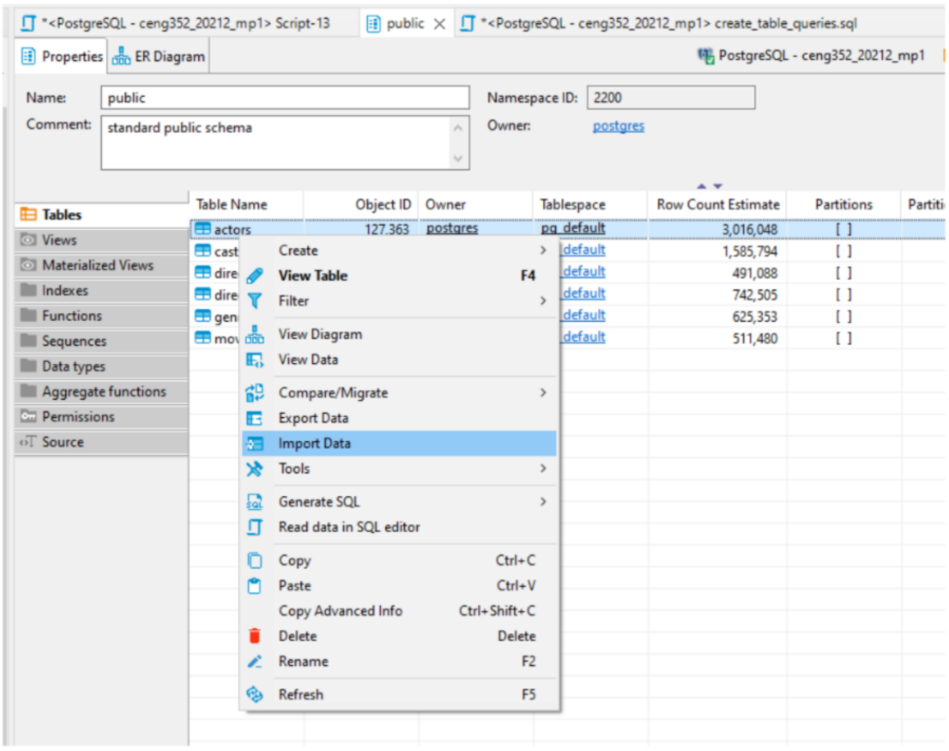


Figure 6: Open tables view


*<PostgreSQL - ceng352_20212_mp1> Script-13
public
*<PostgreSQL - ceng352_20212_mp1> create_table_queries.sql

```sql
create table if not exists actors (
    ...
    primary key ("actorId")
);


create table if not exists directors (
    ...
    primary key ("directorId")
);

create table if not exists movies (
    ...
    primary key ("movieId")
);

create table if not exists directed (
    ...
);

create table if not exists genres (
    ...
);


create table if not exists casts (
    ...
);
```

Figure 7: Run create scripts



Name: public

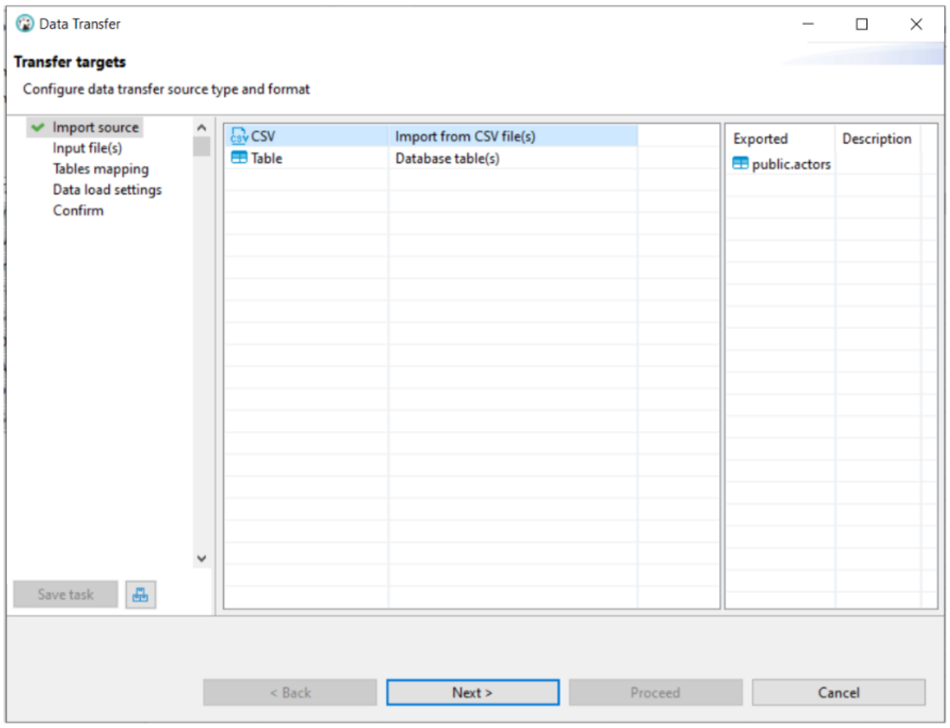
Comment: standard public schema

Namespace ID: 2200

Owner: postgres

Table Name	Object ID	Owner	Tablespace	Row Count Estimate	Partitions	Partiti
actors	127.363	postgres	pg_default	3,016,048	[ ]	
cast			default	1,585,794	[ ]	
dire			default	491,088	[ ]	
dire			default	742,505	[ ]	
gen			default	625,353	[ ]	
mov			default	511,480	[ ]	

Figure 8: Import data



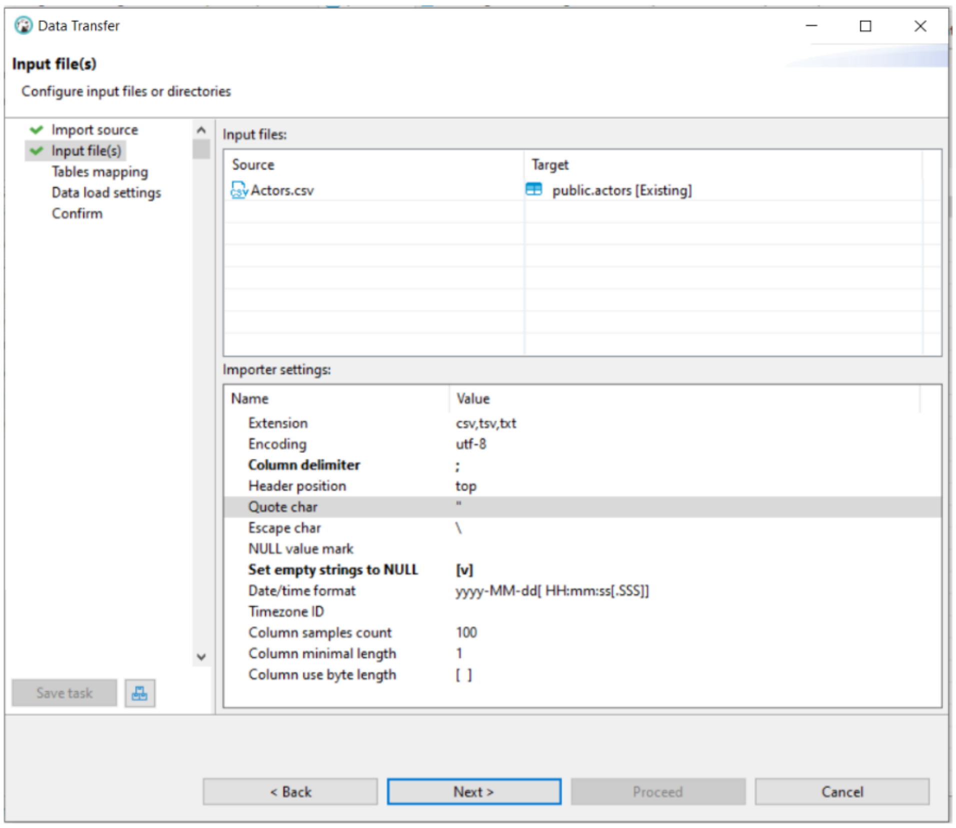
Transfer targets

Configure data transfer source type and format

CSV	Import from CSV file(s)
Table	Database table(s)

Exported	Description
public.actors	

Figure 9: Choose CSV



Input file(s)

Configure input files or directories

Input files:

Source	Target
Actors.csv	public.actors [Existing]

Importer settings:

Name	Value
Extension	csv,tsv,txt
Encoding	utf-8
Column delimiter	**;**
Header position	top
Quote char	"
Escape char	\
NULL value mark	
Set empty strings to NULL	**[v]**
Date/time format	yyyy-MM-dd[HH:mm:ss[.SSS]]
Timezone ID	
Column samples count	100
Column minimal length	1
Column use byte length	[ ]

Figure 10: Set delimiter