Spatial Databases

Lecture notes

for Geoinformatics/Cartography Master Students

Dr. UNGVÁRI Zsuzsanna

assistant professor

ELTE Institute of Cartography and Geoinformatics

Contents

Foreword

These lecture notes were created for the Spatial Databases course in the Geoinformatics and Cartography Master's programme at Eötvös Loránd University, Budapest, Hungary. It also includes an introductory section on database management and queries. Based on the author's experience, the best way to master SQL is through extensive practice with queries, clauses, functions, etc. Therefore, this note is consists of the following parts:

- A description of **basic SQL** with several practical examples.
- A chapter on **PostGIS**, including theoretical concepts, function descriptions, and numerous examples.
- Practical guidance on **installing PostgreSQL and PostGIS**, as well as uploading databases and performing other related tasks.

The author strongly recommends readers to have some prior experience with QGIS or ArcGIS, as it simplifies the understanding of spatial databases.

The tasks in these note are based on **PostgreSQL 13** and **PostGIS 3.1**. Newer versions may have some differences, so please refer to the **official documentation** for updates.

I wish you an enjoyable learning experience!

the author

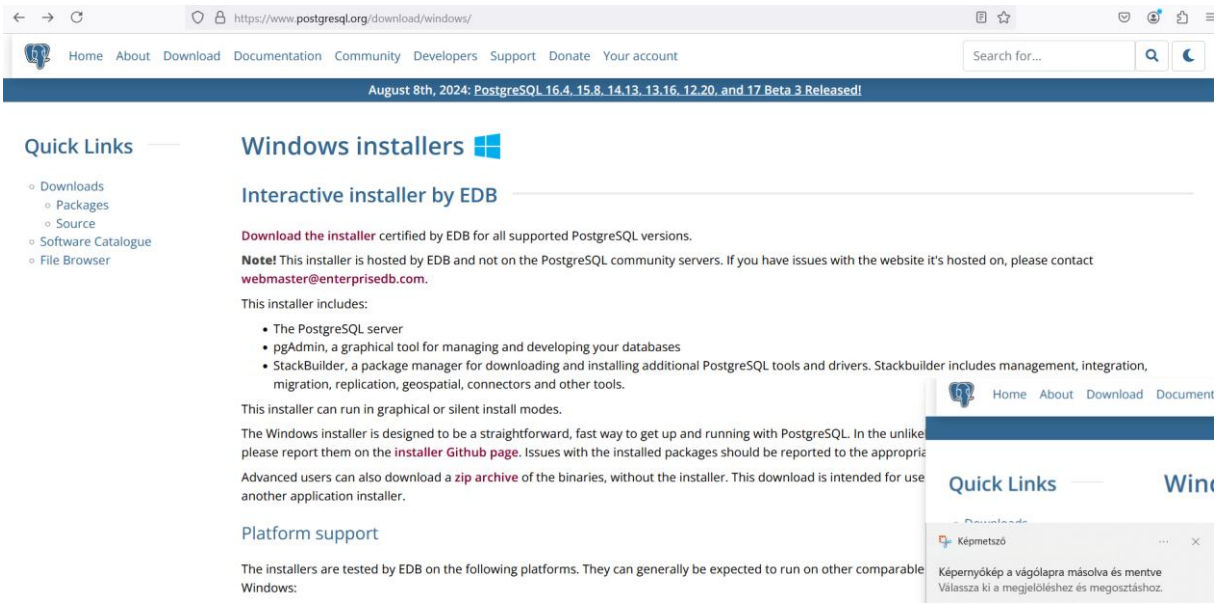
Reviewed by Dr. Gede Mátyás

Chapter 1: Installing pgAdmin4, PostgreSQL and POSTGIS, a short introduction to pgAdmin4

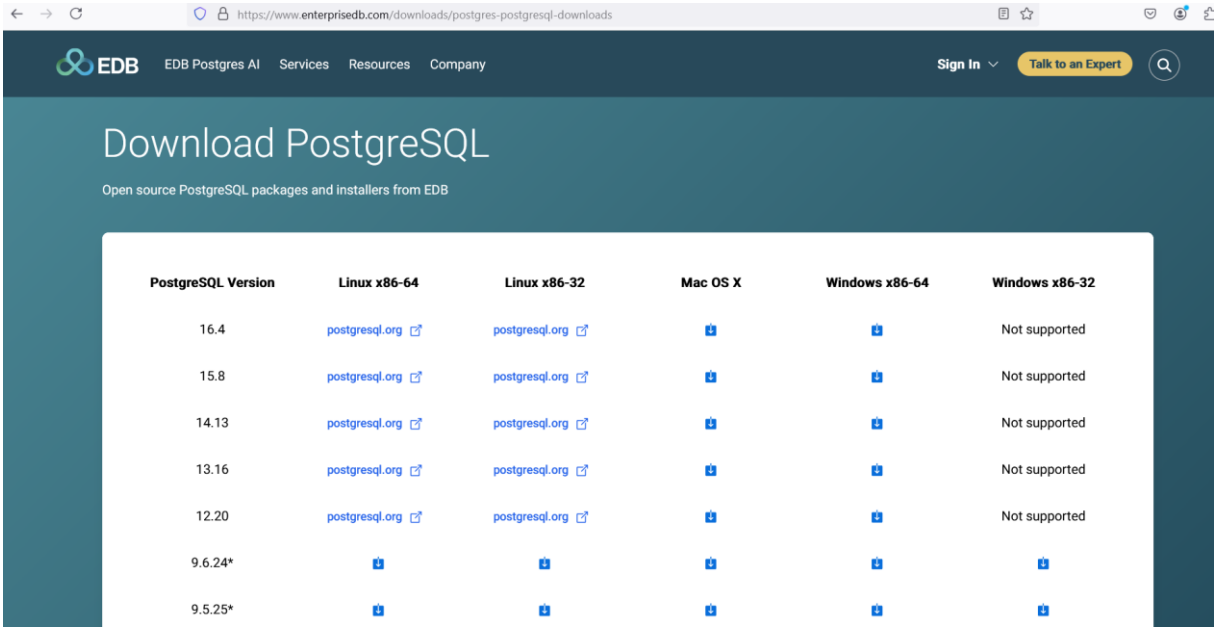
„pgAdmin is the most popular and feature rich open source administration and development platform for PostgreSQL, the most advanced open source database in the world”.[1] However, pgAdmin itself is only a database management tool. To use it with a PostgreSQL database, PostgreSQL must be installed alongside pgAdmin. I recommend using the following website to install these programs. Install the available newest version. *(Note: when I wrote these lecture notes, I used the PostgreSQL version 13 and pgAdmin4. If you will found some differences in syntax or in pictures, that caused by the different software versions. In this case, please check the documentation!)*

https://www.postgresql.org/download/windows/

Click on **Download the installer**, then select your computer's operating system and the software version.



[1] https://www.pgadmin.org/



The installion process is straightforward:

1. Click **Next** through the setup and accept the default options.
2. Set the port number. The default is **5432**. If it is not already in use, keep it.
3. Choose a username and a password (*for example: postgres/appletree*). Do not forget these credentials!

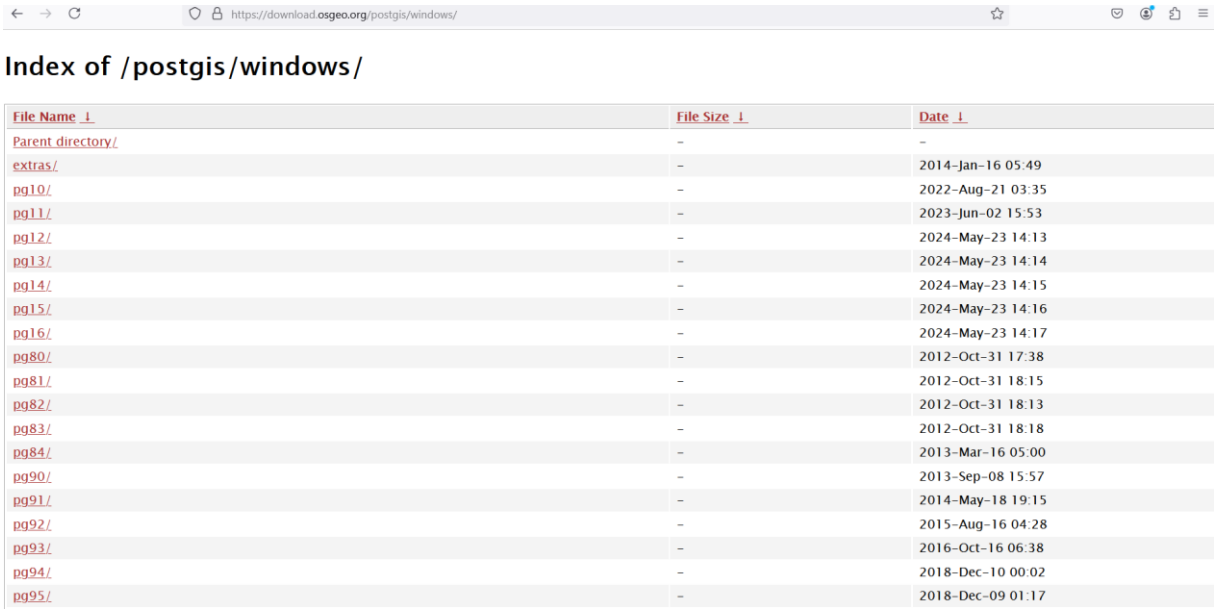
Finally, you will set up the PostgreSQL DB with pgAdmin. While pgAdmin is a professional, robust and stable software, its user interface is not very user-friendly. Therefore, in the next chapter, the author will suggest an alternative solution (DBeaver).

However, if you prefer or are more familiar with pgAdmin, you can still use it. Keep in mind that some tasks in this guide are written specifically for DBeaver.

To handle spatial data, you need to install the PostGIS extension. Please visit the following website to download and install PostGIS on Windows using one of the two methods below:

Method 1: Download and Install the Setup File:

1. Open this website: https://download.osgeo.org/postgis/windows/
2. Find the appropriate version of PostGIS and download the setup file (EXE).

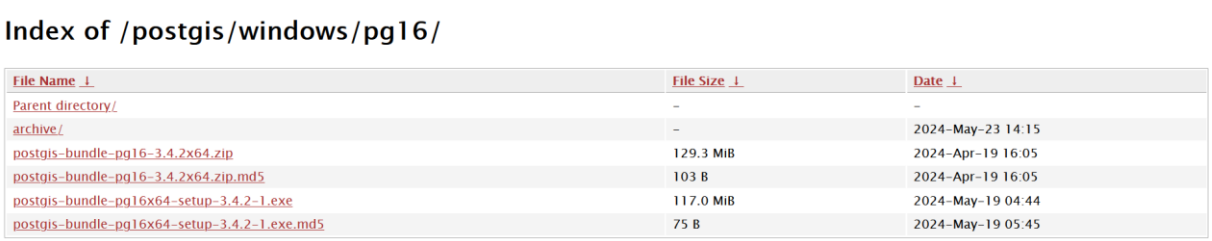


https://download.osgeo.org/postgis/windows/

Index of /postgis/windows/

File Name ↓	File Size ↓	Date ↓
Parent directory/	-	-
extras/	-	2014-Jan-16 05:49
pg10/	-	2022-Aug-21 03:35
pg11/	-	2023-Jun-02 15:53
pg12/	-	2024-May-23 14:13
pg13/	-	2024-May-23 14:14
pg14/	-	2024-May-23 14:15
pg15/	-	2024-May-23 14:16
pg16/	-	2024-May-23 14:17
pg80/	-	2012-Oct-31 17:38
pg81/	-	2012-Oct-31 18:15
pg82/	-	2012-Oct-31 18:13
pg83/	-	2012-Oct-31 18:18
pg84/	-	2013-Mar-16 05:00
pg90/	-	2013-Sep-08 15:57
pg91/	-	2014-May-18 19:15
pg92/	-	2015-Aug-16 04:28
pg93/	-	2016-Oct-16 06:38
pg94/	-	2018-Dec-10 00:02
pg95/	-	2018-Dec-09 01:17

3. The installer and follow the instructions to complete the installation.

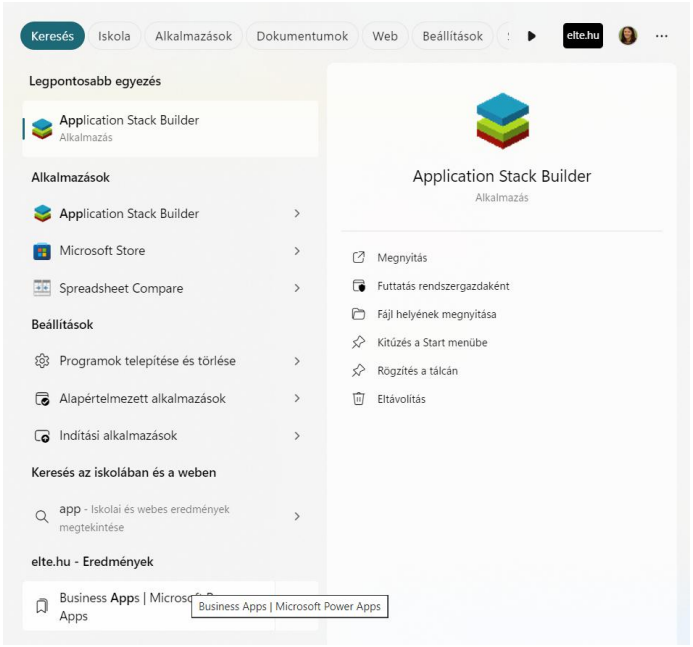


Index of /postgis/windows/pg16/

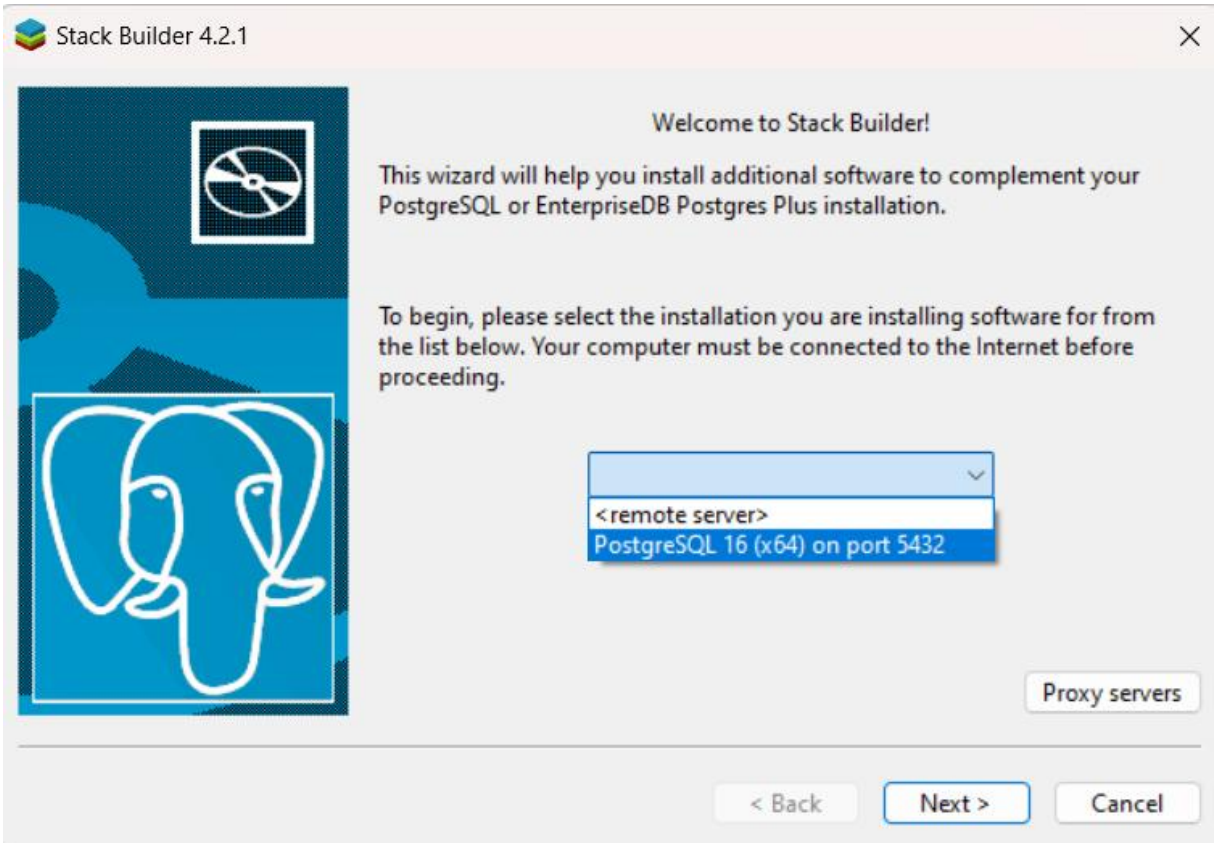
File Name ↓	File Size ↓	Date ↓
Parent directory/	-	-
archive/	-	2024-May-23 14:15
postgis-bundle-pg16-3.4.2x64.zip	129.3 MiB	2024-Apr-19 16:05
postgis-bundle-pg16-3.4.2x64.zip.md5	103 B	2024-Apr-19 16:05
postgis-bundle-pg16x64-setup-3.4.2-1.exe	117.0 MiB	2024-May-19 04:44
postgis-bundle-pg16x64-setup-3.4.2-1.exe.md5	75 B	2024-May-19 05:45

Method 2: Install via Application Stack Builder

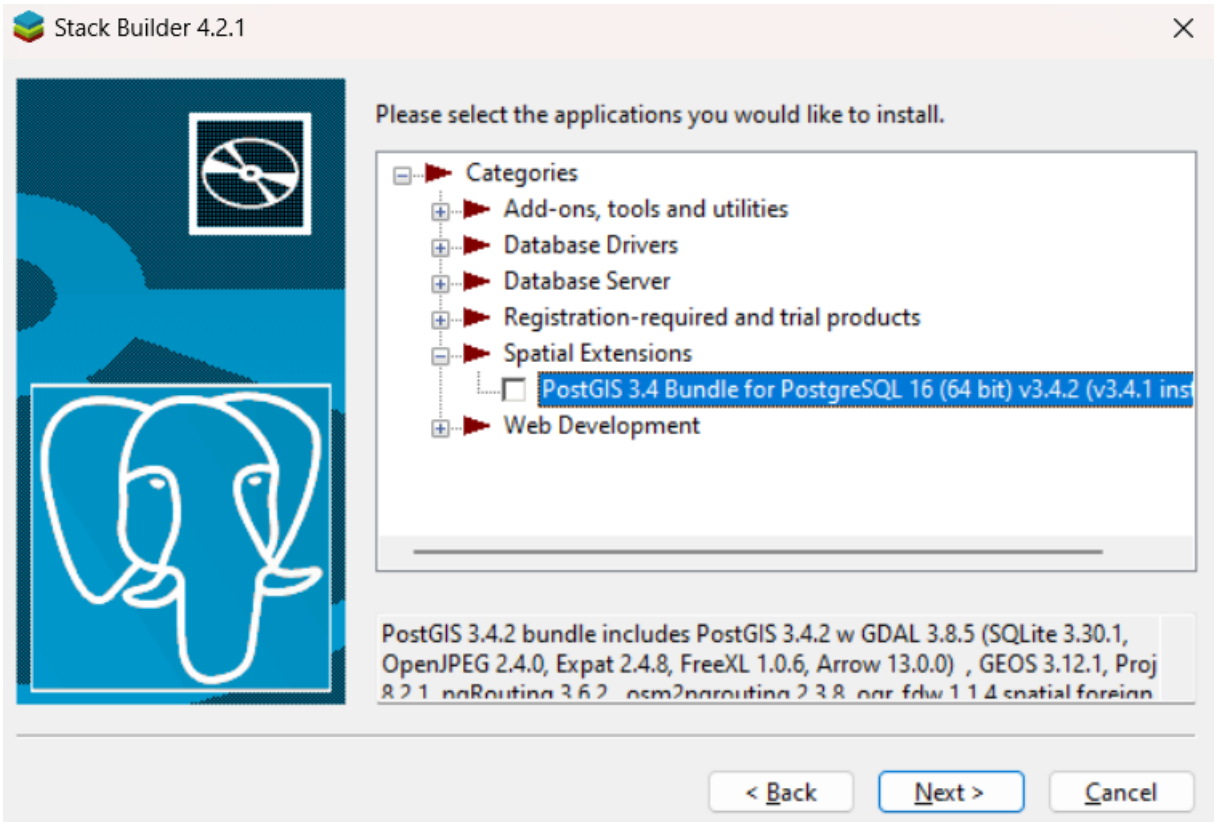
1. PostgreSQL includes the **Application Stack Builder**. Open it.



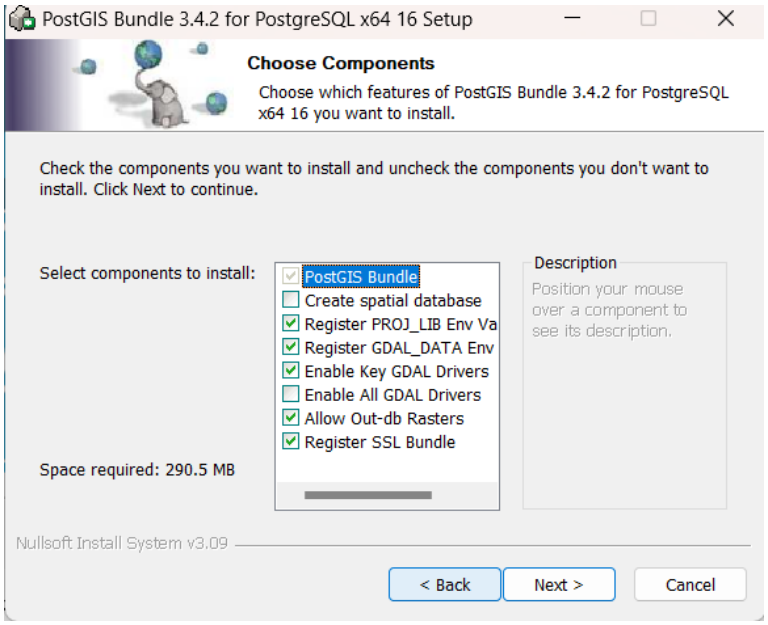
2. Select your PostgreSQL version, and click **Next**.



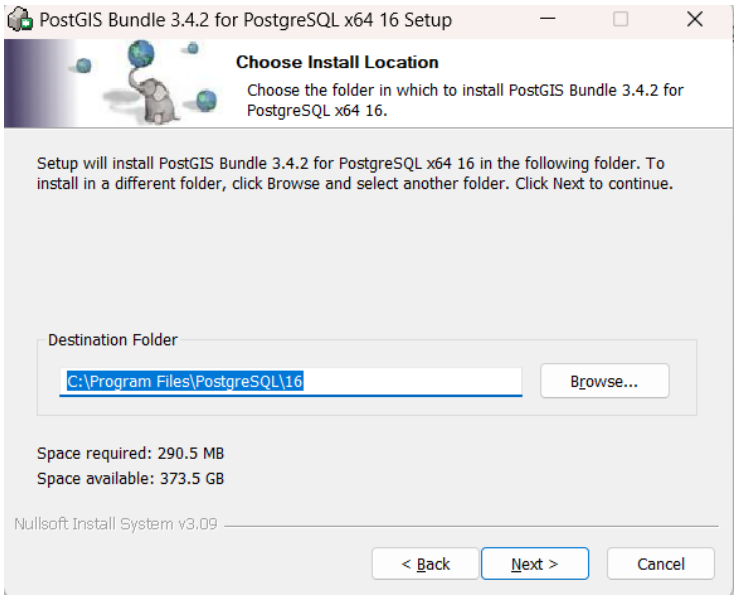
3. Expand the **Spatial Extensions** list and check **PostGIS Bundle**.
4. Choose the folder where you want to download the PostGIS Bundle.



5. Accept the license agreement, then review and confirm the components to be installed.



6. Set the destination folder to your PostgreSQL directory and confirm.
7. The installation process will begin. Please wait until it is completed.



Now, you are ready to start working with pgAdmin.

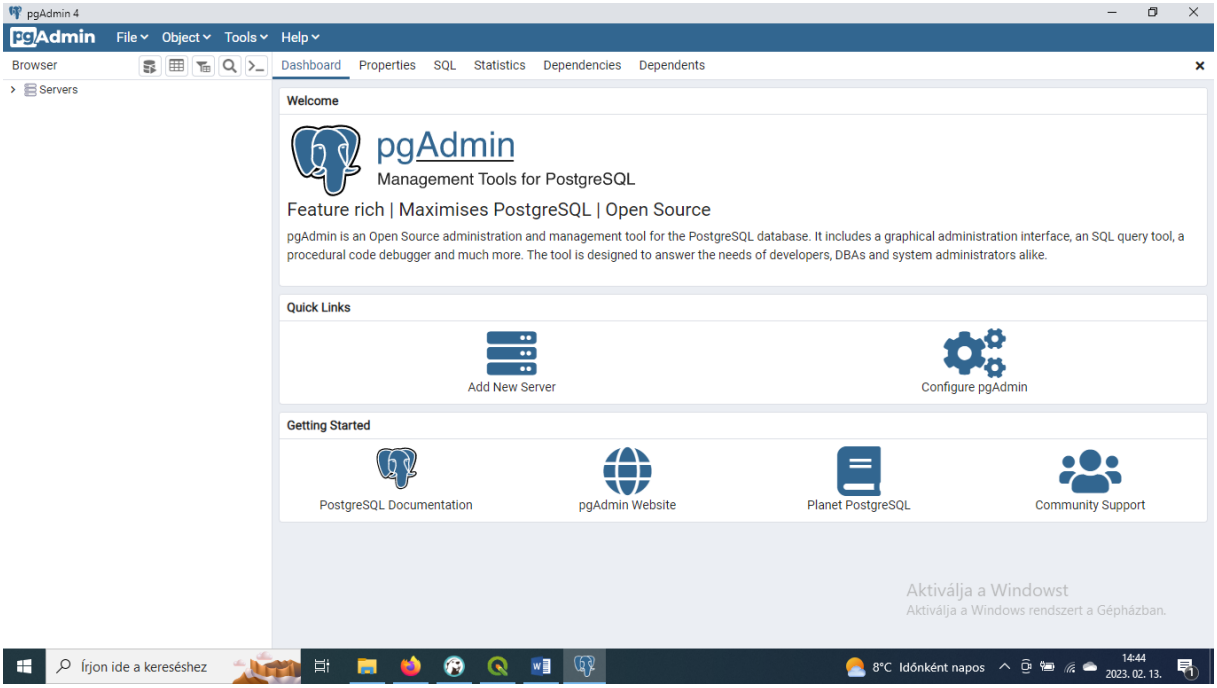
pgAdmin quick start guide

The following sections present some basics functions of pgAdmin.

Starting pgAdmin

First, launch pgAdmin. The initial step is logging in. You must remember the username and password you set during the installation process. After entering your password, you will gain access to your

databases. This is the welcome screen:



Navigating the Interface

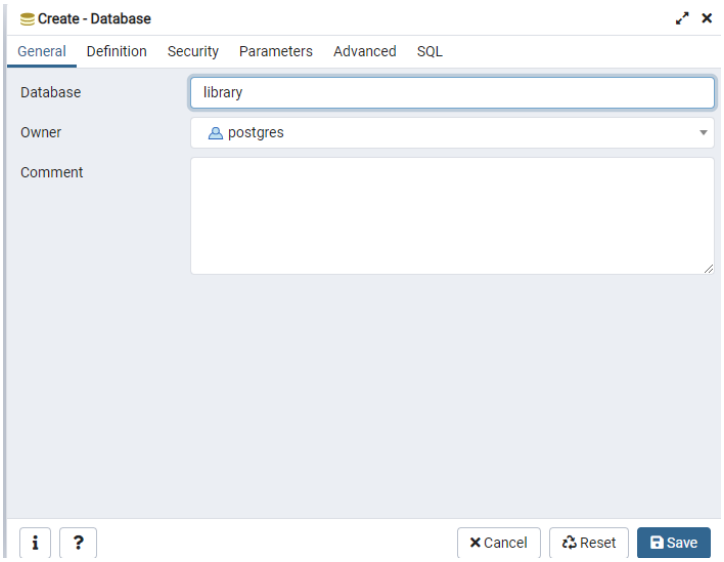
In the left column (**Object Explorer**), open the **Servers** menu. Under **Servers**, you can find the **Databases**. By default, there is always an empty, initial DB called *postgres*.

Importing the Library Database

Now, let's import the *library* DB from *library.sql* file.

Create a New Database

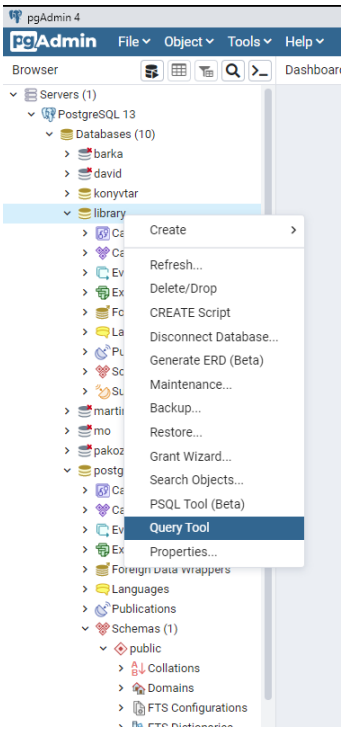
- Right-click on **Servers**, then select **Create → New Database**.
- A dialog window will appear.



- Enter a name for the database and click the **Save** button.
- The database is now created.

Executing SQL Queries

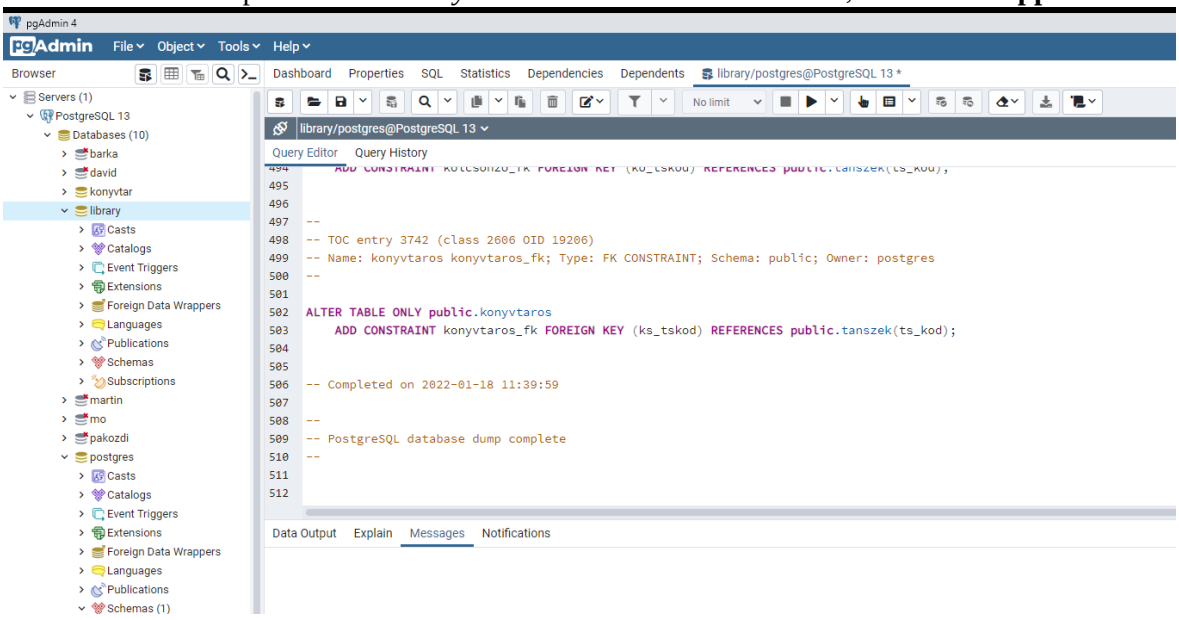
- Right-click on the *library* database and select **Query Tool**.
- Open the *library.sql* file in a text editor (e.g., Notepad++).
- Copy all the text from the file.



- Paste the copied text into the **Query Tool** window.
- Click the **Play** button to run the queries.

This set of queries will create the necessary tables in the *library* database, and populate them with data. Once the process is complete, you should receive the message: *'The query was successful*'.

For a detailed description of the *library* database structure and its tables, refer to the **Appendix**.

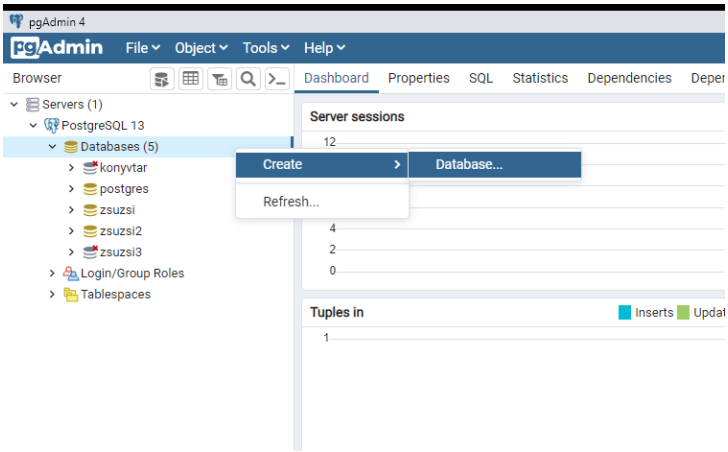


Importing a spatial database

In the following chapters, we will be working in DBeaver, a universal database management tool. However, DBeaver has a weakness: importing process of large size SQL files (30–50MB +) often results int the error message „*importing process has failed*”. To avoid this issue, I recommend using pgAdmin to import *hungary.sql*, our sample spatial database for this semester. pgAdmin is a more stable and robust tool for handling large imports.

Creating the Hungary Database

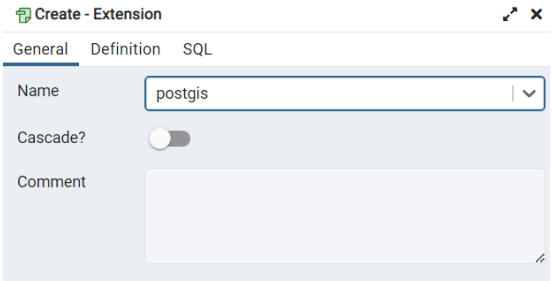
1. Open pgAdmin.
2. Right-click on **Databases → Create → Database**.
3. Enter *hungary* as the database name and click **Save**.



Adding the PostGIS Extension

Since the *hungary* DB contains spatial data, you must first enable the PostGIS extension before import ing the SQL file.

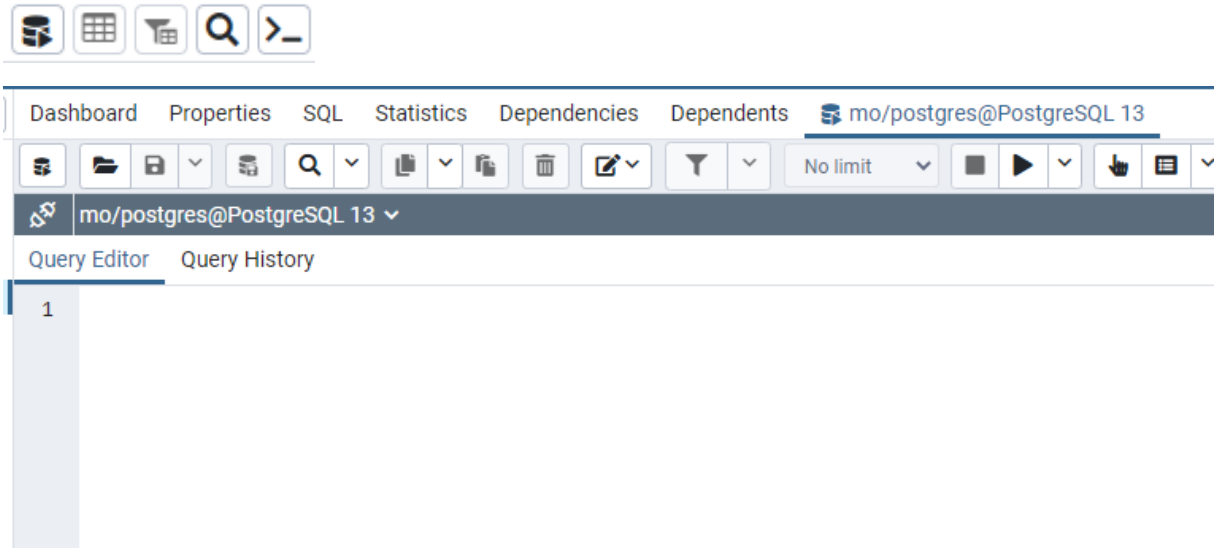
1. In pgAdmin, locate the **Extensions** section under your database.
2. Right-click on **Extensions → Create → Extension**
3. In the dialog window, search for **PostGIS** and select it.
4. Click **Save** to add the extension.



Importing the hungary.sql File

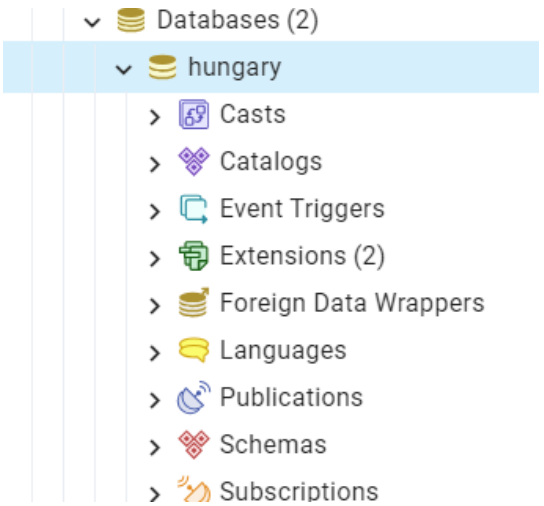
1. Open the **Query Tool**:
   a. Right-click on the *hungary* database and select **Query Tool**.
   b. Alternatively, you can click on the **barrel icon** in the toolbar or access it via **Tools → Query Tool**.
2. Open *hungary.sql* in a text editor (e.g., Notepad++).

3. Copy all the text from the file.
4. Paste the copied text into the **Query Tool** window.
5. Click the **Play** button to execute the queries. Wait for 20–30 seconds while the database is created and populated with data.

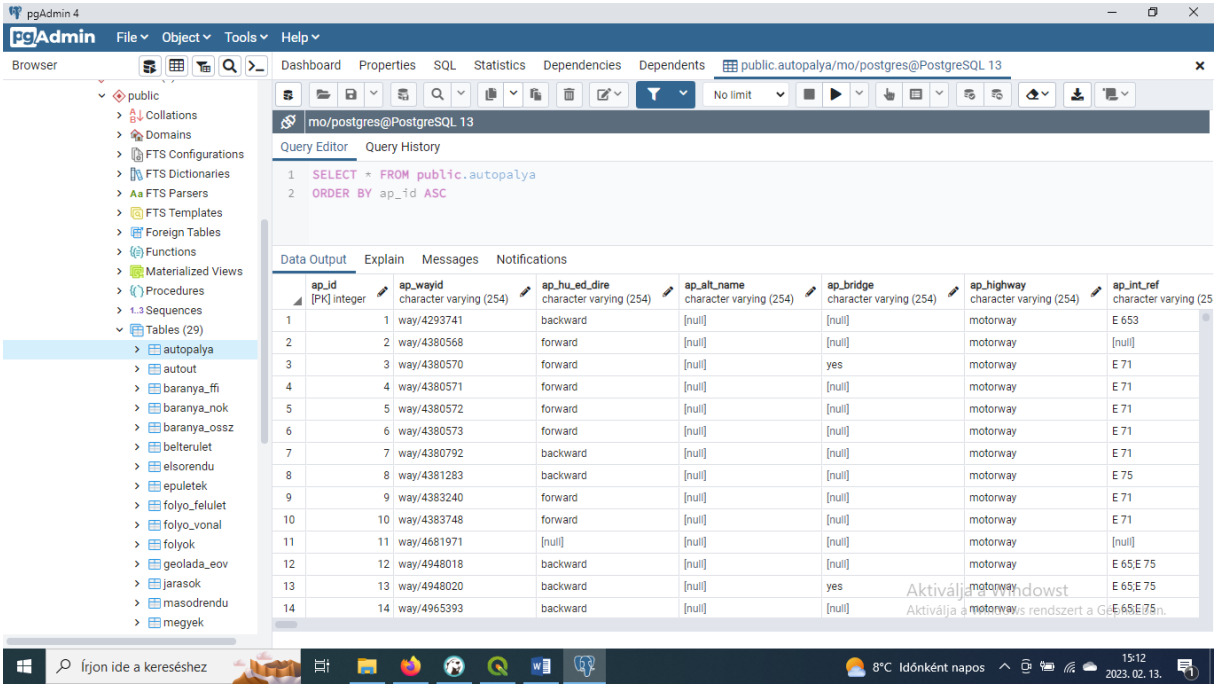


Viewing Databases in pgAdmin

Once the database is imported, you will see the following structure in the database tree. The **Schemas** contains the tables of the DB. To view a table, select it and click on the table icon above the **Object Explorer**. Browser



A `select *` query will run automatically, displaying all columns (fields) and rows (records). This is the **Data Output** section.

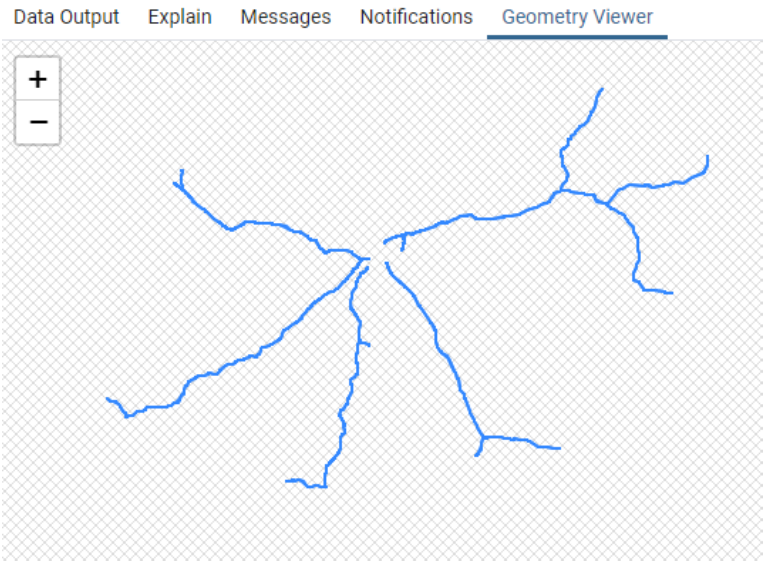


	ap_id [PK] integer	ap_wayid character varying (254)	ap_hu_ed_dire character varying (254)	ap_alt_name character varying (254)	ap_bridge character varying (254)	ap_highway character varying (254)	ap_int_ref character varying (25
1	1	way/4293741	backward	[null]	[null]	motorway	E 653
2	2	way/4380568	forward	[null]	[null]	motorway	[null]
3	3	way/4380570	forward	[null]	yes	motorway	E 71
4	4	way/4380571	forward	[null]	[null]	motorway	E 71
5	5	way/4380572	forward	[null]	[null]	motorway	E 71
6	6	way/4380573	forward	[null]	[null]	motorway	E 71
7	7	way/4380792	backward	[null]	[null]	motorway	E 71
8	8	way/4381283	backward	[null]	[null]	motorway	E 75
9	9	way/4383240	forward	[null]	[null]	motorway	E 71
10	10	way/4383748	forward	[null]	[null]	motorway	E 71
11	11	way/4681971	[null]	[null]	[null]	motorway	[null]
12	12	way/4948018	backward	[null]	[null]	motorway	E 65;E 75
13	13	way/4948020	backward	[null]	yes	motorway	E 65;E 75
14	14	way/4965393	backward	[null]	[null]	motorway	E 65;E 75

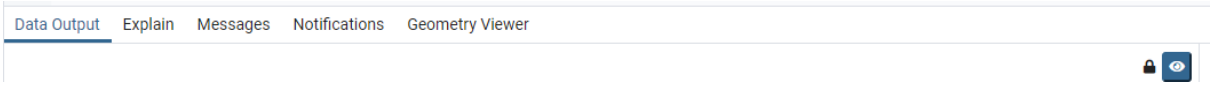
Viewing Databases in pgAdmin

If the table contains geometry data, click on the geometry column header to open the **Geometry Viewer**. If the geometry projection is EPSG: 4326 (Plate Carée projection on WGS84 ellipsoid), an OpenStreetMap (OSM) basemap will appear in the background. This projection works with geographic coordinates (degrees). If the projection uses meter units, such as in the *motorway* table, the OSM basemap will not be visible.

	ap_geom geometry
1	0105000020945C0000010000000102000000090000076579A6E4CAC1C4118BDDDF2AE35FE40DAEBD192B7AC1C41BC7D80CB5832FE40DD518A553BAD1C41E6802B59642EFE4078F
2	0105000020945C0000010000000102000000020000031432DDEFF132441B4EE30E760330E41A1419E41EF152441A4A53418763A0E41
3	0105000020945C0000010000000102000000060000C5AE3ECD8DAB24415A31B409D8E50E41954A9800A9AB244109EC31B73BE60E41D2257AEC08AC2441A699285386E70E415EEF
4	0105000020945C0000010000000102000000170000004869AD2F2CAD2441313F0A55E2EA0E4123B2C7F3A8AD244157F9FA4612EC0E410B498743F4AD24413E6143F3BCEC0E41D522



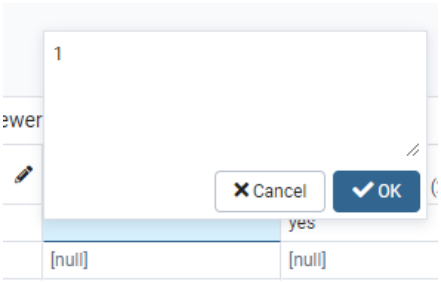
To exit the **Geometry viewer**, returnn the **Data Output** tab.



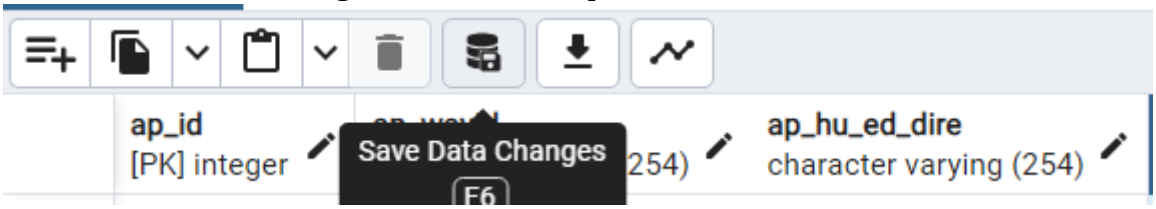
Editing Data in Tables

To modify data, click the **small pencil icon** in the field header.

A small editor window will open. Make the necessary changes.

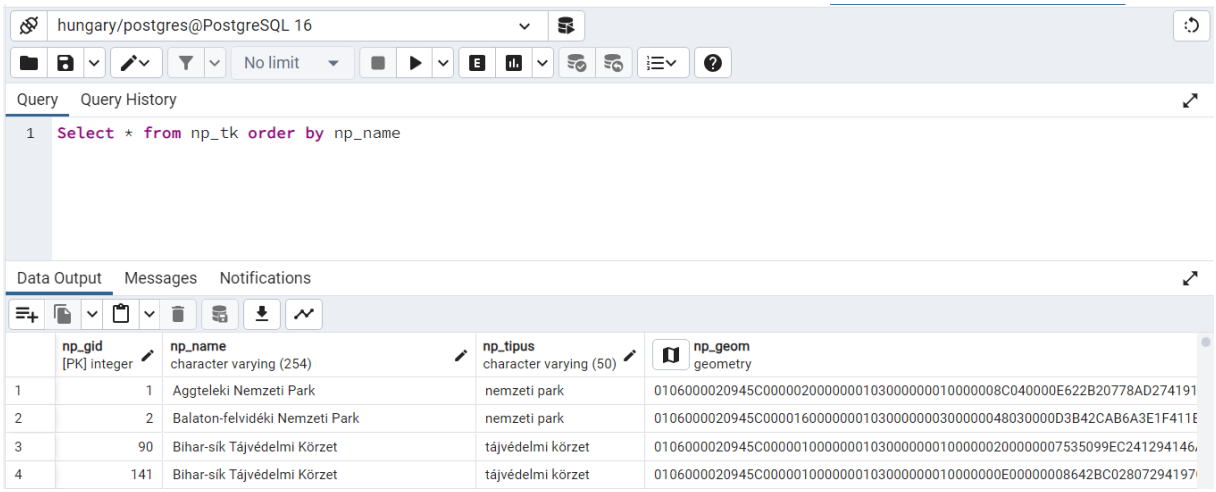


Click **Save the Data Changes**. in the **Data Output** toolbar.



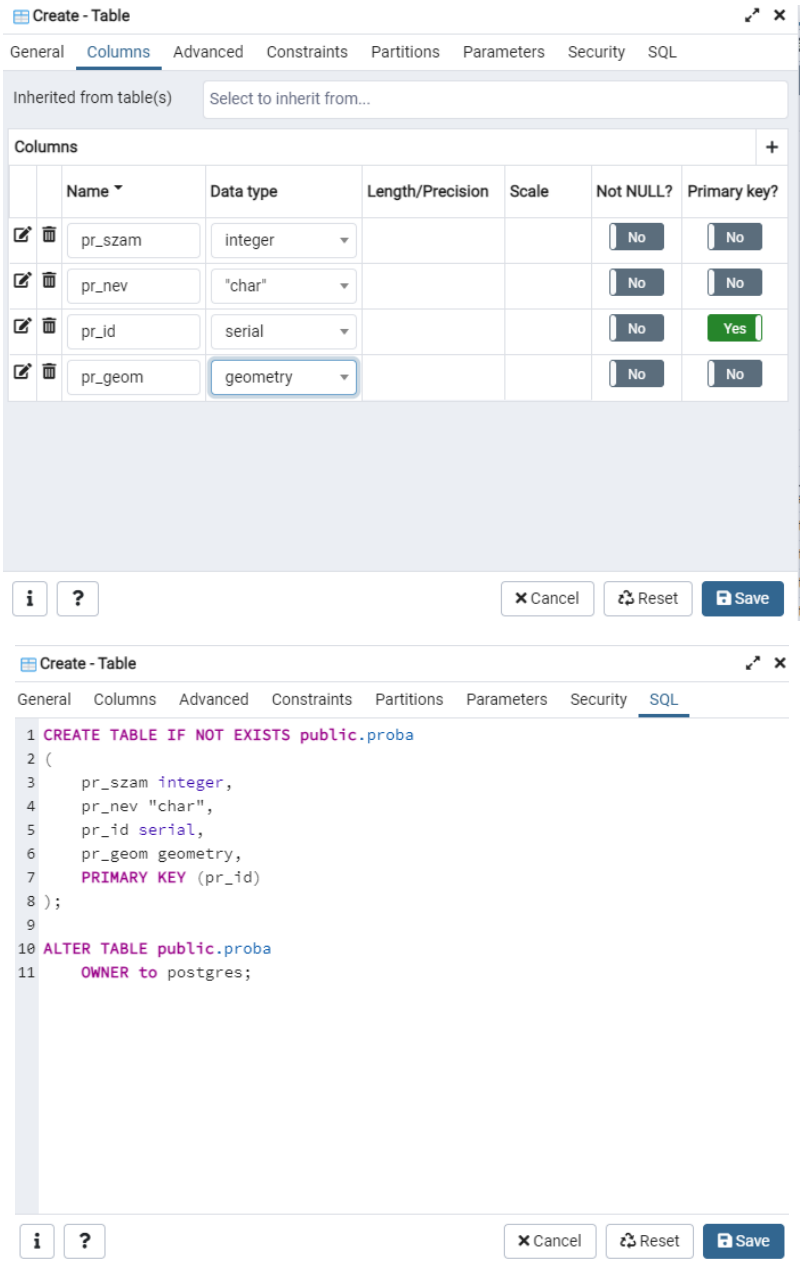
Running Queries

Open the **Query Tool** as described earlier. Type your SQL query. Click the **Play** button to run it.



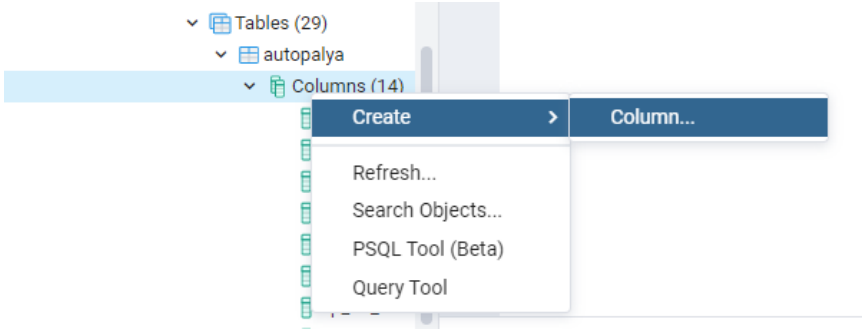
Creating a New Table

Right click on **Tables → Create → Table** In the first tab, enter the table name. Navigate to the **Columns** tab and add field name, data type and other properties (e.g primary key, NULL constraints, etc.). The SQL tab will display the generated SQL query. Click **Save** to create the table.



Adding New Column

Expand the **Columns** section in **Object Explorer**. Right click on → **Create** → **Column**. In the dialog window, enter the column name (**General** tab); select the data type (**Definition** tab). Configure additional properties if needed. The SQL tab will show the generated SQL query. Click **Save** to apply changes.



Deleting Columns or Tables

To remove a column or table: right-click on the column/table name and select **Delete**.

Modifying Column Properties

To change column properties (e.g., name, type): Right-click on the column name and select **Properties**. Modify the desired attributes in the dialog window.

Refreshing the Database View

If changes are not visible, then right-click on **Tables** or **Database** and select **Refresh**.

Chapter 2: DBeaver, the free universal database tool

DBeaver Community is a free cross-platform database tool for developers, database administrators, analysts, and everyone working with data. It supports all popular SQL databases like MySQL, MariaDB, PostgreSQL, SQLite, Apache Family, and more.[2]

Advantages:

- Basic support for relational databases: MySQL, SQL Server, PostgreSQL and others
- Data Editor
- SQL Editor
- Database schema editor
- Basic ER Diagrams
- Basic charts
- Data export/import
- Task management
- Database maintenance tools

We will use this software to handle spatial databases (PostgreSQL+PostGIS).

You can download it from here: https://dbeaver.io/download/

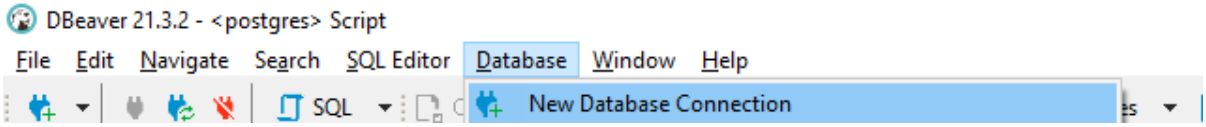
Windows installation:

Installing DBeaver on Windows is straightforward:

1. Download the installer from the link above.
2. Run the setup file.
3. Clic***k* Next** and accept the default settings when prompted.

Creating a New Connection in DBeaver

When you start DBeaver for the first time, you will see the **New Connection** wizard. You can also access this menu anytime via **Database → New Database Connection**.

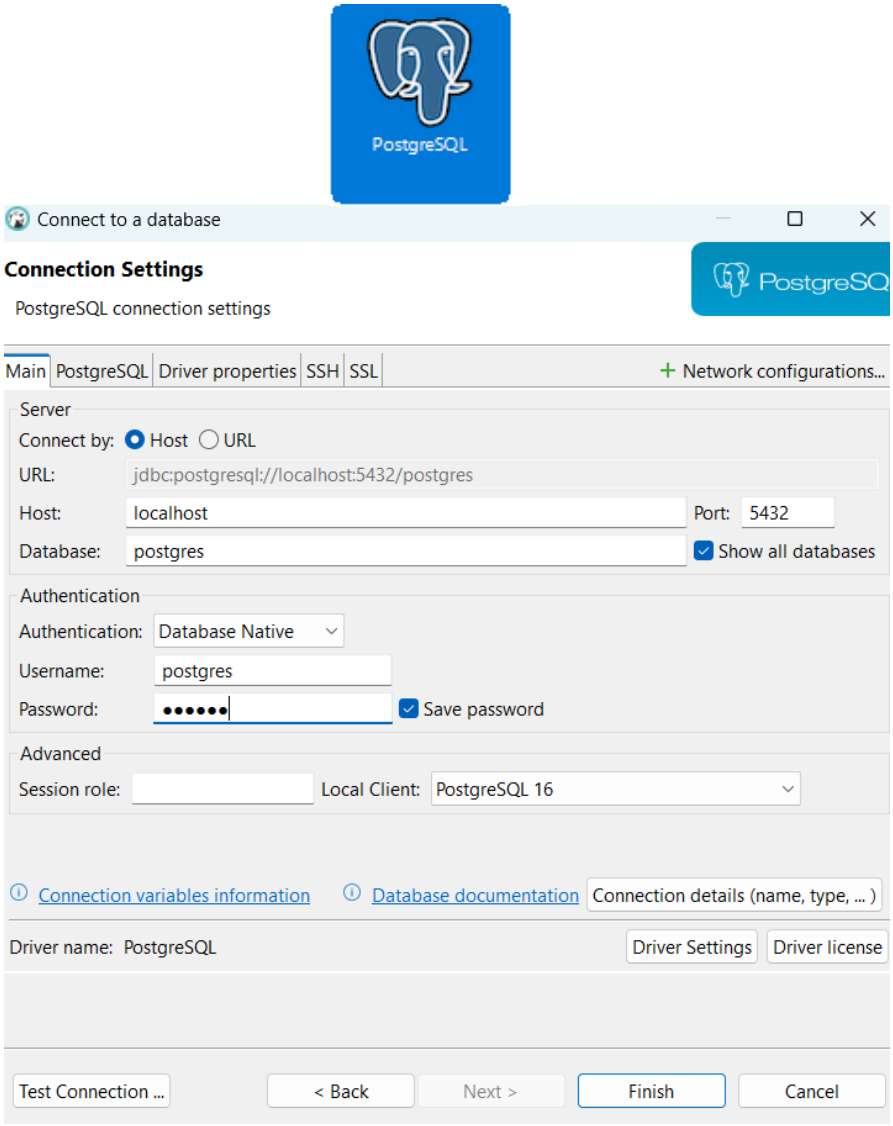


Setting Up a PostgreSQL Connection

1. Select PostgreSQL and click **Next**.
2. Enter the following details: Server name localhost; port 5432 (default), username and password (same as in pgAdmin)
3. Check the **Show all databases** option.
4. Click **Finish** to complete the setup.

Note: The first time you connect to PostgreSQL, DBeaver will download the necessary drivers. Allow DBeaver to proceed with the download.

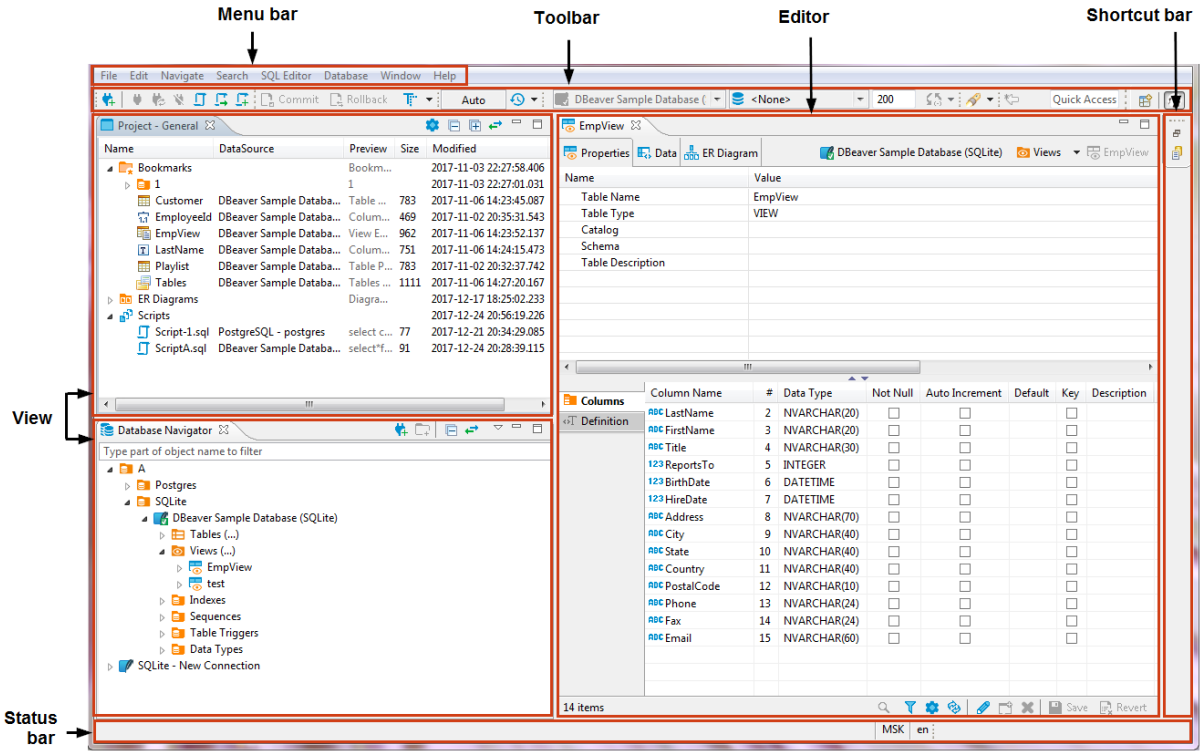
[2] https://dbeaver.io/

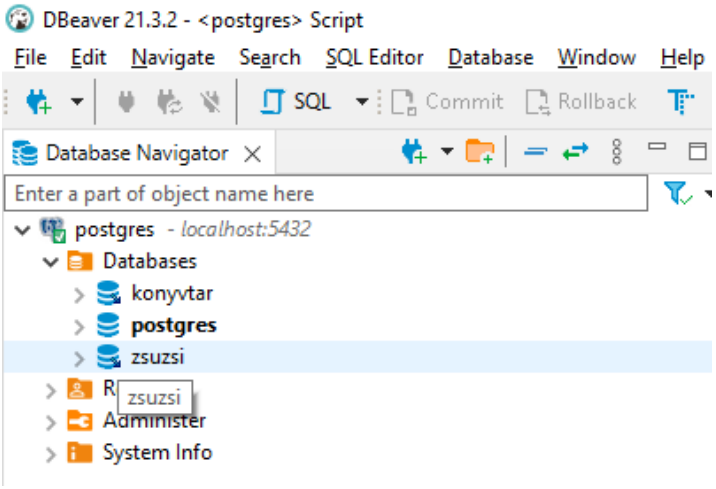


DBeaver User Interface

DBeaver has a user-friendly interface, where frequently used menus are easily accessible. The figure below illustrates keyy components of the user interface. **Database Navigator** displays the DBs, you are connected to. **Editor** allows you to browse tables, view data, access the **ER Diagram**, and run queries.

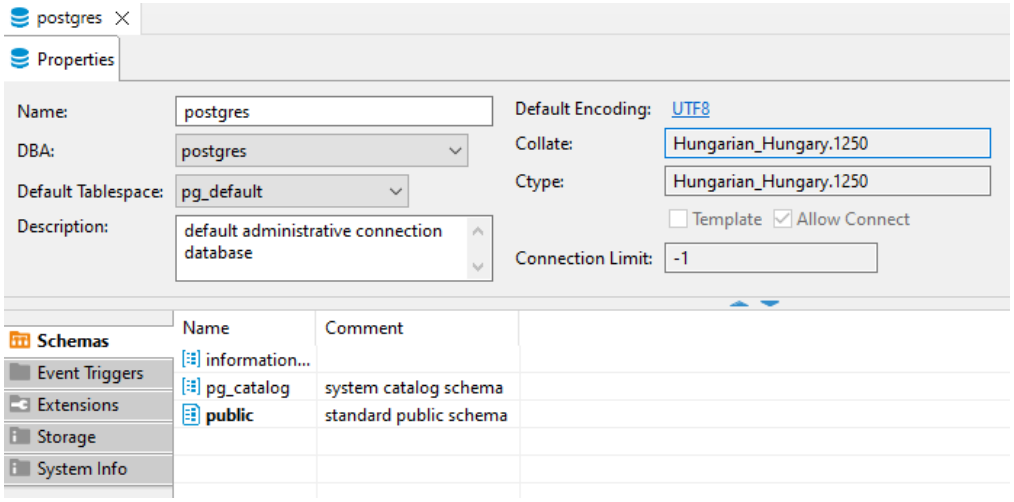
A double-click in the **Database Navigator** opens the corresponding database, table, or column sheet.





Database Stucture in DBeaver

When you open a database sheet, you can modify the database name and collation (character encoding, case sensivity, accent sensitivity, aplhabetical order based on language). The default encoding is UTF-8.

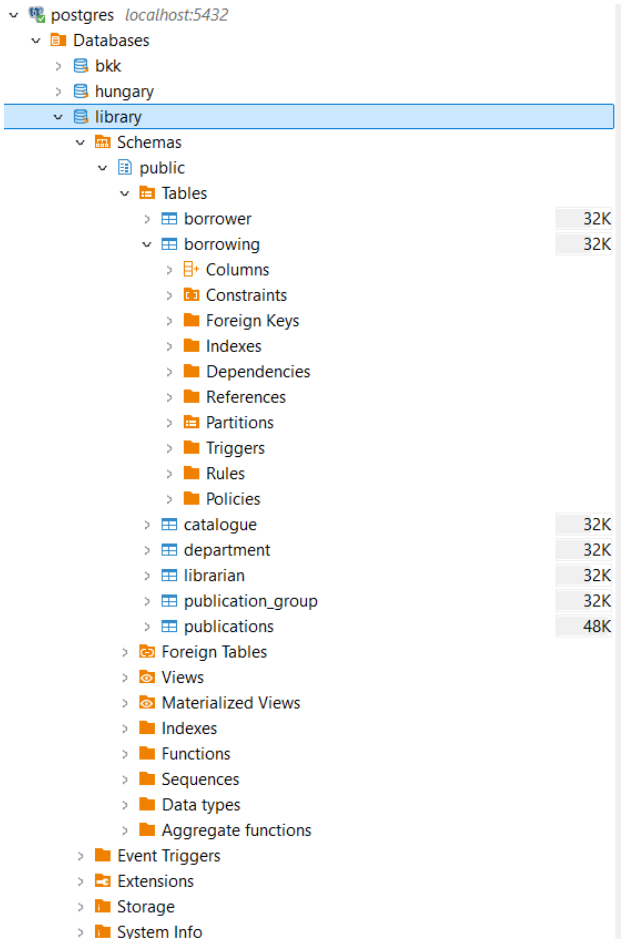


When you open a database, two important submenus appear in the navigation tree:

1. **Schemas → public** (default database schema). A Schema defines how data is organized within a relational database. It includes tables, fields, data types, and their relationships.
2. **Extensions** displays additional modules installed on your system. PostGIS is a key extension for handling spatial data (geometries).

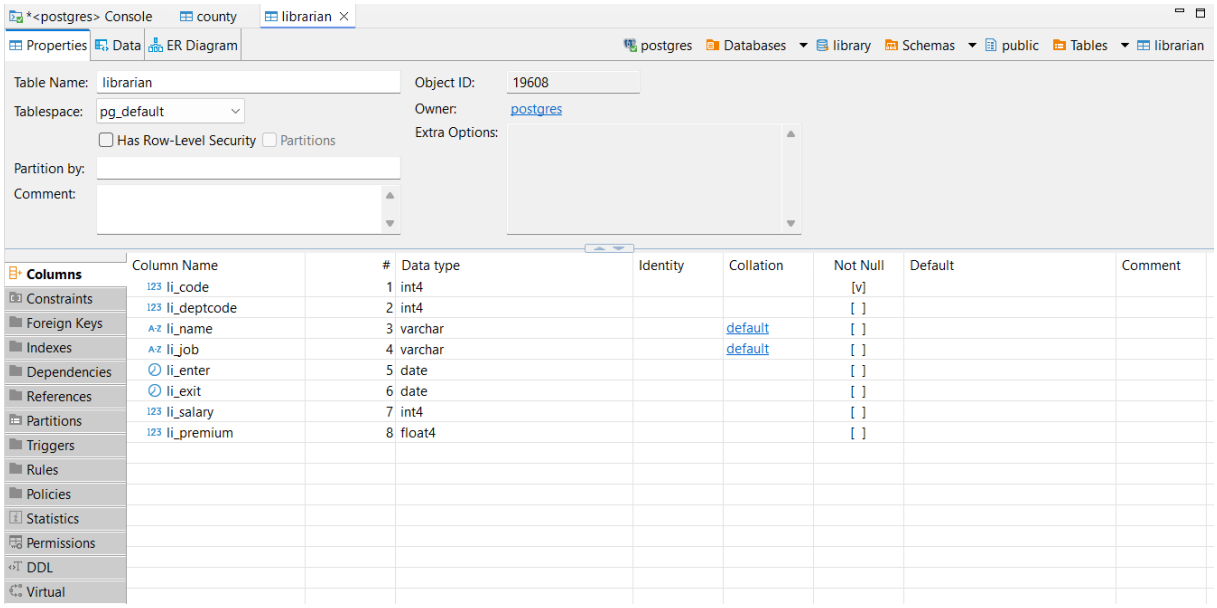
   To add a new extension: right-click on **Extensions → Create New Extension**. If PostGIS is installed, find it in the list and enable it.

Viewing and Managing Tables in DBeaver

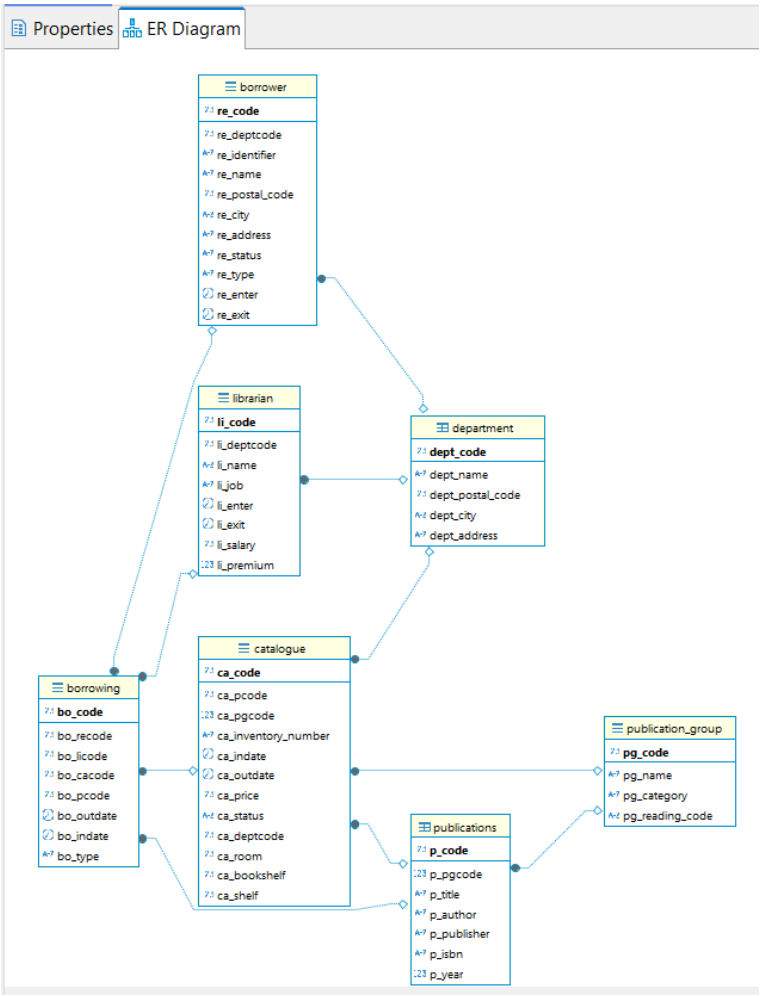


If you open **Schemas**, you can access **Tables**. To see the table details, make a double-click on table name or right click and **View Table**. This opens the table sheet, which consists of three tabs:

1. **Properties**: Table structure, settings, and metadata.
2. **Data**: View and edit table records.



3. **ER Diagram**: Visual representation of table relationships. The ER Diagram provides an overview of database structure: it displays tables, columns, data types, primary and foreign keys. If the database contains primary and foreign keys, they will be represented in the ER Diagram.



Working with Table Columns

The **Columns** tab displays field names, data types, encoding settings, and NULL constraints. There are some submenus in the right of Editor window.

1. **Tables**: To create a new column: Right-click → **Create new column**. Here you can set the fields of the table with data type, character encoding and `Null` or `Not Null` values.

Always **Save** changes after modifying the table structure!

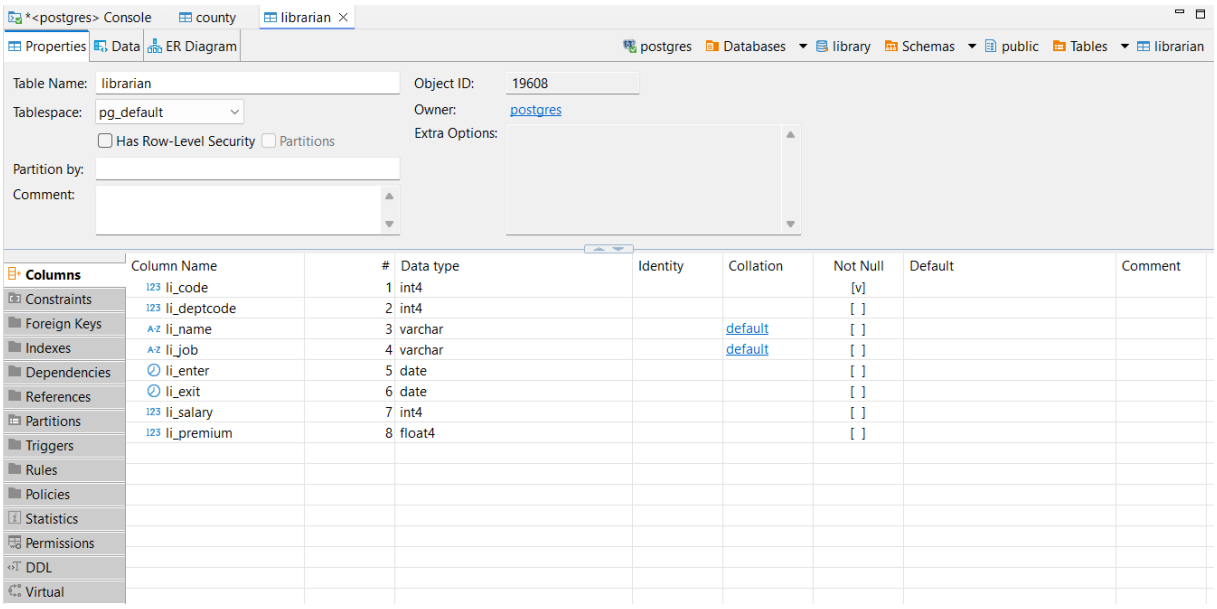
Constraints: Primary key or unique key ensures data integrity.

Foreign keys: Defines relationships between tables.

Cascade delete and update: If a primary key value is modified or deleted, related foreign key records will also be updated/deleted.

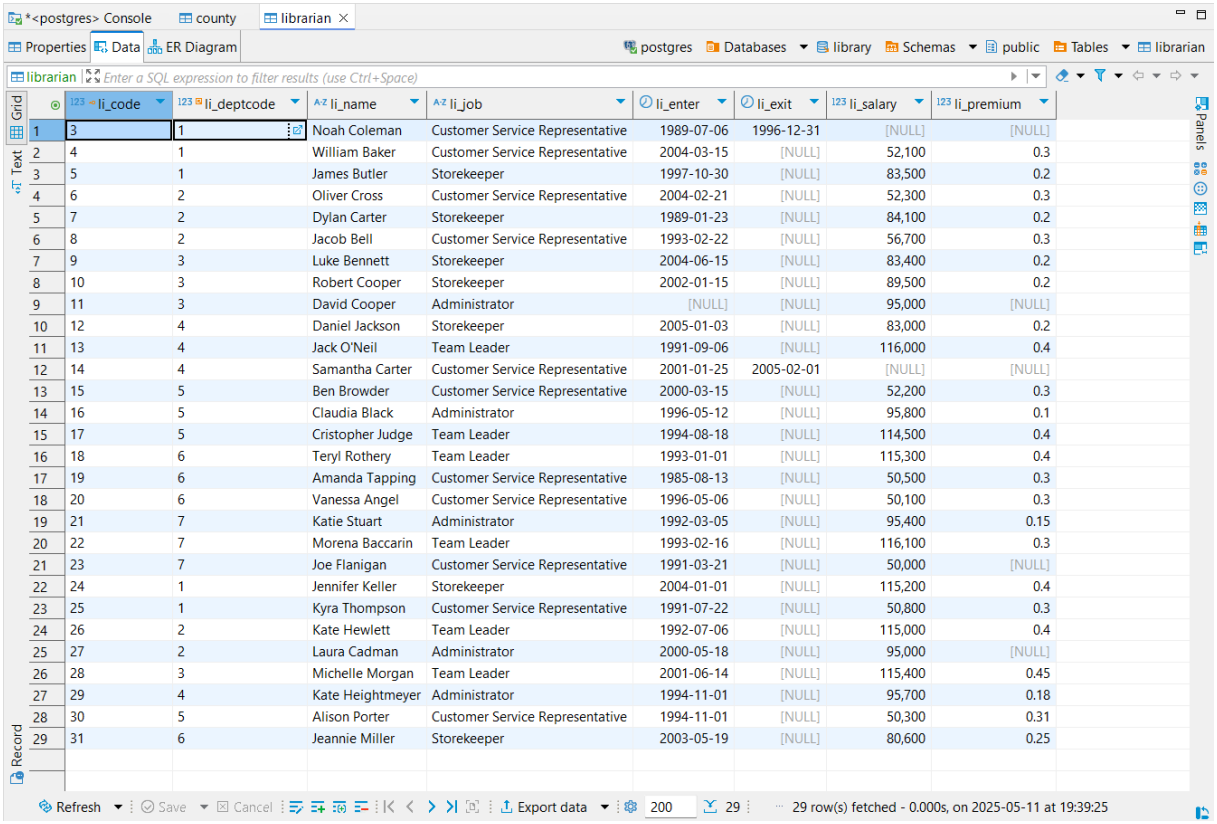
Restricted delete and update: Prevents modifications or deletions in the parent table if they would leave orphaned foreign keys.

Indexes: Improves search performance.

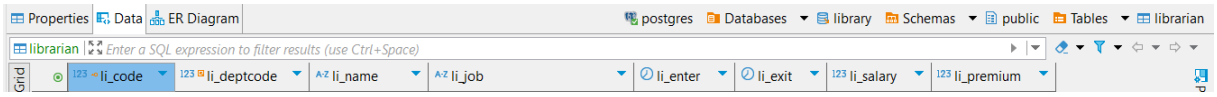


Editing Table Data

Each cell in the **Data** tab is editable. **If you made any modifications, do not forget to save the changes!** After modifying data, always click **Save** at the bottom of the Editor.



Filtering Data

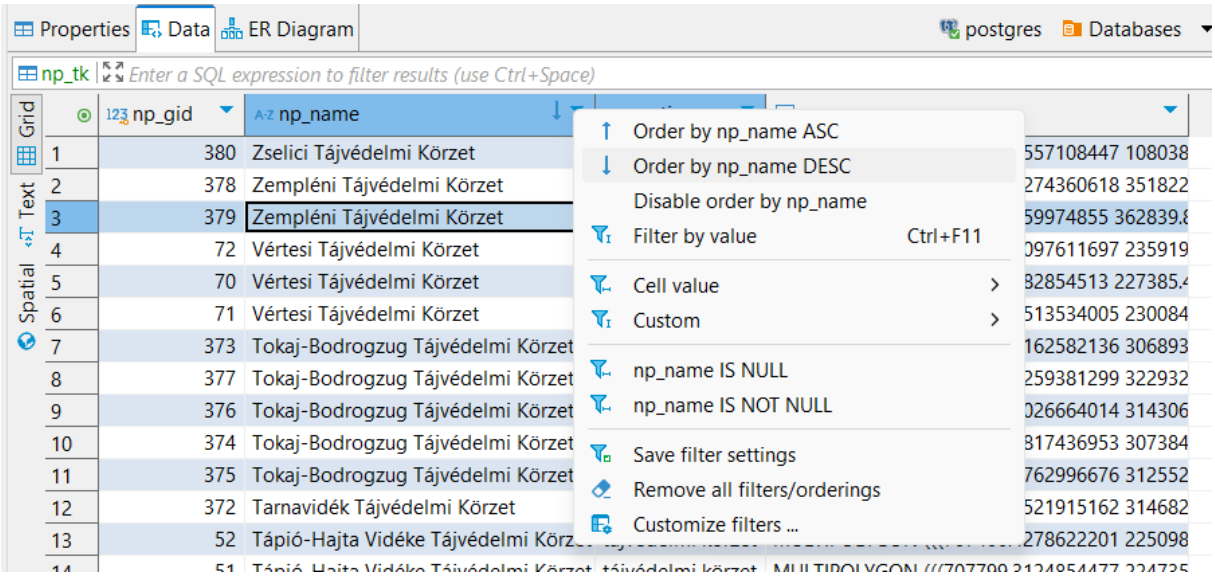


Below the **Properties** and **Data** tabs, you can filter data using SQL conditions.

```
select * from katalogus where CONDITION
```

The first part (`select * from katalogus`) is pre-written, you only need to **fill in the WHERE condition**. Additional filtering options:

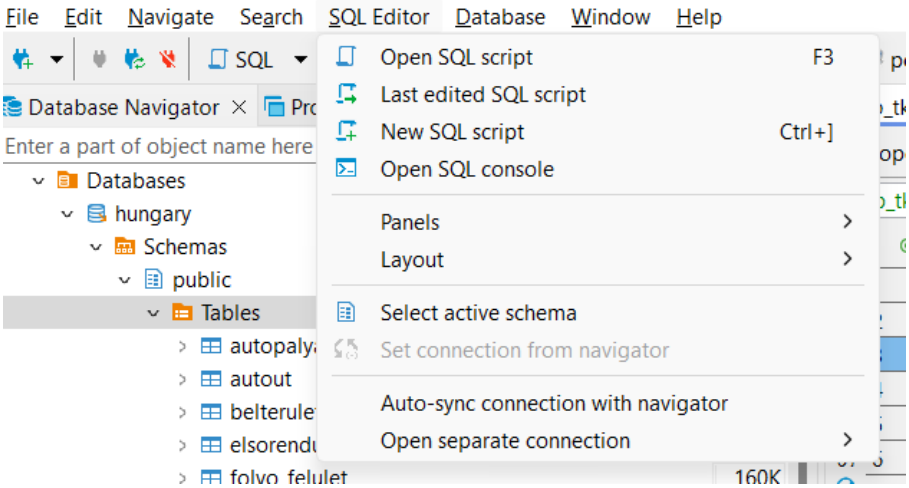
- Sort data (ascending/descending) → Right-click the column header.
- Filter by column value → Click the **funnel icon** in the column header.



Querying in the SQL Console

To execute SQL queries, open the **SQL Console**: It is available in the **Menu bar → SQL Editor → Open SQL Console**, or right-click on Tables → **Open SQL Console or Icon bar →** Click on the **SQL icon → Open SQL Console**

Important note: Always set the focus on Schema Public before running queries, otherwise you may get a "failed" error message.



To execute a query, click the orange **Play** button. 

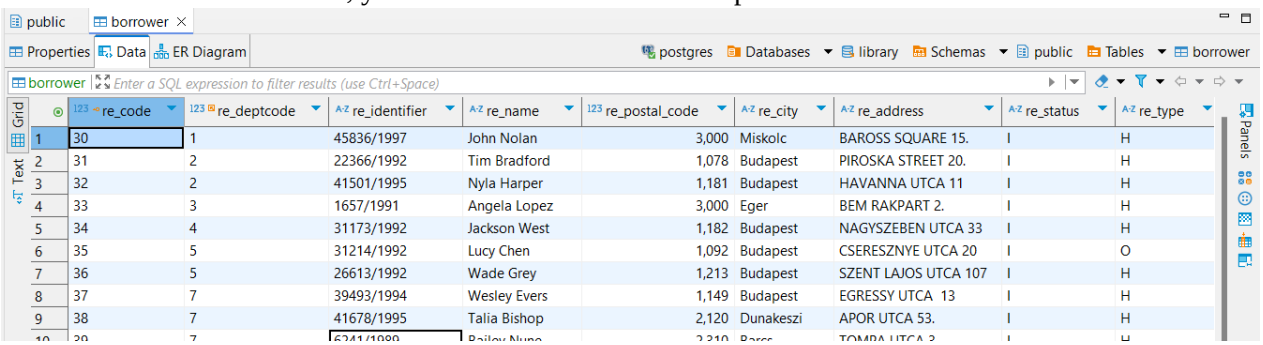
Now, let’s try an SQL query in the next Chapter.

Chapter 3. Basics SQL

SQL Basics 1

Use the Chapter 4. to check the correctness of your queries.

1. Get familiar with the library database! Open each data tables and review its content! At the end of these lecture notes, you will find a reference and explanation for all columns.



	re_code	re_deptcode	re_identifier	re_name	re_postal_code	re_city	re_address	re_status	re_type
1	30	1	45836/1997	John Nolan	3,000	Miskolc	BAROSS SQUARE 15.	I	H
2	31	2	22366/1992	Tim Bradford	1,078	Budapest	PIROSKA STREET 20.	I	H
3	32	2	41501/1995	Nyla Harper	1,181	Budapest	HAVANNA UTCA 11	I	H
4	33	3	1657/1991	Angela Lopez	3,000	Eger	BEM RAKPART 2.	I	H
5	34	4	31173/1992	Jackson West	1,182	Budapest	NAGYSZEBEN UTCA 33	I	H
6	35	5	31214/1992	Lucy Chen	1,092	Budapest	CSERESZNYE UTCA 20	I	O
7	36	5	26613/1992	Wade Grey	1,213	Budapest	SZENT LAJOS UTCA 107	I	H
8	37	7	39493/1994	Wesley Evers	1,149	Budapest	EGRESSY UTCA 13	I	H
9	38	7	41678/1995	Talia Bishop	2,120	Dunakeszi	APOR UTCA 53.	I	H

SQL is a standard language for querying, modifying and manipulating databases.

An SQL statement looks like this:

```
select COLUMN_NAME(S) from TABLE_NAME
```

After `select`, write the field name(s), or use * to retrieve all column in the result table.

2. List the identifier, purchase date, and the price for all publications from the catalogue table!

Ordering in the result table

In ascending order:

```
select COLUMN_NAME from TABLE_NAME order by COLUMN_NAME asc
```
or simply:

```
select COLUMN_NAME from TABLE_NAME order by COLUMN_NAME
```
In descending order:

```
select COLUMN_NAME from TABLE_NAME order by COLUMN_NAME desc
```

3. List the ISBN number, title and the author from the publications table, in descending order by title!

The WHERE condition

If you want to filter the data, use a `where` condition in your query.

```
select COLUMN_NAME from TABLE_NAME where CONDITION order by COLUMN_NAME
```

A condition consists of a column name, a relational operator and a value. Example:

```
ca_price>800
```

Relational operators include: `<, >, <=,>=, =`, and `<>` (not equal).

Values can be numbers (integer e.g. 12 or float/real e.g. 12.345), or text (e.g. 'this is a text type data') and dates (e.g. '2024-09-15' or '2024.09.15').

4. List the ISBN number and the year of publication from the publications table!
5. Query those catalogue items that cost more than 3,000 Forints. List the inventory number and price!
6. Query those catalogue items, that are cheaper than 3,000 Forints. Print the inventory number, the price and the publication group code. Order the results by price in descending order.
7. Query the primary key and the title of publications that do not belong to publication group number 3!

AND / OR operators

Sometimes you need to apply multiple conditions. Use `and` and `or` operators combine them.

`and`: All conditions must be true.

`or`: At least one condition must be true.

```
select COLUMN_NAME from TABLE_NAME where CONDITION1 and CONDITION2
select COLUMN_NAME from TABLE_NAME where CONDITION1 or CONDITION2
```

Exercises

8. Print the data of catalogue items that are more expensive than 9,000 Forints and are marked as non-borrowable!

9. Print all data of reserved items from „borrowing” table!

10. Print all data of catalogue items that were purchased before 2004-03-01!

11. Print data of loans (borrowings) that were made after 2004-03-01!

12. List the inventory number, publication code, and status code of catalogue items, belonging to publication group 4!

13. List the name, job and date of entering of the librarians, ordered by the date in ascending order!

14. Print all data from the librarian table. Order the result by job in ascending order, and by the librarian name in descending order.

DISTINCT

The `distinct` keyword in SQL is used to eliminate duplicate values in the result set. It ensures that only unique (non-duplicated) records are returned for the columns specified in the query.

```
select distinct COLUMN_NAME from TABLE_NAME
```

15. Query the distinct job titles from the librarian table! (Every job should appear only once.)

Practice

16. Query librarians with the job title „Storekeeper” and „Customer Service Representative”. Order the results by date of entering in descending order!

17. List the librarians whose salary is more than 50,000 Forints whose job is „Storekeeper”. Order the results by date of entering in ascending order.

18. List librarians whose salary is less than 60,000 or more than 100,000.

19. List librarians who work in the department 2 or 5.

20. List the departments that are not located in Budapest!

21. List the department's locations from the department table! Every locations should appear only once!

BETWEEN / NOT BETWEEN operators

The `between` operator selects values within a given range. The values can be numbers or dates.

The `between` operator is inclusive: beginning and end values are included.

`not between` returns the values that are outside of the given range.

Examples:

```
where column_name between value A and value B
where column_name not between value A and value B
```

22. List the librarians whose salary is between 50,000 and 100,000!
23. List the librarians whose salary is not between 50,000 and 100,000!

IN / NOT IN operators

The `in` operator allows you to specify multiple values in a `where` clause. `not in` returns the values that are not in the specified list.

```
where column_name in (value A, value B)
where column_name not in (value A, value B)
```

24. List the librarians whose job is Storekeeper and Team Leader!

25. List the librarians whose job is neither Storekeeper nor Team Leader!

Testing for NULL value (no data value)

The `is null` operator is used to check if a column contains a `null` value. It is commonly used in `where` clauses to filter records where a column has no value (i.e., it is `null`).

`is not null` returns those records where the column contains a value.

```
where column_name is null
where column name is not null
```

SQL ALIASES

SQL aliases are used to temporarily rename a column or table. They are often used to make column names more readable and exist only for the duration of the query.

```
column_name AS new_column_name
```

26. Which publications were returned? (Check where there is a date in bo_indate column in the borrowing table?)

27. Display the name, the job and the income (salary+premium calculates as a percentage of salary) of the librarians! Rename the column to salary.

LIKE / NOT LIKE Operators

The `like` operator is used in a `where` clause to search for a specified pattern in a column.

- `%` represents zero, one, or multiple characters
- `_` represents a single character

'NOT LIKE' returns the results that do not match the specified pattern.

```
where column_name like 'j%'
where column_name not like 'j%'
```

28.List the names of librarians whose names start with J!

29. List the names of librarians whose second character is A!

30. List the names of librarians whose last character is R!

31. List the names of department that contain 'geo'!

Let's do these excercises

32. List the librarians whose job is Storekeeper and Team Leader and whose salary is between 60,000 and 100,000!

33. List the librarians who do not receive a salary!

34. List the publications whose title contains 'atlasz'!

35. Create a list about the catalogue items with a price between 10,000 and 30,000 and a purchase date before 2002-01-01!

36. List the publications that have no author and were published in the year 2000.

Simple functions

MySQL offers many built-in functions that help in working with different kinds of data such as text, number (integers and floats) and dates.

What does a function look like?

```
functionname(value1, value2, etc.)
```

Examples:

```
Upper('dog')  → and the result is: DOG.
Lower('CAT') → cat
```

`Lower` → Converts a string to lowercase,

`Upper` → Converts a string to uppercase.

37. Display the titles of publications in lowercase, and the publishers in uppercase!

CONCAT

`Concat()` joins two or more strings into one.

Example:

```
Concat('large ','black ','dog') → large black dog
```

38. Create a list of borrowers from Eger! Show the name, the identifier (ID), and the date of entering. Add the word 'CUSTOMER' before each name.

CHAR_LENGTH

`Char_Length()` returns the number of characters in a string.

```
Char_Length('dog') → 3
```

39. Show the length of each borrower's name!

SUBSTRING

`Substring(string, start, length)` extracts a part of a string. The length is optional; if omitted, it returns the rest of the string.

Examples:

```
Substring('black cat', 3, 2) → ac
Substring('black cat', 3) → ack cat
```

LEFT and RIGHT

`Left(string, n)` → Returns the first n characters of a string.

`Right(string, n)` → Returns the last n characters of a string.
Examples:

```
Left('black cat', 3) → bla
Right('black cat', 3) → cat
```

STRPOS

In PostgreSQL, `StrPos(string, substring)` returns the position of a substring in a string.

```
StrPos('black cat',' ') → 6
```

Tasks:

40. Print characters from 3 to 7 from the borrower's name!

41. Print borrower's name starting from the 5th character!

42. Print the first 5 characters of the borrower's name!

43. Print the last 5 characters of the borrower's name!

44. Print the position of the 'space' character from the borrower's name!

45 Print the borrower's first and last name in separate columns.

46. Same as 45, but display the first name in lowercase and the last name in uppercase.

47. Show the length of the publication titles using char_length and octet_length! Explain the difference.

Math functions

48. Run and explain these functions!

`Abs(1000)`, `Abs(-100)` and `Abs(-27.11)` → Returns the absolute value of a number

`Sqrt(25)`, `Sqrt(-36)` → square root (watch for invalid input).

`Power(2,5)` → Power (2^5 = 32)

`Round(-123.456,2)`, `Round(123.456,0)`, `Round(123.456,-2)`, `Round(-123.4567,2)` → Rounding with different precision. It rounds a number to a specified number of decimal places (the second parameter is positive number). If the second parameter a negative number, it rounds to tens, hundreds, thousends etc. 0 round to integer.

`SIGN(10), SIGN(0), SIGN(-5)` → Returns the sign of a number.

`CEIL(6.1), CEIL(-6.1)` → Round up to the nearest integer

`FLOOR(6.1), FLOOR(-6.1)` → Round down to the nearest integer

`PI()` → 3.1415…

Division

49. Try the following and explain:

10/3 What do you get, integer or decimal number? - Use ::decimal or ::integer to declare the data type!

10%3 What is this? (Remainder (modulus))

Date functions

You can subtract dates to get the number of days between them.

```
'2024-09-27'-'2024-09-20' → 7 days
```

NOW()

Returns the current date and time with time zone:

```
2025-02-14 12:58:02.623 +0100
```

Tasks:

50. Show borrowing records where the duration was less than one week!

51. Show borrowing records where the duration was more than one week.

52. How many days have passed since your birth = How old are you in days?

EXTRACT

Used to get specific parts of a date or time:

```
Extract(year from Now())
Extract(month from Now())
```

```
Extract(day from Now())
Extract(hour from Now())
Extract(minute from Now())
Extract(second from Now())
Extract(dow from Now()) or Extract(dow from DATE '1989-01-13')  → day of week,
Extract(timezone_hour from now())
```

53. Extract the year/month/day/hour/minute/second from today!

54. Extract only the days from the query in task 50.

55. What day of the week is today?

56. What day of week were you born?

Group By

The `group by` statement groups rows that have the same values into summary rows, like "find the number of customers in each country". An aggregate function is a function that performs a calculation on a set of values and returns a single value. Aggregate functions are often used with the `group by` clause of the `select` statement. The `group by` clause splits the result-set into groups of values and the aggregate function can be used to return a single value for each group.

Common aggregate functions:

The `group by` statement is often used with aggregate functions (`Count(), Max(), Min(), Sum(), Avg()`) to group the result set by one or more columns.

- `Count()` → Returns the number of rows that match a condition.

Most aggregate functions ignore NULL values (except `Count(*),` which includes all rows).

- `Min()` → Returns the smallest value in a column.
- `Max()` → Returns the largest value in a column.
- `Sum()` → Returns the total of a numeric column.
- `Avg()` → Returns the average of a numeric column.

Tasks

57. Display the minimum, maximum, average, and the total salary from the librarian table! How many librarians are there?

58. What was the lowest and highest price in the catalogue table?

59. How many borrowers are from Budapest?

60. Show the oldest and the most recent purchase dates in the catalogue table!

61. How many different jobs are there in the librarian table?

The order of clauses are the following:

```
select… from … where… group by… having… order by …
```

62. Display the minimum, maximum, average, and the total salary from the librarian table! Group the results by job title. Also show how many librarians are in each group.

63. Do the same as in Task 62, but group by department code. Order the results by department code.

64. Show the code of each publication group, and the number of publications in each group from the catalogue table!

65. Show the number of publications and the average price for each department from the catalogue table! Group the data by department!

66. What was the average borrowing duration (in days) for all borrowings before 2004-05-30?

67. What is the average borrowing duration (in days) for the publication titled *KISATLASZ*? Group by librarian.

HAVING

The `having` clause is used to filter groups created by `group by`. Unlike `where`, it **can be used with aggregate functions**.

68. Show the minimum, maximum, average, and total salary from the librarian table, grouped by job. How many people are in the librarian table? Show only those groups where the average salary is greater than 80,000 Forints.

69. Show the code of departments, where there are exactly 3 librarians.

70 Group librarians by the length of their names. How many librarians are in each group? Also display the average salary. Order the results by average salary.

71. For each job title, calculate the difference between the maximum and minimum salaries.

LIMIT

`limit 1` – Returns only **one row** from the result set.

72. Show only the group with the biggest difference!

Table Joins

A `join` clause is used to combine rows from two or more tables, based on a related column between them A way to join tables, can be **primary key–foreign key** relationships definition. **Primary key** uniquely identifies each record in a table. n most cases, it is a sequential integer. **Foreign key** refers to the primary key of another table, creating a connection between the two.

Let's see an example in the library database. The department table contains a primary key (dept_code). The librarian table has its own primary key (li_code) and a foreign key (li_deptcode), which refers to the department table primary key (`department.dept_code`). This means that if *Noah Coleman* has `li_deptcode = 1`, and department 1 is "Department of Cartography", then Noah Coleman works in the Department of Cartography.

Relational databases define relationships between tables through **primary key – foreign key** pairs.

li_code	li_deptcode	li_name
3	1	Noah Coleman
4	1	William Baker
5	1	James Butler
6	2	Oliver Cross
7	2	Dylan Carter
8	2	Jacob Bell
9	3	Luke Bennett

dept_code	dept_name
1	Department of Cartography
2	Department of Geography
3	Department of Geophysics

Foreign key – primary key relationships in the Library database:

`dept_code – li_deptcode`: Which department the librarian works in (department and librarian tables)

`pg_code – p_pgcode`: the group code of a publication (publication_group and publication tables)

`pg_code – ca_pgcode`: the group code of a catalogue item (publication_group and catalogue tables)

`p_code – ca_pcode`: Link between publication and catalogue item (publication and catalogue tables)

`dept_code – ca_deptcode`: Which department owns the catalogue item (department and catalogue tables)

`dept_code – re_deptcode`: Which department the borrower belongs to (department and borrower tables)

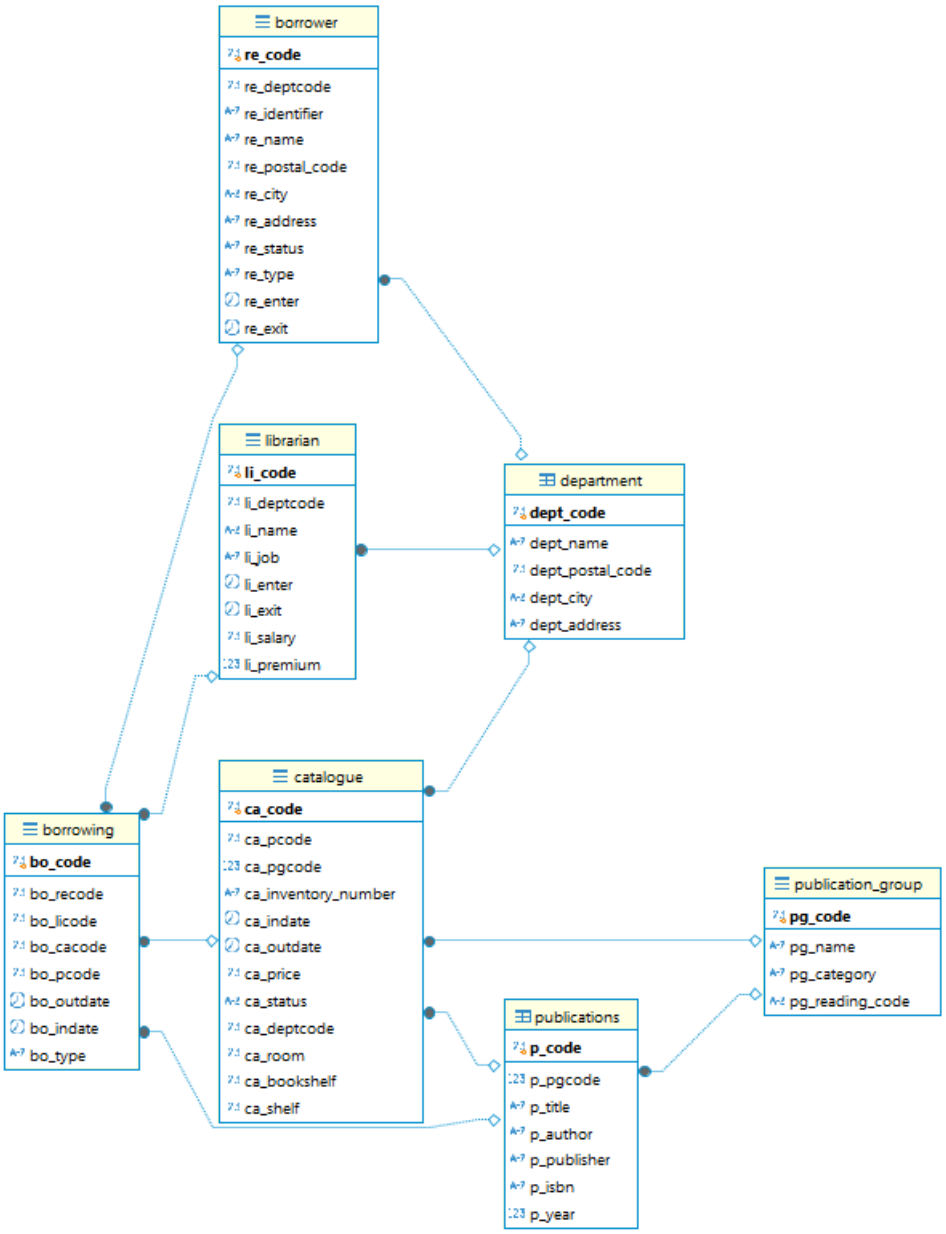
`re_code – bo_recode`: Which borrower made the borrowing (borrower and borrowing tables)

`li_code – bo_licode`: Which librarian handled the borrowing (librarian and borrowing tables)

`ca_code – bo_cacode`: which catalogue item was borrowed (catalogue and borrowing tables)

`p_code – bo_pcode`: which publication was borrowed (publication and borrowing tables)

The following diagram shows this structure: white diamonds represent primary keys, and filled circles represent foreign keys.



How to Write a JOIN Query

There are two standard ways to join tables. Do not mix them in one query!
Implicit join syntax (older style):

```
select FIELD_NAMES from TABLE1, TABLE2 where FOREIGN_KEY=PRIMARY KEY
```

Explicit JOIN syntax (recommended):

```
select FIELD_NAMES from TABLE1 join TABLE2 on FOREIGN_KEY=PRIMARY KEY
```

Examples

73. Display the librarian's name, job title and the name of the department where they work. Order the results by the librarian's name.

74. Display all librarians who work in Sopron.

75. Print the title, author and publisher of publications, which price is between 3,000 and 10,000 Forints!

76. Print the names of borrowers who reserved a publication (bo_type='F')!

77. Print the lowest and the highest salary, grouped by the department name!

78. Print the average salary and the number of librarians, grouped by city!

Joining Three or More Tables

To join three or more tables, identify the correct primary key–foreign key pairs between them.

Example syntax (implicit):

```
select FIELD_NAMES from TABLE1, TABLE2, TABLE3 where FOREIGN_KEY=PRIMARY KEY and FOREIGN_KEY=PRIMARY KEY
```

Example syntax (explicit join):

```
select FIELD_NAMES from TABLE1 join TABLE2 on FOREIGN_KEY=PRIMARY KEY join TABLE3 on FOREIGN_KEY=PRIMARY KEY
```

More Advanced Exercises:

79. Display the inventory number and price of a publication, the name of borrower, full address (city, postal code and address), and the borrowing date but only for borrowings before 2004-05-01.

80. Print the borrower's name, the title of the publication, the name of the librarian and the borrowin code!

81. Display the borrowed publication's title, inventory number, and borrower's name and the date of borrowing!

82 Print the names of borrowers who borrowed the publication titled 'KISATLASZ'. Show the borrower's name, title of the publication, and borrowing date.

83. Show the borrower's name and the borrowing date (in- and out dates) where the librarian's name was Oliver Cross.

Left and Right JOINS

The `left join` keyword returns all records from the left table (table1), and the matching records from the right table (table2). If there is **no match**, the result will still include the row from the left table, and `NULL` values for the right table.

The `right join` keyword does the opposite: it returns **all records from the right table**, and the **matching records from the left table**.
If there is **no match**, the result will still include the row from the right table, and `NULL` values for the left table.

For visual examples, visit: W3Schools SQL JOINs

84. Print the publications and their publication groups. If the group has no publications, it should still appear in the list.

Subqueries

A **subquery** is a query nested within another SQL query. Subqueries are often used inside the WHERE clause and are placed in **round brackets** ().

Let's take a look at the next example:

85. Print the librarians, whose salary is higher than that of of Alison Porter.

This can be solved in two steps: First, you have to know how much money Alison Porter earned. Then, you can compare this value with another librarian's salary.

```
select li_salary from librarian where li_name='Alison Porter'
→The value is 50,300.
select * from librarian where li_salary>50300
```

Or it can be solved in **one query using a subquery**:

```
select * from librarian where li_salary>(select li_salary from librarian where
li_name='Alison Porter')
```

More Subquery Exercises

86. Print the borrower's name and code, whose city is same as Aaron Thorsen's!

87. Print those publications, that are cheaper than the publication titled 'FÖLDRAJZI VILÁGATLASZ'

88. Print those librarians, whose job is the same as Samantha Carter's job.

89. Print those publications that were published earlier than 'Cartography Thematic Map Design'!

If the subquery returns multiple results, use the `IN` operator instead of =.

90. Print the code, name and job of the librarians whose job is the same as any librarian working in the Dept. of Cartography.

91. Print the departments that are in the same city as both the Dept. of Geophysics and the Dept. of Geodesy.

92. List borrowings where the price of the borrowed publication is greater than the average price.

93. Print the borrowings, where the borrowing time was the longest.

94. Print the borrowings, where the borrowing time was the shortest.

95. Print the publications where the title length is longer than hald of the longest title in the table.

96. Print those librarians who earn more than the average salary in Budapest.

97. Print the librarians from Budapest who earn more than the overall average salary.

Chapter 4. KEY TO BASIC SQL Queries

```
2. select ca_inventory_number, ca_indate, ca_price from catalogue
3. select p_title, p_author, p_isbn from publications order by p_title desc
4. select p_isbn, p_year from publications
5. select ca_inventory_number, ca_price from catalogue where ca_price >3000
6. select ca_inventory_number, ca_price,ca_pgcode from catalogue where ca_price
<3000 order by ca_price desc
7. select p_code, p_title from publications where p_pgcode !=3
8. select * from catalogue where ca_price>9000 and ca_status ='N'
9. select * from borrowing where bo_type='F'
10. select * from catalogue where ca_indate<'2004-03-01'
11. select * from borrowing where bo_outdate <'2004-03-01'

12. select ca_inventory_number, ca_code, ca_status from catalogue where ca_pgcode
= 4
13. select li_name, li_job, li_enter from librarian order by li_enter
14. select * from librarian order by li_job, li_name desc
15. select distinct li_job from librarian
16. select * from librarian where li_job='Storekeeper' or li_job='Customer Service
Representative' order by li_enter desc
select * from librarian where li_job in ('Storekeeper', 'Customer Service
Representative') order by li_enter desc
17. select * from librarian where li_job ='Storekeeper' or li_salary >50000 order
by li_enter asc
18. select * from librarian where li_salary <60000 or li_salary >100000
19. select li_name from librarian where li_deptcode in (2,5)
select li_name from librarian where li_deptcode =2 or li_deptcode =5
20. select dept_name from department where dept_city <>'Budapest'
21. select distinct dept_city from department
22. select * from librarian where li_salary between 50000 and 100000
23. select * from librarian where li_salary not between 50000 and 100000
24. select * from librarian where li_job in ('Storekeeper','Team Leader')
25. select * from librarian where li_job not in ('Storekeeper','Team Leader')
26. select * from borrowing where bo_indate is not null
27. select li_name, li_job, li_salary+li_premium*li_salary as salary from
librarian
28. select li_name from librarian where li_name ilike 'j%'
29. select li_name from librarian where li_name ilike '_A%'
30. select li_name from librarian where li_name ilike '%R'
31. select dept_name from department where dept_name ilike '%geo%'
32. select * from librarian where li_job in ('Storekeeper', 'Team Leader') and
li_salary between 60000 and 90000
33. select * from librarian where li_premium is null
34. select * from publications where p_title ilike '%atlasz%'
35. select * from catalogue where ca_price between 10000 and 30000 and
ca_indate<'2002-01-01'
36. select * from publications where p_author is null and p_year=2000

37. select Lower(p_title), Upper(p_publisher) from publications
38. select Concat('CUSTOMER: ', re_name), re_identifier, re_enter from borrower
where re_city='Eger'
39. select re_name, Char_length(re_name) from borrower
40. select re_name, Substring(re_name,3,5) from borrower
41. select re_name, Substring(re_name,5) from borrower
42. select re_name, Substring(re_name,1,5), Left(re_name, 5) from borrower
43. select re_name, Right(re_name, 5) from borrower
```

```
44. select re_name, StrPos(re_name,' ') from borrower
45. select Substring(re_name, 1, StrPos(re_name,' ')-1) as givenname,
Left(re_name, StrPos(re_name,' ')-1) as givenname, Substring(re_name,
StrPos(re_name,' ')+1) as familyname, Right(re_name, Char_Length(re_name)-
StrPos(re_name,' ')) as familyname  from borrower
46. select Lower(Substring(re_name, 1, StrPos(re_name,' ')-1)) as givenname,
Upper(Substring(re_name, StrPos(re_name,' ')+1)) as familyname from borrower
47. select p_title, Char_Length(p_title), Octet_Length(p_title) from publications
Octet_length gives the binary length (accent characters have duplicated the
length), the char_length give the real character length.
49. select 10/3 → 3 we got an integer
   select 10/3::integer
   select 10/3::decimal → we got the decimal number
   select 10%3  →1 this is the modulo operation returns the remainder or signed
remainder of a division.
50. select * from borrowing where (bo_indate - bo_outdate) <7
51. select * from borrowing where (bo_indate - bo_outdate) >7
52. select now()-'1989-01-13'
53. select extract(year from now())
54. select extract(day from (now()-'1989-01-13'))
55. select extract(dow from now())
56. select extract(dow from date '1989-01-13')
57. select Min(li_salary), Max(li_salary), Avg(li_salary), Sum(li_salary),
Count(*) from librarian
58. select Min(ca_price), Max(ca_price) from catalogue
59. select Count(*) from borrower where re_city='Budapest'
60. select Min(ca_indate), Max(ca_indate) from catalogue
61. select Count(distinct li_job) from librarian
62. select li_job, Min(li_salary), Max(li_salary), Avg(li_salary), Sum(li_salary),
Count(*) from librarian group by li_job
 or
select li_job, Min(li_salary), Max(li_salary), Avg(li_salary), Sum(li_salary),
Count(*) from librarian group by 1
63. select li_deptcode, Min(li_salary), Max(li_salary), Avg(li_salary),
Sum(li_salary), Count(*) from librarian group by 1 order by 1
64. select ca_pgcode, Count(*) from catalogue group by 1
65. select ca_deptcode, Count(*), Avg(ca_price) from catalogue group by 1
66. select Avg(bo_indate-bo_outdate) from borrowing where bo_outdate <'2004-05-30'
67. select bo_licode, Avg(bo_indate-bo_outdate) from borrowing where bo_pcode=1
group by 1
68. select li_job, Min(li_salary), Max(li_salary), Avg(li_salary), Sum(li_salary),
Count(*) from librarian group by li_job having Avg(li_salary)>80000
69. select li_deptcode from librarian group by 1 having count(*)=3
70. select Char_Length(li_name), Count(*), Avg(li_salary) from librarian group by
1 order by 3
71. select li_job, Max(li_salary)-Min(li_salary) from librarian group by 1
72. select li_job, Max(li_salary)-Min(li_salary) from librarian group by 1 order
by 2 desc limit 1

73. select li_name, li_job, dept_name, dept_city from librarian, department where
li_deptcode=dept_code order by 1
or
select li_name, li_job, dept_name, dept_city from librarian join department on
li_deptcode=dept_code order by 1
74. select li_name, li_job, dept_name, dept_city from librarian, department where
li_deptcode=dept_code and dept_city='Sopron'
or
select li_name, li_job, dept_name, dept_city from librarian join department on
li_deptcode=dept_code where dept_city='Sopron'
```

```
75. select p_title, p_author, p_publisher from publications, catalogue where
ca_pcode =p_code and ca_price between 3000 and 10000
or
select p_title, p_author, p_publisher from publications join catalogue on ca_pcode
=p_code where ca_price between 3000 and 10000
76. select re_name from borrower, borrowing where re_code=bo_recode and
bo_type='F'
or
select re_name from borrower join borrowing on re_code=bo_recode where bo_type='F'
77. select dept_name, Min(li_salary), Max(li_salary) from department, librarian
where li_deptcode=dept_code group by 1
or
select dept_name, Min(li_salary), Max(li_salary) from department join librarian on
li_deptcode=dept_code group by 1
78. select dept_city, Avg(li_salary), Count(*) from department, librarian where
li_deptcode=dept_code group by 1
or
select dept_city, Avg(li_salary), Count(*) from department join librarian on
li_deptcode=dept_code group by 1
79. select ca_inventory_number, ca_price, re_name, re_address, re_city,
re_postal_code, bo_indate from catalogue, borrowing, borrower where re_code=
bo_recode and bo_cacode=ca_code and bo_indate<'2004-05-01'
or
select ca_inventory_number, ca_price, re_name, re_address, re_city,
re_postal_code, bo_indate from catalogue join borrowing on bo_cacode=ca_code join
borrower on re_code= bo_recode where bo_indate<'2004-05-01'
80. select re_name, p_title, li_name, bo_code from publications, borrowing,
borrower, librarian where bo_pcode=p_code and re_code= bo_recode and
li_code=bo_licode
or
select re_name, p_title, li_name, bo_code from publications join borrowing on
bo_pcode=p_code join borrower on re_code= bo_recode join librarian on
li_code=bo_licode
81. select p_title, ca_inventory_number, re_name, bo_indate, bo_outdate from
borrower, borrowing, catalogue, publications where re_code=bo_recode and
bo_cacode=ca_code and bo_pcode=p_code
or
select p_title, ca_inventory_number, re_name, bo_indate, bo_outdate from borrower
join borrowing on re_code=bo_recode join catalogue on bo_cacode=ca_code join
publications on bo_pcode=p_code
82. select re_name, p_title, bo_indate, bo_outdate from borrower, borrowing,
publications where bo_recode=re_code and bo_pcode=p_code and p_title='KISATLASZ'
or
select re_name, p_title, bo_indate, bo_outdate from borrower join borrowing on
bo_recode=re_code join publications on bo_pcode=p_code where   p_title='KISATLASZ'
83. select re_name, bo_indate, bo_outdate from borrower join borrowing on
bo_recode=re_code join librarian on bo_licode=li_code where li_name='Oliver Cross'
or
select re_name, bo_indate, bo_outdate from borrower, borrowing, librarian where
bo_recode=re_code and bo_licode=li_code and li_name='Oliver Cross'
84. select * from publication_group left join publications on pg_code =p_pgcode
or
select * from publications right join publication_group on pg_code =p_pgcode
85. select * from librarian where li_salary>(select li_salary from librarian where
li_name='Alison Porter')
86. select * from borrower where re_city=(select re_city from borrower where
re_name='Aaron Thorsen')
```

```
87. select * from publications, catalogue where ca_pcode=p_code and
ca_price>(select ca_price from publications, catalogue where ca_pcode=p_code and
p_title='FÖLDRAJZI VILÁGATLASZ')
88. select * from librarian where li_job=(select li_job from librarian where
li_name='Samantha Carter')
89. select * from publications where p_year <(select p_year from publications
where p_title='Cartography Thematic Map Design')
90. select * from librarian where li_job in (select li_job from librarian,
department where li_deptcode=dept_code and dept_name='Department of Cartography')
91. select * from department where dept_city in (select dept_city from department
where dept_name in ('Department of Geodesy', 'Department of Geophysics'))
92. select publications.* from publications, catalogue where ca_pcode=p_code and
ca_price>(select Avg(ca_price) from catalogue)
93. select * from borrowing where (bo_indate-bo_outdate)=(select Max(bo_indate-
bo_outdate) from borrowing )
94. select * from borrowing where (bo_indate-bo_outdate)=(select Min(bo_indate-
bo_outdate) from borrowing )
95. select * from publications where Char_Length( p_title) > (select
Max(Char_Length(p_title))/2 from publications)
96. select * from librarian where li_salary >(select Avg(li_salary) from
librarian, department where li_deptcode=dept_code and dept_city='Budapest')
97. select * from librarian, department where li_deptcode=dept_code and
dept_city='Budapest' and li_salary >(select Avg(li_salary) from librarian)
```

Chapter 5: Comparision sheet to PostgreSQL and MySQL

If you have previously learned MySQL, you may notice some differences when working with PostgreSQL. The following table compares the two SQL dialects and highlights their key differences.

Function, operator, clause	MySQL Example	PostgreSQL Example
ORDER BY two fields	`select * from publications order by p_year desc, p_publisher;`	`select * from publications order by p_year desc, p_publisher`
WHERE with one condition	`select * from catalogue where ca_price >2000;`	`select * from catalogue where ca_price >2000;`
Text type data in SQL	`select * from librarian where li_job='Team Leader'` `select * from librarian where li_job="Team Leader"`	`select * from librarian where li_job='Team Leader'` **Note: Use single quotes. Case sensitivity depends on collation.**

Date type data in SQL	`select * from librarian where li_enter>='1992-01-03'`
	`select * from librarian where li_enter>='1992.01.03'`
	`select * from librarian where li_enter>="1992-01-03"`
	`select * from librarian where li_enter>="1992.01.03"`
		`select * from librarian where li_enter>='1992-01-03'`
		`select * from librarian where li_enter>='1992.01.03'`
		Note:
		Use ISO format 'YYYY-MM-DD'.

IN	`select * from librarian where li_job in ('Storekeeper', 'Team Leader')`	`select * from librarian where li_job in ('Storekeeper', 'Team Leader')`
NOT IN	`select * from librarian where li_job not in ('Storekeeper', 'Team Leader')`	`select * from librarian where li_job not in ('Storekeeper', 'Team Leader')`
BETWEEN	`select * from publications where p_year between 2001 and 2005`	`select * from publications where p_year between 2001 and 2005`
NOT BETWEEN	`select * from publications where p_yeaar not between 2001 and 2005`	`select * from publications where p_yeaar not between 2001 and 2005`
LIKE Case Sensitive!	`select * from publications where p_title like '%ATLASZ%'`	`select * from publications where p_title like '%ATLASZ%'` **Case Sensitive,see the explanation above!**
NOT LIKE Case Sensitive!	`select * from publications where p_title not like '%ATLASZ%'`	`select * from publications where p_title not like '%ATLASZ%'` **Case Sensitive,see the explanation above!**
ILIKE case-insensitive!	PostgreSQL only	`select * from publications where p_title ilike '%atlas%'` **Use ILIKE / NOT ILIKE in PostgreSQL for case-insensitive search.**
NOT ILIKE case-insensitive!	PostgreSQL only	`select * from publications where p_title not ilike '%atlas%'` **Use ILIKE / NOT ILIKE in PostgreSQL for case-insensitive search.**
IS NULL	`select * from publications where p_isbn is null`	`select * from publications where p_isbn is null`

IS NOT NULL	`select * from publications where p_isbn is not null`	`select * from publications where p_isbn is not null`
AND	`select * from publications where p_isbn is not null and p_publisher='ESRI PRESS'`	`select * from publications where p_isbn is not null and p_publisher='ESRI PRESS'`
OR	`select * from publications where p_isbn is not null or p_publisher='ESRI PRESS'`	`select * from publications where p_isbn is not null or p_publisher='ESRI PRESS'`
DISTINCT	`select distinct li_job from librarian`	`select distinct li_job from librarian`
UPPER()	`select Upper(li_name) from librarian`	`select Upper(li_name) from librarian`
LOWER()	`select Lower(li_name) from librarian`	`select Lower (li_name) from librarian`
CONCAT()	`select Concat('ISBN number: ', p_isbn) from publications`	`select Concat('ISBN number: ', p_isbn) from publications`
COALESCE()	`select Coalesce(p_isbn,'****') from publications`	`select Coalesce(p_isbn,'****') from publications`
SUBSTRING()	`select Substring(li_name, 2,3) from librarian`	`select Substring(li_name, 2,3) from librarian`
SUBSTRING()	`select Substring(li_name, 2) from librarian`	`select Substring(li_name, 2) from librarian`
SUBSTRING()	`select Substring(li_name, -2,1) from librarian`	- Negative indexes not supported in PostgreSQL, do it with RIGHT()
SUBSTRING()	`select Substring(li_name, -2) from librarian`	- Negative indexes not supported in PostgreSQL, do it with RIGHT()
LEFT()	`select Left(li_name, 3) from librarian`	`select Left(li_name, 3) from librarian`
RIGHT()	`select Right(li_name, 3) from librarian`	`select Right(li_name, 3) from librarian`
INSTR() STRPOS()	`select Instr(li_name,' ') from librarian`	`select StrPos(li_name,' ') from librarian`

LENGTH()	`select li_name, Length(li_name) from librarian`
	`/binary length: accent character has duplicated length/`
		`select li_name, Length(li_name) from librarian`
		`Number of characters`
		`Octet_Length(string) is the same as the Length() in MySQL: for byte length`

CHAR_LENGTH()	`select li_name, Char_Length(li_name) from librarian` `real number of characters`	`select li_name, Char_Length(li_name) from librarian` `real number of characters`
DIV	`select li_salary, li_salary div 5000 from librarian`	`select li_salary, Div(li_salary,5000) from librarian` `Use function in PostgreSQL`

MOD(), %	`select li_salary, Mod (li_salary,5000) from librarian`
	`or`
	`select li_salary, li_salary mod 5000 from librarian`
	`or`
	`select li_salary, li_salary %5000 from librarian`
		`select li_salary, Mod (li_salary,5000) from librarian`
		`or`
		`select li_salary, li_salary %5000 from librarian`

ABS()	`select Abs(-2.1)`	`select Abs(-2.1)`
SIGN()	`select Sign(-2.1)`	`select Sign(-2.1)`

POWER() vagy POW()	select Power(2,4)	select Power(2,4)
ROUND()	select Round(2.1234, 0) select Round(200.1234, -2) select Round(200.1234, 2)	select Round(2.1234) select Round(2.1234, 0) select Round(200.1234, -2) select Round(200.1234, 2)
TRUNCATE()	select Truncate(200.1234, 0) select Truncate(200.1234, 2) select Truncate(200.1234, -2)	select Trunc(200.1234) select Trunc(200.1234, 0) select Trunc(200.1234, 2) select Trunc(200.1234, -2) PostgreSQL uses Trunc()
CEIL()	select Ceil(-2.1234)	select Ceil(-2.1234)
FLOOR()	select Floor(-2.1234)	select Floor(-2.1234)
SQRT()	select Sqrt(200)	select Sqrt(200)
DATEDIFF()	select * from borrowing where Datediff(bo_outdate,bo_indate)>7	- use subtraction in PostgreSQL: select bo_indate-bo_outdate from borrowing
NOW()	select Now()	select now() →with timezone
YEAR()	select Year(Now())	select Extract(year from Now())
MONTH()	select Month(Now())	select Extract(month from Now())
DAY()	select Day(Now())	select Extract(day from Now())
HOUR()	select Hour(Now())	select Extract(hour from Now())
MINUTE()	select Minute(Now())	select Extract(minute from Now())
SECOND()	select Second(Now())	select Extract(second from Now())
DAYOFWEEK()	select Dayofweek(now())	select Extract(dow from Now())
timezone		select Extract(timezone_hour from Now())
MIN()	select Min(li_salary) from librarian	select Min(li_salary) from librarian
MAX()	select Max(li_salary) from librarian	select Max(li_salary) from librarian
AVG()	select Avg(li_salary) from librarian	select Avg(li_salary) from librarian
SUM()	select Sum(li_salary) from librarian	select Sum(li_salary) from librarian
COUNT()	select Count(*) from librarian	select Count(*) from librarian
Group by	select li_job, Count(*) from librarian group by li_job or select li_job, Count(*) from librarian group by 1	select li_job, Count(*) from librarian group by li_job or select li_job, Count(*) from librarian group by 1
HAVING	select li_job, Count(*) from librarian group by li_job having Count(*)>6	select li_job, Count(*) from librarian group by li_job having Count(*)>6
INNER JOIN	select dept_name, li_name from librarian, department where dept_code=li_deptcode OR select dept_name, li_name from librarian join department ON dept_code=li_deptcode OR	select dept_name, li_name from librarian, department where dept_code=li_deptcode OR select dept_name, li_name from librarian join department ON dept_code=li_deptcode OR

	`select dept_name, li_name from librarian inner join department ON dept_code=li_deptcode`	`select dept_name, li_name from librarian inner join department ON dept_code=li_deptcode`
LEFT or RIGHT JOIN	`select dept_name, li_name from librarian right join department ON dept_code=li_deptcode`	`select dept_name, li_name from librarian right join department ON dept_code=li_deptcode`
Subqueries with relations	`select li_name, li_salary from librarian where li_salary >(select li_salary from librarian where li_name='Daniel Jackson')`	`select li_name, li_salary from librarian where li_salary >(select li_salary from librarian where li_name='Daniel Jackson')`
Subqueries with in	`SELECT * FROM department WHERE dept_city IN (select dept_city FROM department WHERE dept_name IN ('Department of Geography','Department of Geophysics'));`	`SELECT * FROM department WHERE dept_city IN (select dept_city FROM department WHERE dept_name IN ('Department of Geography','Department of Geophysics'));`
EXISTS, NOT EXISTS	`SELECT * FROM department where NOT EXISTS (SELECT 1 FROM librarian WHERE dept_code=li_deptcode);`	`SELECT * FROM department where NOT EXISTS (SELECT 1 FROM librarian WHERE dept_code=li_deptcode);`
LIMIT X	`SELECT * FROM librarian limit 5`	`SELECT * FROM librarian limit 5`
LIMIT X, Y **LIMIT X OFFSET Y**	`SELECT * FROM librarian limit 5,3` OR `SELECT * FROM librarian limit 5 offset 3` Write 3 results, begin from the record 6. (number of offset+1)	`SELECT * FROM librarian limit 3 offset 5` Write 3 results, begin from the record 6. (number of offset+1)

Chapter 6. The PostGIS functions

This chapter based on the offical PostGIS documentation. In the following sections, you'll find explanations and examples of key functions. The full documentation is available here:

https:/postgis.net/docs/manual-3.3/

Getting familiar with PostGIS

PostGIS is a PostgreSQL extension or handling spatial data. Here's a basic example of creating a point using the `ST_Point()` function:

```
select ST_Point(1, 2) as MyFirstPoint;
```

To create a point with a coordinate system (SRID):

```
select ST_SetSRID(ST_Point(-77.036548, 38.895108),4326);
```

Or using WKT (Well-Known Text) format with `ST_GeomFromText()`:

```
select ST_GeomFromText('POINT(-77.036548 38.895108)', 4326);
```

What is WKT format?

WKT (Well-known text format) is an OGC Standard for representing geometries as text. Functions like `ST_GeomFromText()` and `ST_AsText()` convert between WKT and geometry. Coordinates follow the (longitude, latitude) order.

What is WKB format?

WKB (Well-known binary format) is the binary representation of geometries, also standardized by OGC. Though not human-readable, it's more efficient for storage. To view WKB in readable formats, use `ST_AsEWKT() function` or `ST_AsText`.

```
select ST_AsEWKT('0101000020E6100000FD2E6CCD564253C0A93121E692724340');
select ST_AsText('0101000020E6100000FD2E6CCD564253C0A93121E692724340');
```

Representing Points, Lines and Polygons in WKT

The tables below shows single geometries (geometry primitives) and the multipart geometry representations of features.

The following formats are used:

- **Point:** POINT(x y) — a simple coordinate pair inside parentheses.
- **LineString:** LINESTRING(x1 y1, x2 y2, ...) — a sequence of points separated by commas.
- **Polygon:** POLYGON((outer ring), (inner ring1), (inner ring2), ...) — the first set of coordinates defines the outer ring; any additional sets define inner rings (holes). The rings must be closed (first and last points are the same).

Geometry primitives (2D)

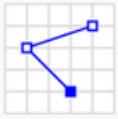
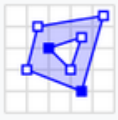
Type	Examples	
Point		`POINT (30 10)`
LineString		`LINESTRING (30 10, 10 30, 40 40)`
Polygon		`POLYGON ((30 10, 40 40, 20 40, 10 20, 30 10))`
		`POLYGON ((35 10, 45 45, 15 40, 10 20, 35 10), (20 30, 35 35, 30 20, 20 30))`

Image source: Wikipedia.

Multi geometries are group of geometry primitives. When do you use them? For example, the United Kingdom consists of several islands: Great Britain, Northern Island, the Hebrides, etc. If these polygons are multipolygons, when you click on one of these features, all parts of the multipolygon will be selected (try it out yourself in QGIS!)

Geometry collections are a special set of features. They can contain points, lines and polygons (as well as multipart geometries). However, their usage is not recommended because if you want to use your data in a geoinformatics software, they won't be supported. GIS software generally does not support this type of geometry.

Multipart geometries (2D)

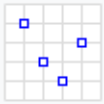
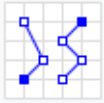
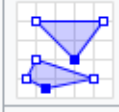
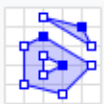
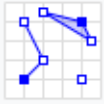
Type	Examples	
MultiPoint		`MULTIPOINT ((10 40), (40 30), (20 20), (30 10))`
		`MULTIPOINT (10 40, 40 30, 20 20, 30 10)`
MultiLineString		`MULTILINESTRING ((10 10, 20 20, 10 40), (40 40, 30 30, 40 20, 30 10))`
MultiPolygon		`MULTIPOLYGON (((30 20, 45 40, 10 40, 30 20)), ((15 5, 40 10, 10 20, 5 10, 15 5)))`
		`MULTIPOLYGON (((40 40, 20 45, 45 30, 40 40)), ((20 35, 10 30, 10 10, 30 5, 45 20, 20 35), (30 20, 20 15, 20 25, 30 20)))`
GeometryCollection		`GEOMETRYCOLLECTION (POINT (40 10), LINESTRING (10 10, 20 20, 10 40), POLYGON ((40 40, 20 45, 45 30, 40 40)))`

Image source: Wikipedia.

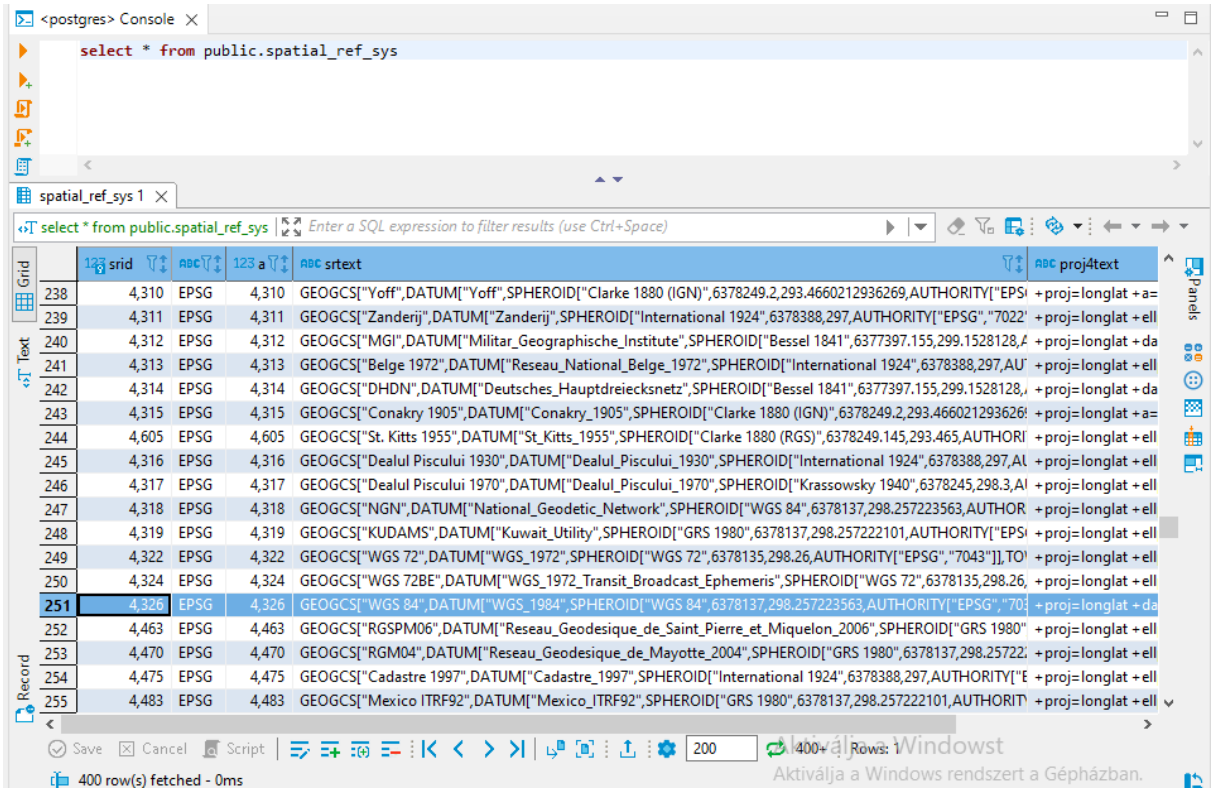
What is EWKT/EWKB?

EWKT (Extended Well-know text) and EWKB (Extended Well-know binary format) are PostGIS specific file formats that store the spatial reference system (SRID). For example:

```
'SRID=4326;POINT(19.1 47.5)'
```

What is SRID?

SRID stands for Spatial Reference Identifier. You can easily access SRIDs and EPSG codes on this website: http://epsg.io/. When you add the PostGIS extension to a database, PostGIS automatically generates a spatial_ref_sys table. Do not delete this table, as it contains the SRIDs.



How many dimensions does the coordinate have?

Coordinate dimensions refer to how many axes the coordinate system has. In most cases, coordinates have two (X, Y) or three (X, Y, Z, or X, Y, M). PostGIS also supports a fourth dimension, the M coordinate, which can be stored for every node.

What can ”M” represent?

-Temporal changes: For example, a bus moving from its starting point to its current stop.

- Special IDs: For instance, an emergency phone number next to a motorway.

Storing special geometries in PostGIS

PostGIS does not support popular Bezier curves or other splines, but these can be approximated using polylines/linestrings. However, PostGIS supports several special curve types:

CircularString: This is a basic curve type, similar to a LINESTRING. A single segment requires three points: the start and end points (first and third) and any other point on the arc. In the case of full circle the first and the last points are the same.

CompoundCurve: it is a combination of CircularString and Linestrings.

```
COMPOUNDCURVE(CIRCULARSTRING(0 0, 1 1, 1 0),(1 0, 0 1))
```

CurvePolygon: Similar to simple polygons, but the outer and inner rings can be CircularStrings or CompoundCurves or LineStrings.

MultiCurve: A collection of curves, which can be CircularStrings, CompoundCurves or LineStrings.

MultiSurfaceA special type of surface that can consist of Polygons, CurvePolygons.

PolyhedralSurface, Triangle, Tin: Additional possibilities to store geometries. See more here: https://postgis.net/docs/using_postgis_dbmanagement.html#PolyhedralSurface

Why do every functions start with ST?

The "ST" prefix stands for Spatial and Temporal. These are the new versions of PostGIS functions.

How to specify geometries and spatial reference identifier (projection)?

`ST_Point(X,Y)` → X: longitude, Y: latitude, point with unknown or default SRID

`ST_Point(X,Y,SRID)` → X: longitude, Y: latitude, with specified SRID.

You can also add the Z or M coordinates:

`ST_PointM(X,Y,M)` or `ST_PointZ(X,Y,Z)` or `ST_PointZM(X,Y,Z,M)`

`ST_PointM(X,Y,M, SRID)` or `ST_PointZ(X,Y,Z,SRID)` or `ST_PointZM(X,Y,Z,M,SRID)`

Other Functions for Geometry Creation:

`ST_MakePoint(X,Y)` →Creates a point with two dimensions

`ST_MakePoint(X,Y, Z)` → Creates a point with three dimensions

`ST_MakePoint(X,Y, Z, M)` → Creates a point with three dimensions

`ST_MakeLine(Point, Point)` → Creates lines from points

`ST_MakePolygon(LineString)` → Convert a linestring to polygon. (Be sure the first and last points are the same.)

Geometry type, Dimension, Coordinate dimension and Number of Nodes

What is the type of the geometry?

`ST_GeometryType(geom)` → `ST_Multipolygon`. Returns the type of the geometry with the ST_ prefix

`GeometryType(geom)` → Returns the type **without the ST_ prefix** (e.g., MULTIPOLYGON).

```
select ST_GeometryType(s_geom), GeometryType(s_geom) from settlement
```

`ST_NumGeometries(geom)` → Returns **the number of geometries** in a geometry collection or multi-geometry.

What are the dimensions?

`ST_Dimension(geom)` → Returns the topological the dimension of geometry. In case of points: 0, lines: 1, polygons: 2.

`ST_CoordDim(geom)` → Returns the **coordinate dimension**: 2 if only X and Y are present, 3 or 4 if Z and/or M values are included.

```
select ST_Dimension(sc_geom), ST_CoordDim(sc_geom) from settlement_centroid
```

Number of Nodes

You can count the number of nodes in a geometry using the ST_NPoints or ST_NumPoints functions:

`ST_NPoints(geom)` → Returns the total number of nodes in a geometry, including in multi-geometries and polygons.

`ST_NumPoints(geom)` → Returns the number of points in a LineString. For other geometry types, it returns NULL.

```
SELECT ST_NPoints(s_geom) As npoints, ST_NumPoints(s_geom) As numpoints from
settlement
```

Advanced in PostGIS

Projections, Spatial Reference Identifiers

What is the SRID of the geometry?

SRID stands for Spatial Reference Identifier, which defines the projection used. PostGIS supports many projections and uses the EPSG identification system. The SRID is equivalent to the EPSG number. You can find all EPSG codes at https://epsg.io/.

In this course, you will use:

EPSG 23700: Hungarian EOV projection with HD72 datum.

EPSG 3857: Web Mercator projection (used in Google Maps, OpenStreetMap etc.)

EPSG 4326: WGS 84 - WGS84 - World Geodetic System 1984, used in GPS

To check the SRID of geometries:

`ST_SRID(geom)` → returns with the SRID of each feature. In the hungary database, all features within the same table generally have the same SRID.

```
select ST_Srid(sc_geom) from settlement_centroid
```

To update the SRID of a geometry column:

`UpdateGeometrySrid('table name', 'geom ', 'EPSG number')` → this function updates the SRID in a table.

```
select UpdateGeometrySRID('settlement','s_geom','23700')
```

Projection transformation

`ST_Transform(geom, EPSG number)` → transforms a geometry into a different projection

```
select s_name, ST_Transform(s_geom, 4326) from settlement
```

Validity of geometry

`ST_isValid(geom)` → check if geometries are valid (e.g., no self-intersections): Returns true or false.

```
select ST_IsValid(s_geom) from settlement
```

Measurements

Before measuring distances, lengths, or areas, consider:

- **Dimensionality (2D or 3D):** How many dimensions do the coordinates have (2D - plain or 3D – spatial coordinates)
- Do you use **planar or 3D spatial measurement functions**?
- Which **coordinate system** do you use?
- Whether measurements are planar (flat surface) or geodetic (curved surface, i.e., globe). In case of hundred of km distance, it can be a significant difference.

Always understand the nature of your dataset before working with measurements!

Measurements on a plane (2D)

Units depend on the dataset's coordinate system. If the feature coordinates are in meter, you get the result in meters or square meters. If coordinates are in degrees, you will get back the size in degrees.

Line length (2D)

`ST_Length(geom)` → length of a line (`ST_Length2D function` is the same)

```
select cr_name, ST_Length(cr_geom) from creek
```

Line length (3D)

`ST_Length3D(geom)` → 3D length, includes the Z coordinate

Perimeter of a polygon

`ST_Perimeter(geom)` → the length of a polygon outline. If the polygon contains hole(s), it's length will be also inlcuded in the result (`ST_Perimeter2D` is the same).

```
select d_name, ST_Perimeter(d_geom) from district
```

Area of a polygon

`ST_Area(geom)` → area of the polygon

```
select d_name, ST_Area(d_geom) from district
```

Measurements using geographic coordinates

Length on a spheroid

`ST_LengthSpheroid(geom, shperoid data)` → returns with the length in meter, while the input coordinates are in degrees. Calculates the length or perimeter of a geometry on an ellipsoid.

```
select ST_LengthSpheroid(ST_GeomFromText('LINESTRING(19.1 47.5, 18.5 47.2)') ,
'SPHEROID["WGS_84",6378137,298.257223563]')
```

Get Coordinates from Points

If you have point data, it is possible to separate the longitude and latitude into two columns. Use

`ST_X(geom)` → returns with X (longitude)

`ST_Y(geom)` → returns with Y (latitude)

Other measurement functions

Distance between geometries

`ST_Distance(geom A, geom B)` → distance of two geometries in meter/degrees etc./depends on the initial units/)

```
select ST_Distance(ST_GeomFromText('LINESTRING(628200 215000, 629000 216000)'),
ST_GeomFromText('POINT(632500 215500)'))
```

3D Distance

`ST_3DDistance(geom A, geom B)` → returns the 3-dimensional minimum Cartesian distance between two geometries in projected units (spatial ref units).

Maximum distance between geometries

`ST_MaxDistance(geom A, geom B)` → returns with the maximal distance between two geometries

```
select ST_MaxDistance(ST_GeomFromText('LINESTRING(628200 215000, 629000 216000)'),
ST_GeomFromText('POINT(632500 215500)'))
```

Distance on a sphere (degrees)

`ST_DistanceSphere(point A in degree, point B in degree)` → Returns linear distance in meters between two lon/lat points. The radius of this sphere: 6 371 008 m

```
select ST_DistanceSphere(ST_GeomFromText('POINT(19.1 47.5)'),
ST_GeomFromText('POINT(18.5 47.2)'))
```

Distance on a spheroid

`ST_DistanceSpheroid(point A in degree, point B in degree, spheroid description)` → Returns linear distance between two lon/lat points given a particular spheroid. Currently only implemented for points.

```
select ST_DistanceSpheroid(ST_GeomFromText('POINT(19.1 47.5)'),
ST_GeomFromText('POINT(18.5 47.2)'),'SPHEROID["WGS 84",6378137,298.257223563]')
```

Shortest & Longest Lines Between Geometries

Shortest line

`ST_ShortestLine(geom A, geom B)` → returns with the shortest line between two geometries

Draw the shortest line between Somogy and Nógrád counties!

```
select ST_ShortestLine((select co_geom from county where co_id=1), (select co_geom
from county  where co_id=17))
```

3D Shortest line

`ST_3DShortestLine(geom A, geom B)`→ Returns the shortest line between two geometries, including elevation (Z coordinate)

`ST_LongestLine(geom A, geom B)` → Returns the longest line between two geometries.

Task:

Draw the longest line between Somogy and Nógrád counties!

Longest line

```
select ST_LongestLine((select co_geom from county where co_id=1),(select co_geom
from county  where co_id=17))
```

`ST_3DLongestLine(geom A, geom B)`→ Returns the longest line between two geometries, including elevation (Z coordinate)

Closest Point

`ST_ClosestPoint(geom A, geom B)` → returns the coordinates of the closest point of the geometry B to geometry A. This point corresponds to the start of the shortest line.

```
select ST_ClosestPoint((select co_geom from county where co_id=1),(select co_geom
from county  where co_id=17))
```

`ST_3DClosestPoint(geom A, geom B)`→ Returns the closest point of geometry B to geometry A, including elevation (Z coordinate).

Azimuth

`ST_Azimuth(point A, point B)` → Returns the azimuth of the line defined by the two points. Azimuth: north= 0; northeast= π/4; east= π/2; southeast = 3π/4; south = π; southwest 5π/4; west= 3π/2; northwest = 7π/4.

```
select degrees(ST_azimuth(ST_GeomFromText('POINT(628200 215000)'),
ST_GeomFromText('POINT(632500 215500)')))
```

Angles

`degrees(radian)`→ Converts the radians to degrees.

`ST_Angle` has three forms:

- `ST_Angle(line A, line B)` → Angle between two lines.
- `ST_Angle(point A, point B, point C)` → Angle at point B formed by three points.
- `ST_Angle(point A, point B, point C, point D)` → Angle between segment AB and segment CD.
Result is given in radians.

Angle between two lines:

```
select degrees(ST_Angle(ST_GeomFromText('LINESTRING(628200 215000,632500
215500)'), ST_GeomFromText('LINESTRING(629500 212500,628200 212500)')))
```

Angle between 3 points, at point B

```
select degrees(ST_Angle(ST_GeomFromText('POINT(628200 215000)'),
ST_GeomFromText('POINT(632500 215500)'), ST_GeomFromText('POINT(629500 212500)')))
```

Angle between line AB, and CD.

```
select degrees(ST_Angle(ST_GeomFromText('POINT(628200 215000)'),
ST_GeomFromText('POINT(632500 215500)'), ST_GeomFromText('POINT(629500 212500)'),
ST_GeomFromText('POINT(628200 212500)')))
```

Distance Metrics

Hausdorff-distance → Returns the Hausdorff distance, a measure of similarity between two geometries.

`ST_HausdorffDistance(geom A, geom B)`

Fréchet distance → Implements algorithm for computing the Fréchet distance restricted to discrete points for both geometries, based on Computing Discrete Fréchet Distance. The Fréchet distance is a measure of similarity between **curves** that takes into account the location and ordering of the points along the curves. Therefore it is often better than the Hausdorff distance.
`ST_FrechetDistance(geom A, geom B, float densifyFrac = -1)`

Bounding boxes and bounding geometries

Envelope

`ST_Envelope(geom)` → returns the coordinates of bounding box as a polygon (5 points, first and the last identical).

`Box2D(geom)` → returns only the lower-left and the upper-right corner points of the bounding box

```
select ST_Envelope(s_geom), Box2D(s_geom) from settlement where s_name ='Sopron'
```

Envelope result

```
POLYGON ((452818.6793498857 256385.44666137366, 452818.6793498857
271262.2491004063, 477455.0613929047 271262.2491004063, 477455.0613929047
256385.44666137366, 452818.6793498857 256385.44666137366))
```

and Box2D result

```
BOX(452818.6793498857 256385.44666137366,477455.0613929047 271262.2491004063)
```

Note: there is also a BOX3D function.

Hulls

`ST_ConvexHull(geom)` → returns the smallest convex polygon that encloses a geometry.

```
select ST_ConvexHull(s_geom) from settlement where s_name ='Sopron'
```

`ST_ConcaveHull(geom, target_percent, allow_holes)` → returns with a concave hull. The target percent indicates the grade between the original object and the convex hull. If target percent is 0, it is the original object, if it is 1, this is the convex hull. Allow holes parameter indicates whether holes are allowed.

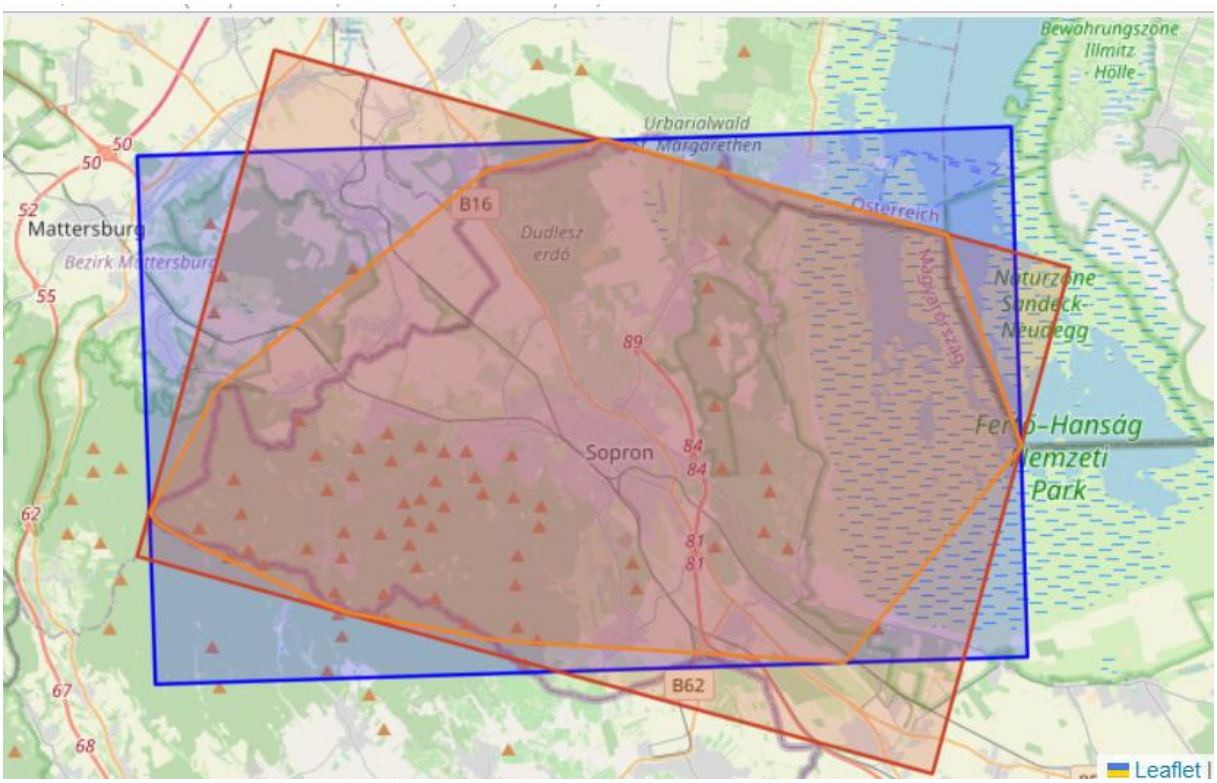
```
select ST_ConcaveHull(s_geom,0.5) from settlement where s_name ='Sopron'
```

`ST_OrientedEnvelope(geom)` →Returns with the smallest area bounding box. It can be rotated, and in the most cases, it is not identical with the Envelope.

```
select ST_OrientedEnvelope(s_geom) from settlement where s_name ='Sopron'
```

The image below shows these geometries. Blue is the envelope, red is the oriented envelope, and the orange is the convex hull.

```
select ST_Envelope(s_geom), ST_OrientedEnvelope(s_geom), ST_Convexhull(S_geom)
from settlement where s_name ='Sopron'
```



Accessing other geometric information

Boundary

`ST_Boundary(geom)` → Returns the outline of a geometry as a linestring. If the polygon element has a hole/inner ring, the result will be a MultiLineString. If the original element is a polyline, the result is a MultiPoint feature. The points are the first and the last point of the line.

```
select ST_Boundary(s_geom) from settlement where s_name ='Sopron'
```

```
SELECT ST_Boundary(geom)FROM (SELECT 'LINESTRING( 10 130, 50 190, 110 190, 140
150, 150 80, 100 10, 20 40 )'::geometry As geom) As f;
→MULTIPOINT ((10 130), (20 40))
```

Start and end point

`ST_StartPoint(geom)` and `ST_EndPoint (geom)` → returns the first or last point of a LineString.

```
SELECT ST_StartPoint(geom)FROM (SELECT 'LINESTRING( 10 130, 50 190, 110 190, 140
150, 150 80, 100 10, 20 40 )'::geometry As geom) As f;
→POINT (10 130)
```

```
SELECT ST_EndPoint(ST_Boundary(geom)) FROM (SELECT 'POLYGON (( 10 130, 50 190, 110
190, 140 150, 150 80, 100 10, 20 40, 10 130 ), ( 70 40, 100 50, 120 80, 80 110, 50
90, 70 40 ))'::geometry As geom) As f; → returns NULL for polygons (no end-point)
```

Exterior ring

`ST_ExteriorRing(geom)` → returns the outer ring/outline of the polygon as a LineString. You get NULL, if the initial geometry is not polygon, or it is a MultiPolygon.

```
SELECT ST_ExteriorRing(geom) FROM (SELECT 'Polygon(( 10 130, 50 190, 110 190, 140
150, 150 80, 100 10, 20 40, 10 130 ))'::geometry As geom) As f;
→ LINESTRING (10 130, 50 190, 110 190, 140 150, 150 80, 100 10, 20 40, 10 130)
```

Geometry validation

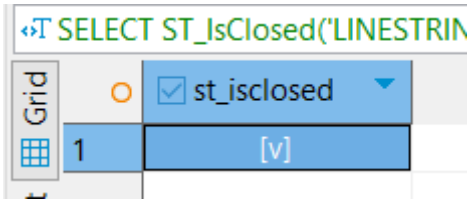
`ST_IsRing(geom)` → Returns true if the LineString is closed and simple (no self-intersection).

`ST_IsSimple(geom)` → Returns true if the geometry has no self-intersections or invalid points.

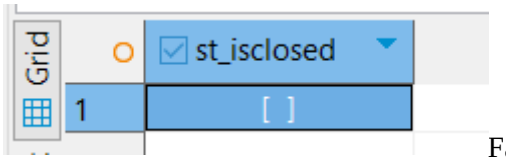
`ST_IsClosed(geom)` → Checks if the first and last points of a LineString are the same. (It also works with CircularString, Curves and PolyhedralSurface).

```
SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1,0 0)');
→True
SELECT ST_IsClosed('LINESTRING(0 0, 0 1, 1 1)');
→False
```

`ST_IsPolygonCW(geom)` or `ST_IsPolygonCCW(geom)` →Check if a polygon's exterior ring is clockwise (CW) or counter-clockwise (CCW).



True



False

Creating geometries

`ST_Buffer(geom, radius of the buffer, other options)` → Creates a buffer around the geometry with the given radius (units are the same as the projection's unit). If the radius is negative, then the result is an inner (shrinking) buffer. The third parameter is optional:

quad_segs = the number of line segments, used to approximate curves. Default is 8.

endcap = round | flat | square → Defines the style of line endings. round (default), flat (ends at last vertex), square (extends beyond line)

join = round | mitre | bevel → Defines how corners are joined. Default is round.

mitre_limit→ Sets the mitre ratio limit.

side = both | left | right → Creates a one- or two-sided buffer.

```
select s_name, ST_Buffer(s_geom, 1000) from settlement where s_name='Sopron'
```

```
select cr_name, ST_Buffer(cr_geom, 500,'endcap=flat') from creek where
cr_name='Benta-patak'
```

Centroid

`ST_Centroid(geom)` → returns the centroid (geometric center) of the geometry.

```
select s_name, ST_Centroid(s_geom) from settlement
```

Delaunay Triangles

`ST_DelaunayTriangles(geom, tolerance)` → returns a triangulated subdivision of the geometry.

```
select ST_DelaunayTriangles(s_geom) from settlement where s_name='Sopron'
```

Line Merge

`ST_LineMerge(geom)` → Merges a collection of lines into a single continuous LineString or MultiLineString.

```
select ST_LineMerge('MULTILINESTRING((10 160, 60 120), (120 140, 60 120), (120
140, 180 120))');
```

Line Simplification and Smoothing

`ST_Simplify(geom, tolerance)` → Line or polygon simplification with Douglas–Peucker algorithm.

```
select s_name, ST_Simplify(s_geom, 500) from settlement where s_name='Sopron'
```

`ST_SimplifyVW(geom, tolerance)` → Line or polygon simplification Visvalingam-Whyatt algorithm.

```
select s_name, ST_SimplifyVW(s_geom, 500) from settlement where s_name='Sopron'
```

`ST_ChaikinSmoothing(geom, number of iterations, preserve_endPoints)` → Applies Chaikin's smoothing algorithm to the geometry. Maximum 5 iterations. The last parameter defines whether to preserve the original endpoints.

```
select cr_name, ST_ChaikinSmoothing(cr_geom, 3, true) from creek where
cr_name='Benta-patak'
```

Understanding spatial relations – Geoprocessing

These functions test spatial relationships and return a boolean value (true/false).

Please study the following website to understand these functions:

http://postgis.net/workshops/postgis-intro/spatial_relationships.html

`ST_Intersects(geom A, geom B)` → Returns true if geometries A and B share any space. If geom A and geom B have a common part in plain or 3D, they intersect. For example: there are two points in same coordinate; a polygon touches a line, or a polygon overlaps with another one, or it overlaps with a point.

`ST_Disjoint(geom A, geom B)` → Returns true if geometries A and B do not intersect. The reverse of intersects. In this case, the two geometries do not intersect.

`ST_Equals(geom A, geom B)` → Returns true if A and B are spatially equal (identical coordinates and structure). If they are perfectly same, it it true (point-point, line-line, polygon-polygon etc).

`ST_Crosses(geom A, geom B)` → Returns true if A crosses B (e.g., linestring-line, point-line, etc.). Possible crossings can be between: multipoint/polygon, multipoint/linestring, linestring/linestring, linestring/polygon and linestring/multipolygon

`ST_Overlaps(geom A, geom B)` → Returns true if A and B overlap and are of the same dimension. Point-Point, line-line and polygon-polygon.

`ST_Touches(geom A geom B)` → Returns true if A and B touch at the boundary but do not overlap. For example: point is on the line or on outline of polygon; lines or polygons touch each other on outline.

`ST_Contains(geom A, geom B)` → Returns true if B is entirely within A (excluding boundaries). Geom B lays in geom A, of no points outside geom A. If a line fits on the outline (outer ring) of a polygon, it returns false. Minimum one points should be inside of the polygon. If two polygons are exactly the same, it returns true.

`ST_Within(geome A, geom B)` → Returns true if A is entirely within B (excluding boundaries). Geom A lays in geom B, of no points outside geom B. If a line fits on the outline (outer ring) of a polygon, it returns false. Minimum one points should be inside of the polygon. If two polygons are exactly the same, it returns true.

Don't forget:

`ST_Contains(geom A, geom B) = ST_Within(geom B, geom A)`

`ST_Covers(geom A, geom B)` → Returns true if A completely covers B (including boundaries). Geom B lays in geom A, of no points outside geom A. If a line fits on the outline (outer ring) of a polygon, it returns **true**. If two polygons are exactly the same, it returns true.

`ST_CoveredBy(geom A, geom B)` → Returns true if A is completely covered by B (including boundaries). Geom A lays in geom B, of no points outside geom B. If a line fits on the outline (outer ring) of a polygon, it returns true. If two polygons are exactly the same, it returns true.

Don't forget:

`ST_Covers(geom A, geom B) = ST_CoveredBy(geom B, geom A)`

Combining geometries

`ST_Intersection(geom A, geom B, gridSize)` → Returns the intersecting part(s) of two geometries (A and B). Optional gridSize increases precision performance.

`ST_Difference(geom A, geom B, gridSize)` → Returns the part of A that does not intersect with B →with other words: `Geom A - ST_Intersection(geom A, geom B)`. Optional gridSize increases precision performance.

`ST_SymDifference(geom A, geom B, gridSize)` → Returns the parts that are in A or B but not in both. → with other words: `ST_Union(geom A, geom B) - ST_Intersection(geom A, geom B)`. → or with other words: `ST_SymDifference(geom A, geom B) = ST_SymDifference(geom B, geom A).` Optional gridSize increases precision performance.

`ST_Union(geom A, geom B, gridSize)` → Merges A and B into a single geometry (can be MultiGeometry or GeometryCollection). Optional gridSize increases precision performance.

```
SELECT ST_Union(co_geom) from county -> give the country polygon.
```

`ST_Split(geome A, geom B)` → Splits geometry A using geometry B (e.g., splitting a polygon with a line or point).

Others

`String_Agg()` → A PostgreSQL aggregation function that concatenates string values with a given separator. For example: `String_Agg(d_name, ',')`

Conversion function in PostGIS

These functions convert between PostGIS geometries (WKT or WKB) and other spatial formats (KML, GeoJSON, SVG, etc.)

`ST_GeomFromGeoJSON(text)` → Creates WKT geometry from GeoJSON string.

`ST_GeomFromKML(text)` → Creates WKT geometry from a KML string

`ST_AsGeoJSON(geom)` → Creates GeoJSON geometry from the database record.

`ST_AsKML(geom )` → Creates KML geometry from the database record.

```
select ST_AsKML(ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))',4326));
```

`ST_AsSVG()` → Creates SVG geometry from database record.

`ST_AsLatLonText(geom)` → Returns the coordinates of a point geometry as text in "latitude longitude" format. You can control the numeric format and number of decimal places.

Chapter 7: Practices in POSTGIS

Part 0: SQL conditionals

CASE
WHEN condition THEN result
WHEN condition THEN result
….
ELSE result
END

1. Print the librarian's name and how long she or he has worked in the library.
   - If the hire date is before 1995-01-01, label it as veteran employment.
   - If the hire date is between 1995-01-01 and 2000-01-01, label it as core staff.
   - If the hire date is after 2000-01-01, label it as rookie.

2. Increase the salary of the librarians.
   - Provide a 20% salary increase to those with the position Customer Service Representative**.**
   - Provide a 15% salary increase to those with the position Storekeeper**.**
   - Provide a 25% salary increase to those with the position Administrative**.**
   - Provide a 17% salary increase to those with the position Team Leader**.**

Update *hungary* database, specifically the county table. Change `co_name` from 'megye' to 'vármegye'**.**

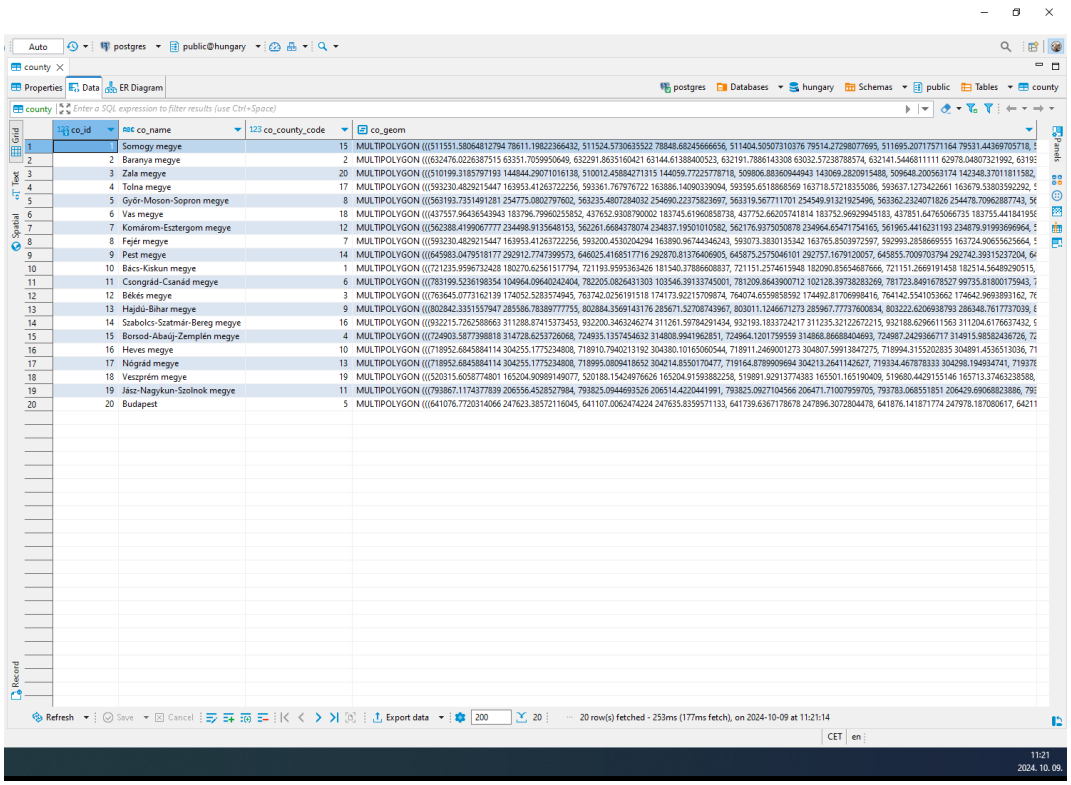
3. Add a new column to the county table and update each county name, replacing 'megye' with 'vármegye'.

Part 1.: Creating and querying geometry

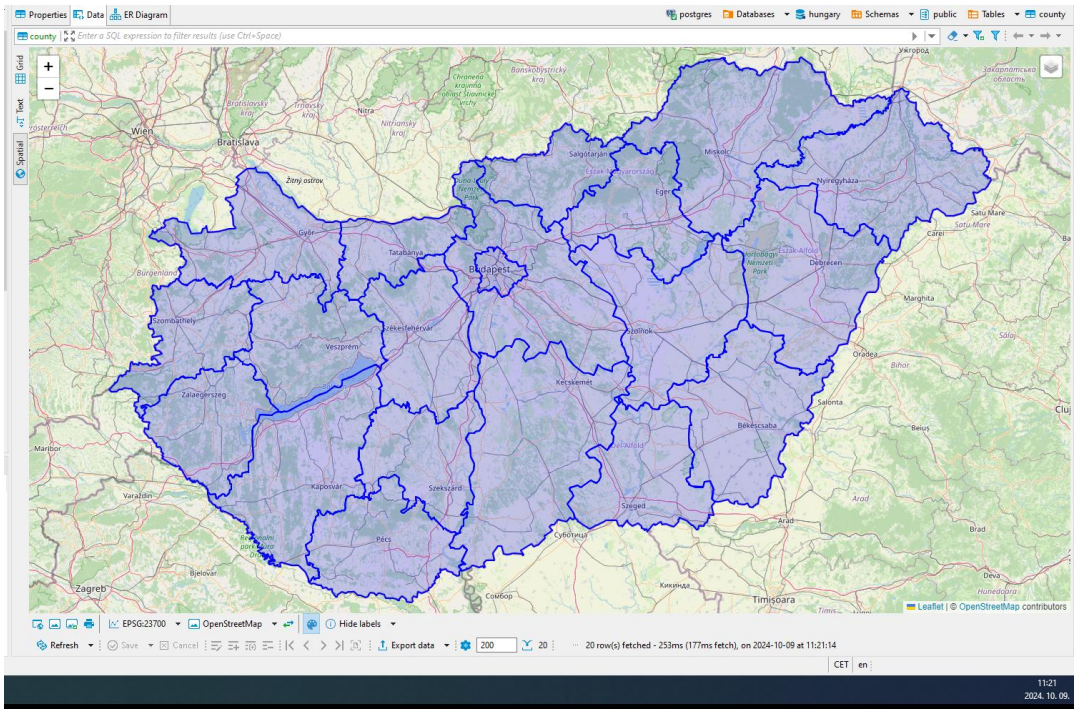
1. Create a new point at Budapest using the ST_Point function (coordinates lat: 47.5° and lon: 19.1°).
2. Use the same coordinates, but define a coordinate system using EPSG:4326!
3. Provide this point in WKT format, and create a geometry representing Budapest (47.5°, 19.1°).
4. Add the spatial reference system ID (SRID) to this point (EPSG:4326)!
5. Create the same point with projection using the ST_GeomFromEWKT() function.!
6. Create a WKT LineString, using these cities as nodes: Budapest–Sárbogárd–Siófok (47.5° 19.1°, 46.88° 18.62°, 46.9° 18.05°).
7. Create a WKT polygon, using the following cities as nodes: Budapest–Sárbogárd–Siófok–Tatabánya (47.5° 19.1°, 46.88° 18.62°, 46.9° 18.05°, 47.58° 18.4°).
8. Add an inner ring/hole to the previous polygon. The nodes of the inner ring: (47.22° 18.66°, 47.24° 18.6°, 47.18° 18.55°)!
9. Convert the line (without the hole) to polygon using ST_MakePolygon (Budapest–Sárbogárd–Siófok–Tatabánya: 47.5° 19.1°, 46.88° 18.62°, 46.9° 18.05°, 47.58° 18.4°).
10. Create a new line between Budapest and Székesfehérvár (47.5° 19.1°, 47.2° 18.4°) using ST_MakePoint, ST_MakeLine!
11. Get familiar with Hungary dataset! Query the geometries from the following table 'settlement_centroid', 'creek', 'district' tables. Try to understand the geometry types based on the previous chapter.

12. Query the type of geometry, the dimension of geometry and the dimension of coordinates for the settlement_centroid, creek and district tables! Let's compare what we find!
13. Check the number of nodes in both simple and multi-geometries (ST_Npoints, ST_NumPoints). You can use the creek and district tables. For simple geometries, use those from exercises 3, 6, and 7.
14. Check the number of geometries in the builtup_area table.
15. Query all counties. Check the spatial view!

Data view – Grid view



Spatial View



Part 2. Measurements, coordinates, projection

1. What is the projection of the features in county table? Print the name of the county and the EPSG code of the projection.
2. What is the area of each county? Print the county name and the area size! What is the unit of measurement?
3. Print the area of all counties in square kilometers ($km^{2)}$ sorted in descending order.
4. Print the area of each county in the layer (table)'s original projection (Hungarian EOV, EPSG: 23700) and in Web Mercator projection (EPSG: 3857)! Sort the data in ascending order based on EOV projection.
5. What is the perimeter of each county? What is the unit of measurement?
6. Print the perimeter of each county in the layer/table's original projection (Hungarian EOV, EPSG: 23700) and in Web Mercator projection (EPSG: 3857)! Sort the data in ascending order based on EOV projection.
7. What is the length of each creek? Print their names and lengths!
8. Print the length of each creek in the layer/table's original projection (Hungarian EOV, EPSG: 23700) and in Web Mercator projection (EPSG: 3857)! Sort the data in ascending order based on EOV projection.
9. Print the X and Y coordinates of the centroid of the settlements in separate columns!
10. What is the distance between Aggtelek and Barcs using the settlement layer?
11. What is the distance between the centroid of Aggtelek and Barcs? Use the settlement layer. (Note: ST_Centroid calculates the centroid of a feature)
12. What is the EOV coordinates (EPSG: 23700) and the geographic coordinates (EPSG: 4326) of the centroid of Barcs?
13. What is the distance in meters between Barcs and the point at X=600 000 and Y=200 000 (EPSG: 23700)?
14. What is the distance in meters between Barcs and the point Lat=47.5° and Lon=19.1° (EPSG: 4326)?
15. What is the distance in meters between the points: 47.5°, 19,1° and 48.2°, 20.2° (EPSG:4326)?
16. What is the shortest and longest distance between the polygons of Barcs and Aggtelek?
17. Display these (shortest and the longest) lines on the map! Check them on the map!
18. If you travel along the shortest path between Lake Balaton and city of Barcs, which node of the Balaton polygon is the closest to Barcs?
19. Which settlement contains this point? (Use St_Contains(geomA, geomB) → where geom A is the settlement, geomB is the point)
20. Which settlement is the farthest settlement from Barcs? What is the distance? (Use settlement_centroid table)
21. What is the azimuth of the line drawn from Barcs to Magosliget?
22. And what is the azimuth of the line drawn from Magosliget to Barcs (the reversed line)

Part 3. Accessing, editing, and processing of geometry

1. Print the bounding box of Barcs! Check it the Spatial View.
2. Print the coordinates of the bounding box of Barcs.
3. Print the bounding box of Barcs with geographic coordinates.
4. Print the minimum bounding rectangle (MBR) of Barcs. Check it the Spatial View.
5. Print the convex hull of Barcs. Check it the Spatial View.
6. What is the area and perimeter of the convex hull?
7. Print the concave hull of Barcs using the following parameters for the second argument: 0,1, 0.25,0.5, 0.75. Check it the Spatial View.
8. Print the centroid of Barcs.
9. Transform the centroid coordinates to the Mercator projection (3857)!
10. Print the outline (boundary) of Pest and Fejér counties.
11. Create a WKT polygon using the following cities as vertices: Budapest–Sárbogárd–Siófok–Tatabánya (47.5° 19.1°, 46.88° 18.62°, 46.9° 18.05°, 47.58° 18.4°). Add a hole: the nodes of the ring: (47.22° 18.66°, 47.24° 18.6°, 47.18° 18.55°)! Print the outer ring of this polygon.
12. Check whether the ring of the previously created polygon is clockwise or counterclockwise.
13. Create a line using the cities Budapest–Székesfehérvár–Siófok–Tatabánya–Budapest. (47.5° 19.1°, 47.2° 18.4°, 46.9° 18.05°, 47.58° 18.4°, 47.5° 19.1°). Is the line closed? Is it a ring?
14. Check the line created in the previous task has any self-intersections.
15. Create a 1km buffer zone around 'Dera-patak'.
16. Ensure the buffer has square end caps, and beveled joins.
17. Display only the left side of the buffer zone!
18. Simplify the path of Dera-patak's using Douglas-Peucker and Visvalingam-Whyatt algorithms! Compare the results. Check how each algorithm works.
19. Apply Chaikin-smoothing to Dera-patak. Check how the Chaikin algorithm works!
20. Create a triangulated irregular network (TIN) subdivision for Barcs!

Part 4.: Understanding spatial relations, geoprocessing

1. Check whether the Danube (Duna) and Tisza intersect (use river_polygon table).
2. Check whether the Danube (Duna) and Tisza are disjoint.
3. Check whether the point (616100, 206300 in Hungarian EOV projection) is located within Fejér county ('Fejér megye').
4. Check whether the point (47.2° N, 18.6° E) is located within Fejér county ('Fejér megye').
5. Check whether Fejér county touches Pest county? What are the results (true or false), for the following spatial relationships: intersects, touches, disjoint, crosses, overlaps, within, contains. Explain why?
6. Check whether Fejér county contains the polygon of Székesfehérvár?
7. Check whether Fejér county overlap or crosses the polygon of Székesfehérvár?
8. Test the ST_COVERS/ST_CONTAINS and the ST_COVEREDBY/ST_WITHIN functions. Observe their subtle differences.
   -Does this polygon 'POLYGON((0 0,0 1,1 1,1 0,0 0))' contains/covers an identical polygon POLYGON((0 0,0 1,1 1,1 0,0 0)) ?
   -Does this linestring 'LINESTRING(0 0,0 1,1 1,1 0)' contains/covers the polygon POLYGON((0 0,0 1,1 1,1 0,0 0))?
   Does this linestring 'LINESTRING(0 0,0.5 0.5,0 1,1 1,1 0)' ' contains/covers the polygon? POLYGON((0 0,0 1,1 1,1 0,0 0))
   -Does the point 'POINT(0 0)' contains/covers the polygon POLYGON((0 0,0 1,1 1,1 0,0 0)) ?
   -Run the same tests using ST_COVEREDBY/ST_WITHIN functions!

9. Draw section of the Danube (Duna) that lies within Fejér county!
10. Draw all national parks that are outside Veszprém county! If a national park is partially within the county, only display the part outside it.
11. Print the names of settlements in Fejér county! (use settlement_centorid table)
12. Print the names and geometry of settlements in Fejér county!
13. Print the names of all settlements not located in Fejér county!
14. How many settlements are not located in Fejér county!
15. Draw the section of M7 motorway that passses through Fejér county!
16. What is the total length of this road?
17. What is the length of one direction (either backward or forward) in kilometers?
18. What is the total area of the protected regions in Veszprém county (in km^2)?
19. What percentage of Veszprém County is protected?
20. What percentage of the **entire country** is protected?
21. Which geoboxes are located in Fejér county?
22. How many geoboxes are located within Fejér county?
23. Which geoboxes are located outside Fejér county?
24. Which geoboxes are located in the 10 km buffer zone around the Danube (Duna)?
25. Which geoboxes are located in the 10 km buffer zone around the M7 motorway?
26. Which geoboxes are located in the 10 km buffer zone around M7 motorway in Fejér county?
27. Which creek is the closest to Hollókő castle. What is this distance?
28. Which node of this creek is the closest to Hollókő? (Print its coordinates.)
29. Split Hungary (polygon) into two halves along the 47° latitude (between 16°and 22°).
30. Draw the areas that belong to either the 'Budai Tájvédelmi Körzet' or Budapest, but exclude the areas where these features intersect.
31. Print the district name and and associated castle(s), ordered in ascending order.
32. Formulate a question for the following query!

```
select b_name, string_agg(va_name,', ') from belterulet, var where
st_contains(st_intersection((select m_geom from megyek where m_name='Nógrád megye'),
b_geom),st_transform(va_geom, 23700)) group by b_name order by 1
```

33. Print all projection pairs, where the sum of their EPSG code is 23700!
34. Which districts border 'Székesfehérvári járás'?
Create a list of districts, with each row containing the district name and a second column listing its neighboring districts.

Chapter 8: Key

Part 0: The CASE conditional

1.
```
select li_name, li_job,

case
      when li_enter between '1995-01-01' and '2000-01-01' then 'Core staff'
      when li_enter >'2000-01-01' then 'rookie'
      else 'Veteran employment'
end jobrange
from librarian
```

2.
```
select li_name, li_job, li_salary,

case
      when li_job='Customer Service Representative' then li_salary*1.2
      when li_job='Storekeeper' then li_salary*1.15
      when li_job='Administrator' then li_salary*1.25
      when li_job='Team Leader' then li_salary*1.17
end as newsalary
from librarian
```

3.
```
ALTER TABLE public.county ADD new_name varchar NULL;

update county set new_name =(concat(substring(co_name,1,(Length(co_name)-
5)),'vármegye')) where Substring(co_name,(Length(co_name)-5))=' megye'
```

Part 1.: Creating and querying geometry

1. `select ST_Point(19.1, 47.5);`

2. `select ST_SetSRID(ST_Point(19.1, 47.5), 4326);`

3. `select ST_GeomFromText('POINT(19.1 47.5)');`

4. `select ST_GeomFromText('POINT(19.1 47.5)', 4326);`

5. `select ST_GeomFromEWKT('SRID=4326;POINT(19.1 47.5)');`

6. `select ST_GeomFromText('LINESTRING(19.1 47.5, 18.62 46.88, 18.05 46.9)', 4326);`

7. `select ST_GeomFromText('POLYGON((19.1 47.5, 18.62 46.88, 18.05 46.9, 18.4 47.58, 19.1 47.5))', 4326);`

8. `select ST_GeomFromText('POLYGON((19.1 47.5, 18.62 46.88, 18.05 46.9, 18.4 47.58, 19.1 47.5),(18.66 47.22,18.6 47.24, 18.55 47.18,18.66 47.22))', 4326);`

9. `select ST_MakePolygon('LINESTRING(19.1 47.5, 18.62 46.88, 18.05 46.9, 18.4 47.58, 19.1 47.5)');`

10. `select ST_MakeLine(ST_MakePoint(19.1, 47.5), ST_MakePoint(18.4, 47.2));`

11.

```
select sc_geom from settlement_centroid;

select cr_geom from creek;

select d_geom from district;
```

12. `select ST_GeometryType(sc_geom), GeometryType(sc_geom), ST_Dimension(sc_geom), ST_CoordDim(sc_geom) from settlement_centroid`

`select ST_GeometryType(cr_geom), GeometryType(cr_geom), ST_Dimension(cr_geom), ST_CoordDim(cr_geom) from creek`

`select ST_GeometryType(d_geom), GeometryType(d_geom), ST_Dimension(d_geom), ST_CoordDim(d_geom) from district`

13. `select ST_NPoints(d_geom) from district;`

`select ST_NumPoints(d_geom) from district;` → NULL

`select ST_NPoints(cr_geom) from creek;`

`select ST_NumPoints(cr_geom) from creek;` → NULL

`select ST_NumPoints(ST_GeomFromText('LINESTRING(19.1 47.5, 18.4 47.2, 18.05 46.9)', 4326));`

`select ST_NPoints(ST_GeomFromText('LINESTRING(19.1 47.5, 18.4 47.2, 18.05 46.9)', 4326));`

`select ST_NPoints(ST_GeomFromText('POLYGON((19.1 47.5, 18.4 47.2, 18.05 46.9, 18.4 47.58, 19.1 47.5))', 4326));`

`select ST_NumPoints(ST_GeomFromText('POLYGON((19.1 47.5, 18.4 47.2, 18.05 46.9, 18.4 47.58, 19.1 47.5))', 4326));`→ NULL

14. `select ST_NumGeometries(ba_geom) from builtup_area;`

15. `select * from county;`

Part 2. Measurements, coordinates, projection

```
1. select co_name, ST_Srid(co_geom) from county;
2. select co_name, ST_Area(co_geom) from county; /Unit in square meter/
3. select co_name, ST_Area(co_geom)/1000000 from county order by 2 desc
4. select co_name, ST_Area(co_geom)/1000000 as eov,
   ST_Area(ST_Transform(co_geom,3857))/1000000 as webmerc from county order by
   2 asc
5. select co_name, ST_Perimeter(co_geom) from county; /meter/
6. select co_name, ST_Perimeter(co_geom) as eov,
   ST_Perimeter(ST_Transform(co_geom, 3857)) as webmerc from county order by 2
7. select cr_name, ST_Length(cr_geom) from creek; /meter/
8. select cr_name, ST_Length(cr_geom) as eov, ST_Length(ST_Transform(cr_geom,
   3857)) as webmerc from creek;
9. select ST_X(sc_geom), ST_Y(sc_geom) from settlement_centroid;


10.select ST_Distance((select s_geom from settlement where
   s_name='Aggtelek'),(select s_geom from settlement where s_name='Barcs'))

11.select ST_Distance((select ST_Centroid(s_geom) from settlement where
   s_name='Aggtelek'),(select ST_Centroid(s_geom) from settlement where
   s_name='Barcs'))

12.select ST_Centroid(s_geom) as eov, ST_Transform(ST_Centroid(s_geom),4326)
   as geog from settlement where s_name='Barcs'

13.select ST_Distance((select sc_geom from settlement_centroid sc where
   sc_name='Barcs'), ST_GeomFromText('POINT(600000 200000)',23700))

   or
   select ST_Distance((select s_geom from settlement where s_name='Barcs'),
   ST_GeomFromText('POINT(600000 200000)',23700))

14.select ST_Distance((select s_geom from settlement where s_name='Barcs'),
   ST_Transform(ST_GeomFromText('POINT(19.1 47.5)',4326),23700))
   or
   select ST_Distance((select sc_geom from settlement_centroid sc where
   sc_name='Barcs'), ST_Transform(ST_GeomFromText('POINT(19.1
   47.5)',4326),23700))

15.select ST_DistanceSphere(ST_GeomFromText('POINT(19.1 47.5)',4326),
   ST_GeomFromText('POINT(20.2 48.2)',4326)); /Unit is meter/
16.select ST_Distance((select s_geom from settlement where
   s_name='Aggtelek'),(select s_geom from settlement where s_name='Barcs')),
   ST_MaxDistance((select s_geom from settlement where
   s_name='Aggtelek'),(select s_geom from settlement where s_name='Barcs'))
17.select ST_ShortestLine((select s_geom from settlement where
   s_name='Aggtelek'), (select s_geom from settlement where s_name='Barcs')),
   ST_LongestLine((select s_geom from settlement where s_name='Aggtelek'),
   (select s_geom from settlement where s_name='Barcs'))
18.select ST_ClosestPoint((select la_geom from lake where
   la_name='Balaton'),(select s_geom from settlement where s_name='Barcs'))
19.select * from settlement where ST_Contains(s_geom, ST_ClosestPoint((select
   la_geom from lake where la_name='Balaton'),(select s_geom from settlement
   where s_name='Barcs')))
```

```
20. select sc_name, ST_Distance((select sc_geom from settlement_centroid where
    sc_name='Barcs'), sc_geom) from settlement_centroid  order by 2 desc limit
    1
21. select ST_Azimuth((select sc_geom from settlement_centroid sc where
    sc_name='Barcs'), (select sc_geom from settlement_centroid where
    sc_name='Magosliget') )*180/pi()
    or
    select degrees(ST_Azimuth((select sc_geom from settlement_centroid sc where
    sc_name='Barcs'), (select sc_geom from settlement_centroid where
    sc_name='Magosliget')))
 22. select ST_Azimuth((select sc_geom from settlement_centroid sc where
     sc_name='Magosliget'), (select sc_geom from settlement_centroid where
     sc_name='Barcs') )*180/pi()
      or
      select degrees(ST_Azimuth((select sc_geom from settlement_centroid sc
      where sc_name='Magosliget'), (select sc_geom from settlement_centroid
      where sc_name='Barcs')))
```

Part 3. Accessing, editing, and processing of geometry

1. `select ST_Envelope(s_geom) from settlement where s_name='Barcs'`
2. `select Box2D(s_geom) from settlement where s_name='Barcs'`
   `or select Box(s_geom) from settlement where s_name='Barcs'`
3. `select ST_Transform(ST_Envelope(s_geom),4326) from settlement where s_name='Barcs'`
4. `select ST_OrientedEnvelope(s_geom) from settlement where s_name='Barcs'`
5. `select ST_ConvexHull(s_geom) from settlement where s_name='Barcs'`
6. `select ST_Area(ST_ConvexHull(s_geom)), ST_Perimeter(ST_ConvexHull(s_geom)) from settlement where s_name='Barcs'`
7. `select ST_ConcaveHull(s_geom,0) from settlement where s_name='Barcs'`

   Change the second parameter to 0.25, 0.5, 0.75 and 1.

8. `select ST_Centroid(s_geom) from settlement where s_name='Barcs'`

9. `select ST_Transform(ST_Centroid(s_geom),3857) from settlement where s_name='Barcs'`

10. `select ST_Boundary(co_geom) from county where co_name in ('Fejér megye', 'Pest megye')`

-> Fejér is a LineString, Pest is a MultiLineString because of the ring of Budapest

11. `select ST_ExteriorRing(ST_GeomFromText('POLYGON((19.1 47.5,18.62 46.88,18.05 46.9, 18.4 47.57,19.1 47.5),(18.66 47.22,18.6 47.24, 18.55 47.18, 18.66 47.22))'));`

12. `select ST_IsPolygonCW(ST_GeomFromText('POLYGON((19.1 47.5,18.62 46.88,18.05 46.9, 18.4 47.57,19.1 47.5),(18.66 47.22,18.6 47.24, 18.55 47.18, 18.66 47.22))'));`

→ true

`select ST_IsPolygonCCW(ST_GeomFromText('POLYGON((19.1 47.5,18.62 46.88,18.05 46.9, 18.4 47.57,19.1 47.5),(18.66 47.22,18.6 47.24, 18.55 47.18, 18.66 47.22))'));`

→ false

13. `select ST_IsRing(ST_GeomFromText('LINESTRING(19.1 47.5,18.62 46.88,18.05 46.9, 18.4 47.57,19.1 47.5)')) ;`

-> true

14. `select st_IsSimple(ST_GeomFromText('LINESTRING(19.1 47.5,18.62 46.88,18.05 46.9, 18.4 47.57,19.1 47.5)')) ;`

->true, no self-intersection (try to do alone a self-intersected object and test it!)

15. `select ST_Buffer(cr_geom,1000) from creek where cr_name='Dera-patak'`
16. `select ST_Buffer(cr_geom,1000,'endcap=square') from creek where cr_name='Dera-patak'`
17. `select ST_Buffer(cr_geom,1000,'side=left') from creek where cr_name='Dera-patak'`
18. `select ST_Simplify(cr_geom,100), ST_SimplifyVW(cr_geom,5000), cr_geom from creek where cr_name='Dera-patak'`

Click here to read more about Douglas-Peucker algorithm and Visvalingam-Whyatt algorithm

Both algorithms are point reduction algorithms. Read the articles in the url above. In cartography, we often use them to simplify or generalize lines or polygons. Read the PostGIS Topology Module, before you use these algorithms in polygon simplification.

Check and understand the differences between the tolerance values!

19. `select ST_Chaikinsmoothing(cr_geom,5), cr_geom from creek where cr_name='Dera-patak'`

Chaikin's smoothing

Smoothing is used to smooth a polyline. Smoothing is often used after simplification to make the new line with less nodes. Th appperance is much more curve-like.

20. `select ST_DelaunayTriangles(s_geom), s_geom from settlement where s_name='Barcs'`

Delaunay Triangulation is used to tessellate or subdivide a plane.

Part 4.: Understanding spatial relations, geoprocessing.

1. `select ST_Intersects((select rp_geom from river_polygon where rp_name='Duna'),(select rp_geom from river_polygon where rp_name='Tisza'))` → False
2. `select ST_Disjoint((select rp_geom from river_polygon where rp_name='Duna'),(select rp_geom from river_polygon where rp_name='Tisza'))` → True
3. `select ST_Intersects(ST_SetSRID(ST_Point(616100, 206300),23700), (select co_geom from county where co_name='Fejér megye'))`
4. `select ST_Intersects(ST_Transform(ST_SetSRID(ST_Point(18.6, 47.2),4326),23700), (select co_geom from county where co_name='Fejér megye'))`
5. `select ST_Intersects(ST_Transform(ST_GeomFromText('POINT(18.6 47.2)',4326),23700), (select co_geom from county where co_name='Fejér megye'))`
   `select ST_Touches((select co_geom from county where co_name='Fejér megye'),(select co_geom from county where co_name='Pest megye'))` → True

Use other spatial relations!

`select ST_Contains((select co_geom from county where co_name='Fejér megye'),(select co_geom from county where co_name='Pest megye'))` → False

`select ST_Disjoint((select co_geom from county where co_name='Fejér megye'),(select co_geom from county where co_name='Pest megye'))` → False

`select ST_Within((select co_geom from county where co_name='Fejér megye'),(select co_geom from county where co_name='Pest megye'))` → False

`select ST_Crosses((select co_geom from county where co_name='Fejér megye'),(select co_geom from county where co_name='Pest megye'))` → False

`select ST_Overlaps((select co_geom from county where co_name='Fejér megye'),(select co_geom from county where co_name='Pest megye'))` → False

6. `select ST_Contains((select co_geom from county where co_name='Fejér megye'),(select s_geom from settlement where s_name='Székesfehérvár'))` → True
7. `select ST_Overlaps((select co_geom from county where co_name='Fejér megye'),(select s_geom from settlement where s_name='Székesfehérvár'))` → False
   `select ST_Crosses((select co_geom from county where co_name='Fejér megye'),(select s_geom from settlement where s_name='Székesfehérvár'))` → False
8. `select ST_Within((select ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))')),(select ST_GeomFromText ('POLYGON((0 0,0 1,1 1,1 0,0 0))')))` → True
   `select ST_CoveredBy((select ST_GeomFromText ('POLYGON((0 0,0 1,1 1,1 0,0 0))')),(select ST_GeomFromText ('POLYGON((0 0,0 1,1 1,1 0,0 0))')))` → True
   `select ST_Within((select ST_GeomFromText('LINESTRING(0 0,0 1,1 1,1 0)')),(select ST_GeomFromText ('POLYGON((0 0,0 1,1 1,1 0,0 0))')))` → False
   `select ST_CoveredBy((select ST_GeomFromText('LINESTRING(0 0,0 1,1 1,1 0)')),(select ST_GeomFromText ('POLYGON((0 0,0 1,1 1,1 0,0 0))')))` → True
   `select ST_Within((select ST_GeomFromText('LINESTRING(0 0,0.5 0.5,0 1,1 1,1 0)')),(select ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))')))` → True
   `select ST_CoveredBy((select ST_GeomFromText('LINESTRING(0 0,0.5 0.5,0 1,1 1,1 0)')),(select ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))')))` → True
   `select ST_Within((select ST_GeomFromText('POINT(0 0)')),(select ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))')))` → False

```
select ST_CoveredBy((select ST_GeomFromText('POINT(0 0)')),(select
ST_GeomFromText('POLYGON((0 0,0 1,1 1,1 0,0 0))'))) → True
```

To run the queries above with ST_COVERS or ST_CONTAINS reverse the order of geometries!

```
ST_COVERS(geomA, geomB) <-> ST_CoveredBy(geomB, geomA)

ST_CONTAINS(geomA, geomB) <-> ST_Within(geomB, geomA)
```

```
9.  select ST_Intersection((select co_geom from county where co_name='Fejér
    megye'),(select rp_geom from river_polygon where rp_name='Duna'))
10. select ST_Difference(np_geom,(select co_geom from county where
    co_name='Veszprém megye')) from nature_protection
11. select sc_name from settlement_centroid where ST_Intersects(sc_geom,(select
    co_geom from county where co_name='Fejér megye'))
12. select sc_name, sc_geom from settlement_centroid where
    ST_Intersects(sc_geom,(select co_geom from county where co_name='Fejér
    megye'))
13. select sc_name, sc_geom from settlement_centroid where not ST_Contains((select
    co_geom from county where co_name='Fejér megye'), sc_geom)
14. select count(sc_name) from settlement_centroid where not ST_Contains((select
    co_geom from county where co_name='Fejér megye'), sc_geom)
15. select ST_Intersection((select co_geom from county where co_name='Fejér
    megye'), m_geom ) from motorway where m_ref ilike '%M7%' and
    ST_Intersects((select co_geom from county where co_name='Fejér megye'),
    m_geom)

    this is why M7 text can not be found everywhere in the name column

    or do it with st_union

    select ST_Intersection((select co_geom from county where co_name='Fejér
    megye'), ST_Union(m_geom)) from motorway where m_ref ilike '%M7%'

16. select ST_Length(ST_Intersection((select co_geom from county where
co_name='Fejér megye'), ST_Union(m_geom))) from motorway where m_ref ilike '%M7%'
17. select ST_Length(ST_Intersection((select co_geom from county where
co_name='Fejér megye'), ST_Union(m_geom)))/1000 from motorway where m_ref ilike
'%M7%'and m_hu_ed_dire='forward'
or
select ST_Length(ST_Intersection((select co_geom from county where co_name='Fejér
megye'), ST_Union(m_geom)))/1000 from motorway where m_ref ilike '%M7%'and
m_hu_ed_dire='backward'

18. select ST_Area(ST_Union(ST_Intersection((select co_geom from county where
co_name='Veszprém megye'),np_geom)))/1000000 from nature_protection

19. select ST_Area(ST_Union(ST_Intersection((select co_geom from county where
co_name='Veszprém megye'),np_geom)))/1000000/(select ST_Area(co_geom)/1000000
from county where co_name='Veszprém megye')*100 from nature_protection

20. select ST_Area(ST_Union(np_geom))/1000000/(select
ST_Area(ST_Union(co_geom))/1000000 from county)*100 from nature_protection
```

21. `select * from geobox where ST_Contains((select co_geom from county where co_name='Fejér megye'), gl_geom)`

22. `select count(*) from geobox where ST_Contains((select co_geom from county where co_name='Fejér megye'), gl_geom)`

23. `select * from geobox where ST_Disjoint((select co_geom from county where co_name='Fejér megye'), gl_geom)`

`or select * from geobox where not ST_Contains((select co_geom from county where co_name='Fejér megye'), gl_geom)`

24. `select * from geobox where ST_Intersects((select ST_Buffer(rp_geom, 10000) from river_polygon where rp_name='Duna'), gl_geom)`

25. `select * from geobox where ST_Intersects((select ST_Union(ST_Buffer(m_geom, 10000)) from motorway where m_ref ilike '%M7%'), gl_geom)`

26. `select * from geobox where ST_Intersects(ST_Intersection((select ST_Union(ST_Buffer(m_geom, 10000)) from motorway where m_ref ilike '%M7%'),(select co_geom from county where co_name='Fejér megye')), gl_geom)`

27. `select cr_name, ST_Distance((select ST_Transform(ca_geom,23700) from castles where ca_name='Hollókői vár'), cr_geom) from creek order by 2 asc limit 1`

28. `select ST_Endpoint(ST_ShortestLine((select ST_Transform(ca_geom,23700) from castles where ca_name='Hollókői vár'), cr_geom)) from creek order by 1 asc limit 1`

or

`select ST_ClosestPoint((select ST_Transform(ca_geom,23700) from castles where ca_name='Hollókői vár'), cr_geom) from creek order by 1 asc limit 1`

29. `select ST_Split(ST_Union(co_geom), ST_Transform(ST_GeomFromText('LINESTRING(16.0 47.0, 22.0 47.0)',4326),23700)) from county`

30. `select ST_SymDifference((select ST_Union(np_geom) from nature_protection where np_name='Budai Tájvédelmi Körzet'),(select co_geom from county where co_name= 'Budapest'))`

31. `select d_name, ca_name from district, castles where ST_Contains(d_geom, ST_Transform(ca_geom,23700)) order by d_name`

or

`select d_name, ca_name from district, castles where d_geom ~ ST_Transform(ca_geom,23700)  and ST_Contains(d_geom, ST_Transform(ca_geom,23700)) order by d_name`

32. Draw the section of Route 8 that lies between Bory Castle in Székesfehérvár and Thury Castle in Várpalota. When trimming the section, make sure it ends at the point on the road closest to each castle.

33. `select SRS1.srid, SRS2.srid from spatial_ref_sys SRS1, spatial_ref_sys SRS2 where (SRS1.srid+SRS2.srid)=23700`

divide by 2, because it repeated twice

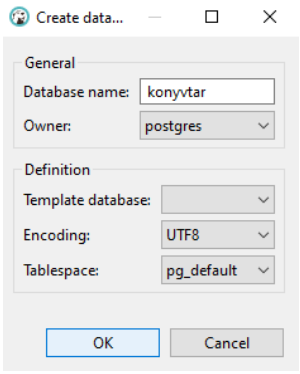
34. `select d2.d_name from district d1, district d2 where ST_Touches(d1.d_geom, d2.d_geom) and d1.d_name='Székesfehérvári járás'`

```
35. select d1.d_name, count(*), string_agg(d2.d_name,', ') from district d1,
district d2 where ST_Touches(d1.d_geom, d2.d_geom) group by 1 order by 1
```

Chapter 9. Data import

Creating a new database, and importing data from SQL files

Right-click on **Database**, and create a new empty database, e.g. *library* (**Create → Database**).



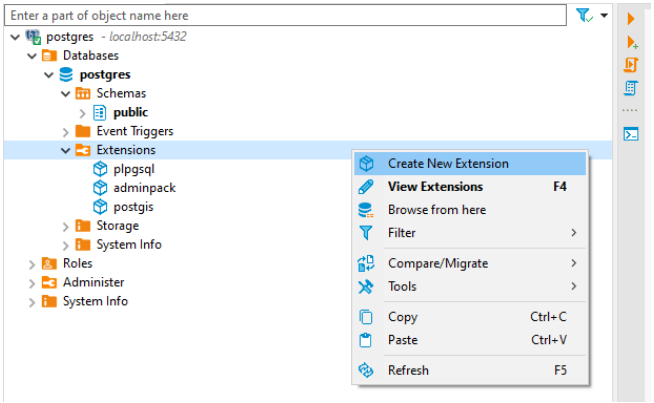
Open the *library.sql* file via **SQL editor** menu → **Open SQL script**.

Execute the script → **Execute script** (use the 3rd icon on the left of the SQL window) then press **F5** (refresh the database). 

If your database is in MySQL, you cannot directly use SQL files to migrate it to PostgreSQL, as MySQL and PostgreSQL dumps are not compatible. In such cases, use CSV text files for database migration instead.

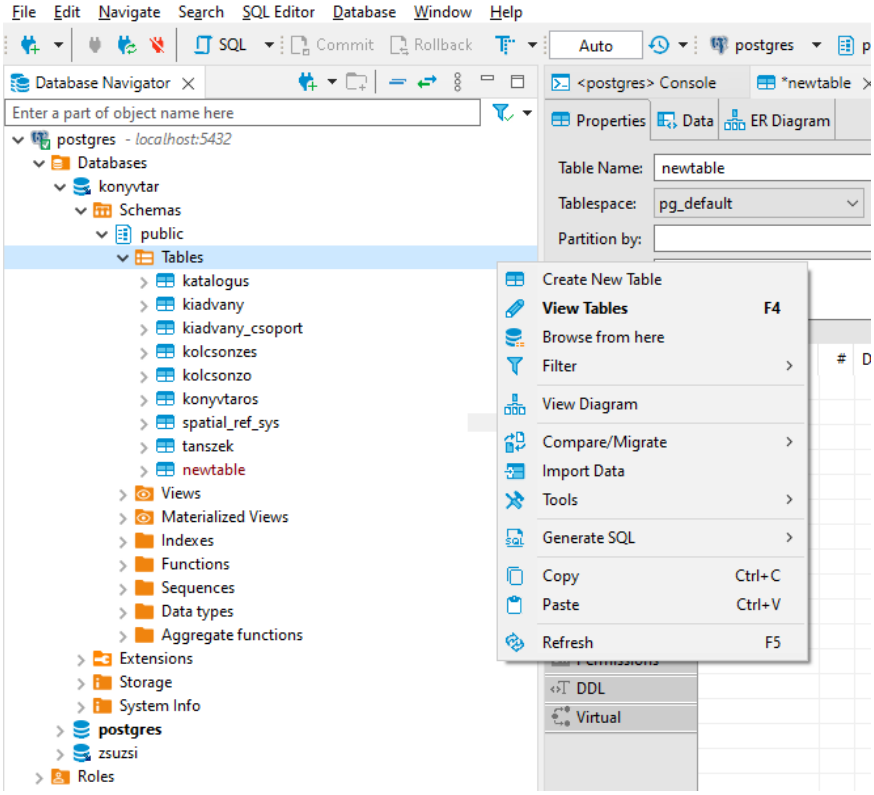
Adding extensions (PostGIS, PostGIS Raster, PostGIS Topology)

A spatial database requires at least the PostGIS extension. Don't forget to add this to your database before importing data. Go to **Database → Schemas → Extensions**, and right-click → **Create New Extension** → Search for the extension by name (e.g., PostGIS) and add it.



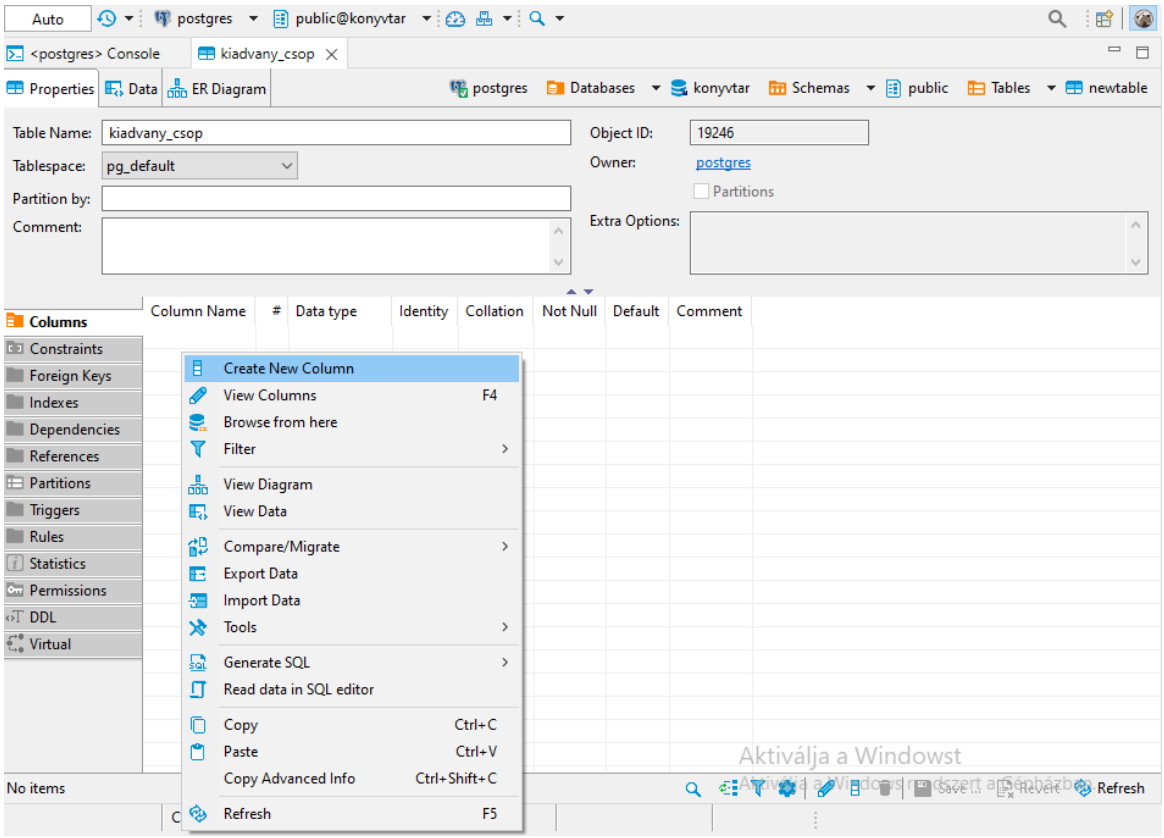
Uploading Data into a PostgreSQL Database from CSV – Using an Existing table structure

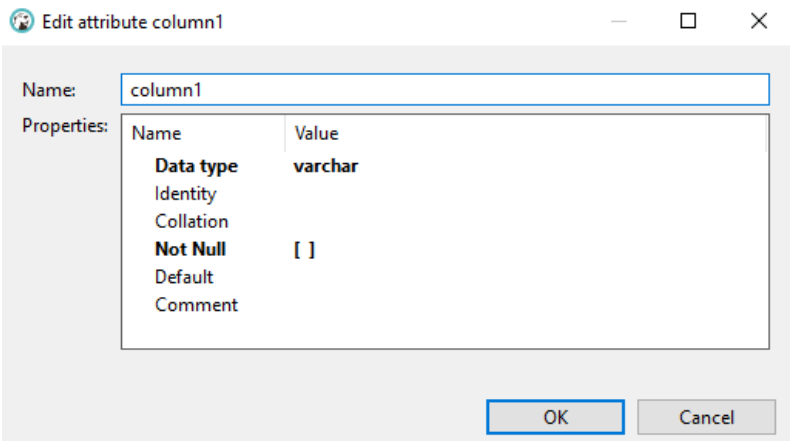
The first step is to create a new empty table. In **Database Navigator** right-click → **Tables** → ***Create New Table***.



In the Properties window, define the table columns, data types, collations, and other settings. To add a new column: right-click under **Columns** → **Create New Column**.

Once ready, don't forget to save the table (**Save** icon is at the bottom of the window).

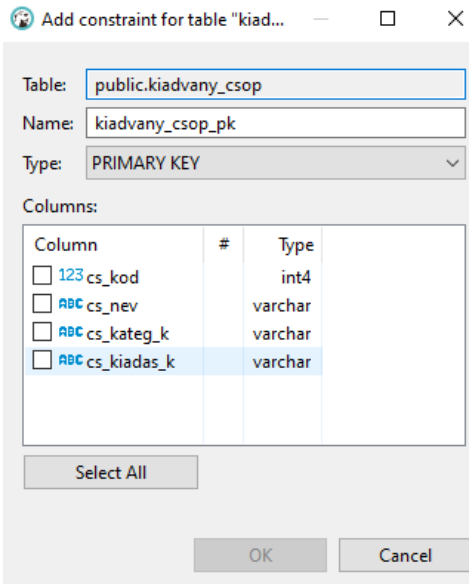




Columns can have constrainst, indexes and keys.

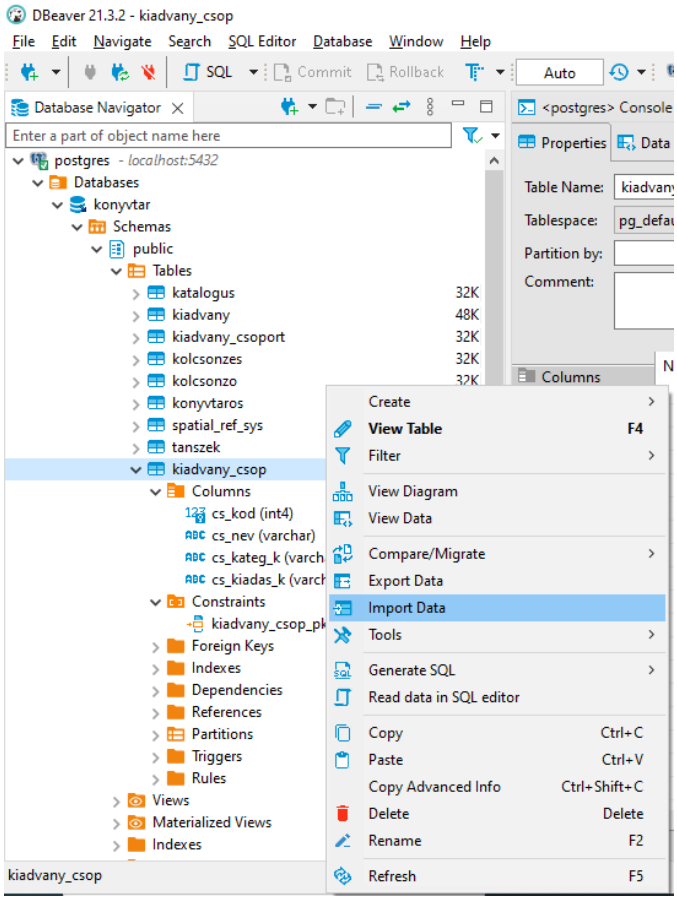
A primary key is a column that contains unique, non-null values (e.g., a sequence like 1, 2, 3...n).

To define it: go to the **Constraints** submenu, right-click → **Create New Constraint** → select **Primary Key**.



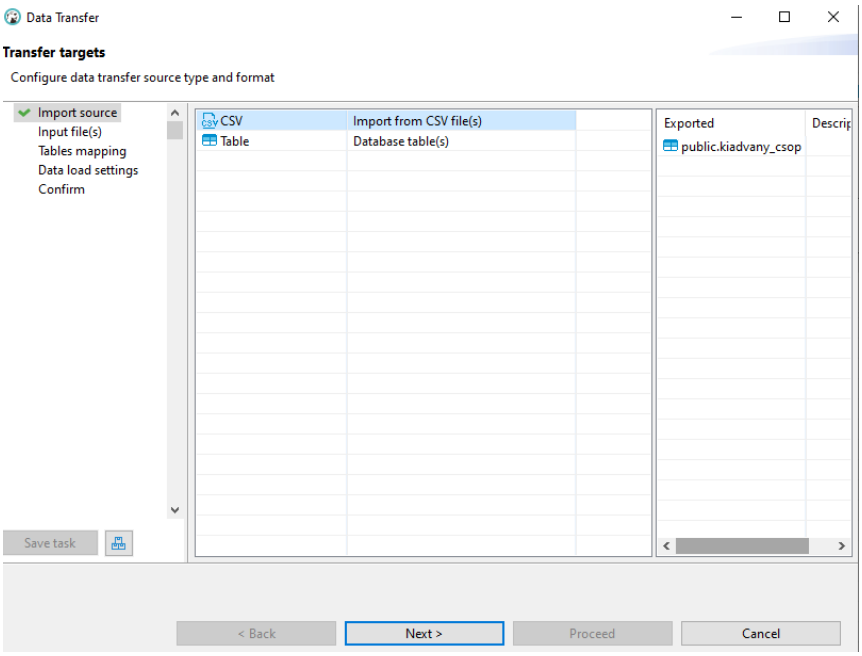
After any modification, always save your changes.

If you already have a CSV file and a predefined table structure, you can import data from the CSV file.



Right-click on the **Database Navigator → Import data.**

Select the file format: CSV. **Next.**



Then configure the CSV properties:

Extension: (CSV, TSV,TXT)

Encoding: character encoding

Column delimiter: semicolon, comma, TAB etc.

Header position: does the file include a header? If so, specify how many rows it spans.

Quote char: e.g.” if used around values

Escape char: usually \

Null value mark: what is the NULL value?, `Null` or „” (empty string)

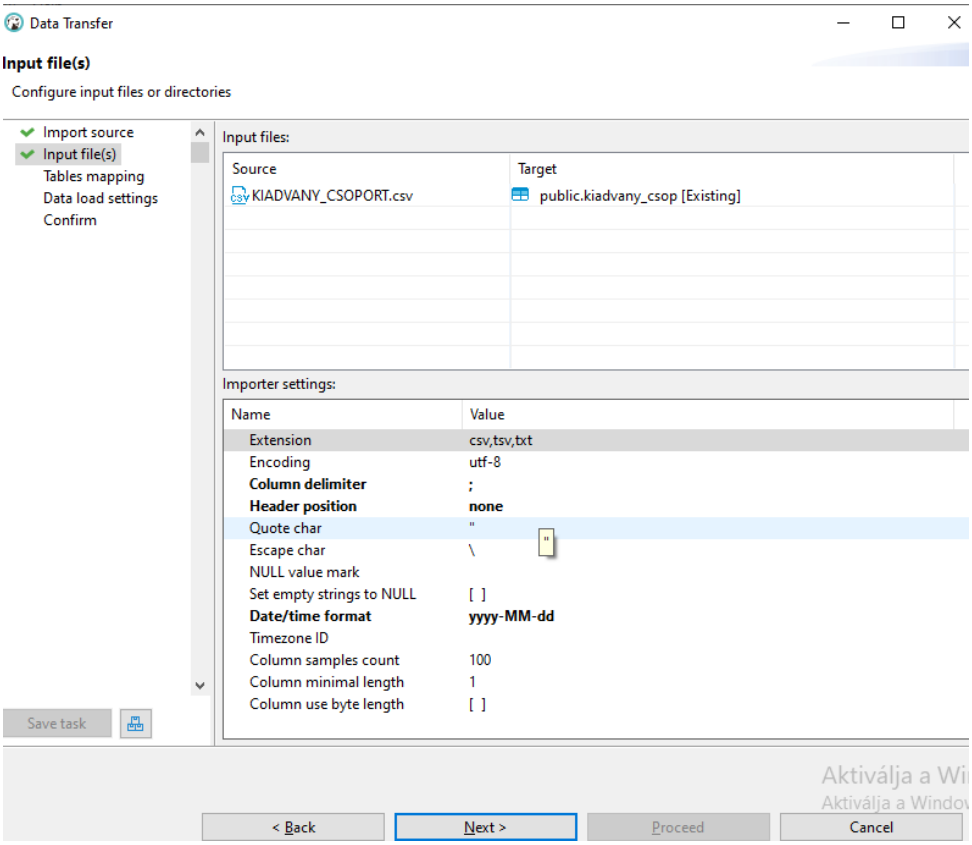
Set empty strings to `Null`

Date Time format: specify the format used in the file

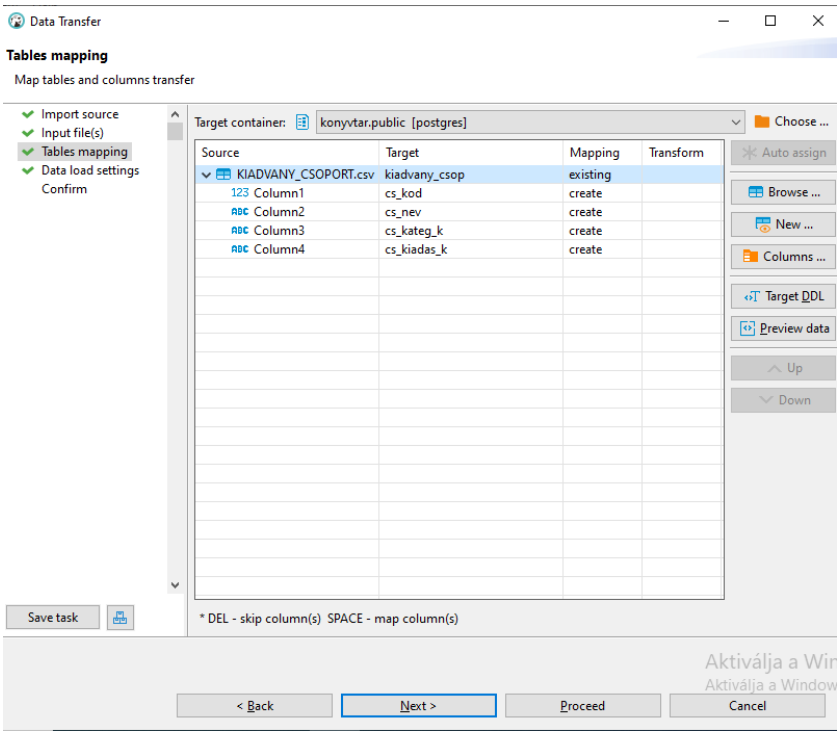
Timezone ID: set if applicable

Column length: optional column width

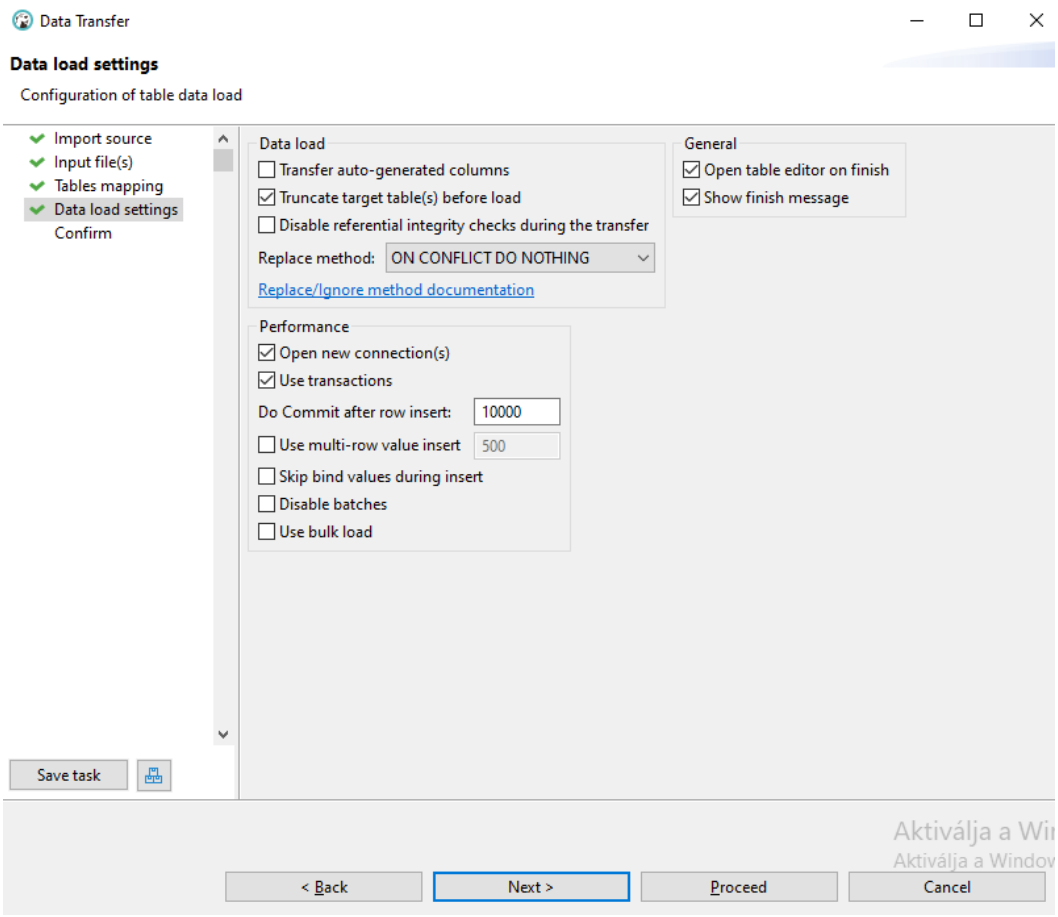
Click **Next.**



In the following window, map the columns in your CSV to the corresponding columns in the database table.



Final settings: Check the option "Truncate target table before load" if you want to delete all existing data from the table before importing new data. Click **Next**. **Proceed**.

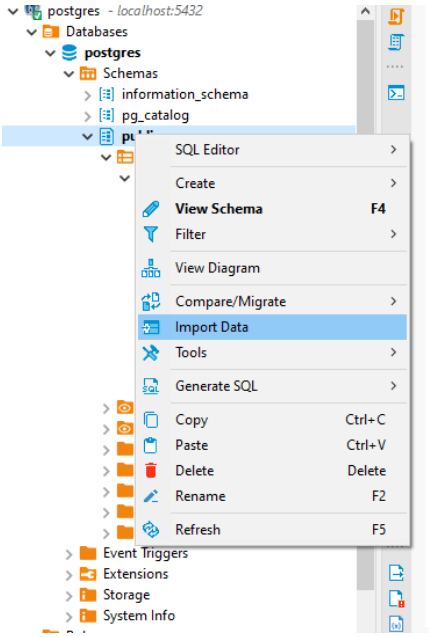


After the import, check the table to confirm the data has been loaded (refresh the table if necessary).

Uploading Data to a PostgreSQL database from a CSV – Missing table structure

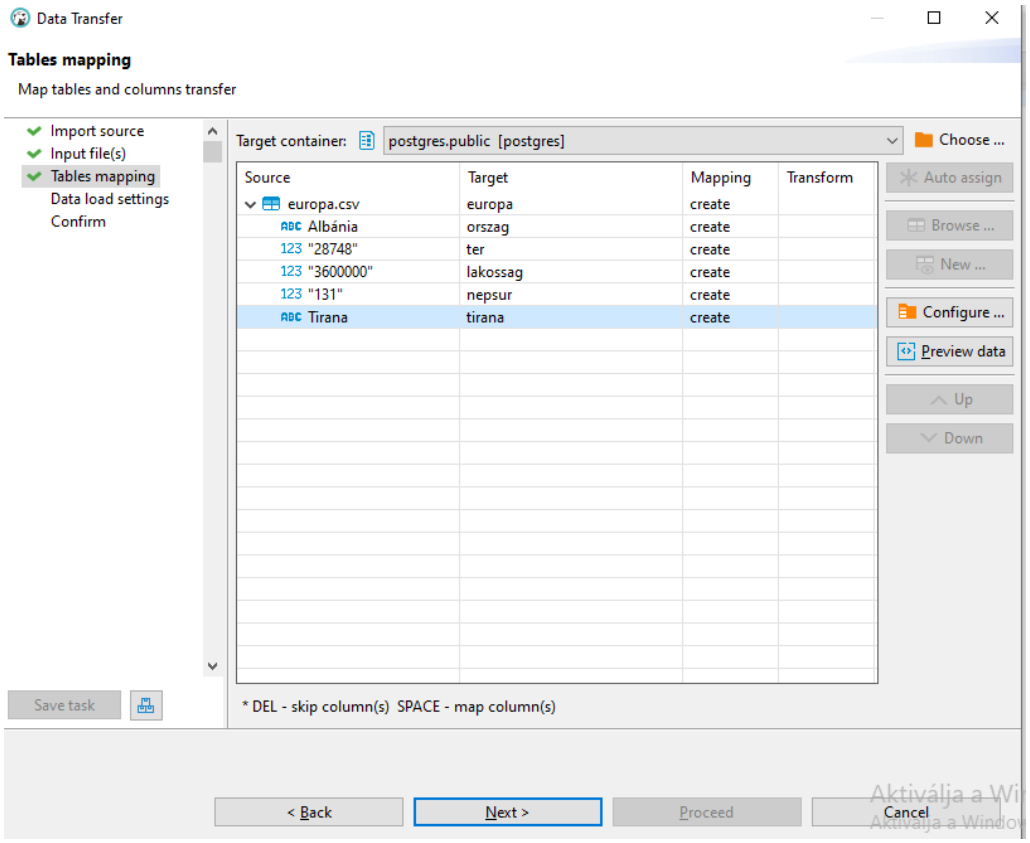
In this task, you need to upload a CSV file and the table structure will be inferred from the file itself. The example used here is the europe.csv file.

Right-click on the **Schemas** section of your database and select → **Import Data.**

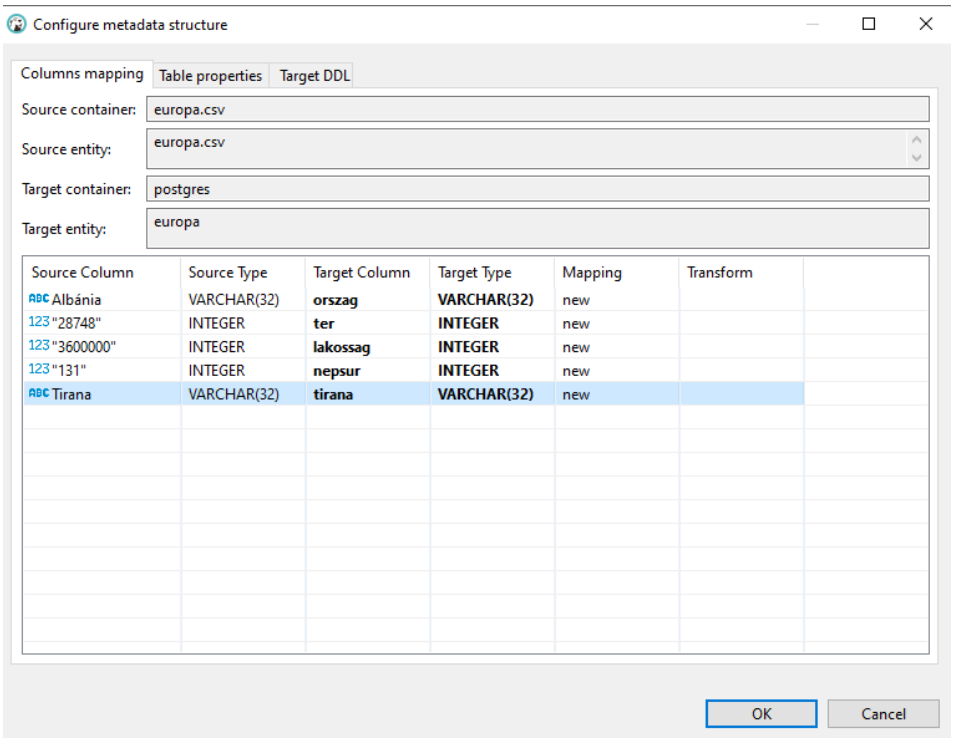


Select the format **CSV** and configure the CSV settings as shown in the previous example.

If the **Table Mapping** settings appear, you can define a new table structure here. In the mapping settings, choose **Create**, which will generate the columns. Then click the **Configure** button.



Here, you can review and adjust the data types for each column if needed.



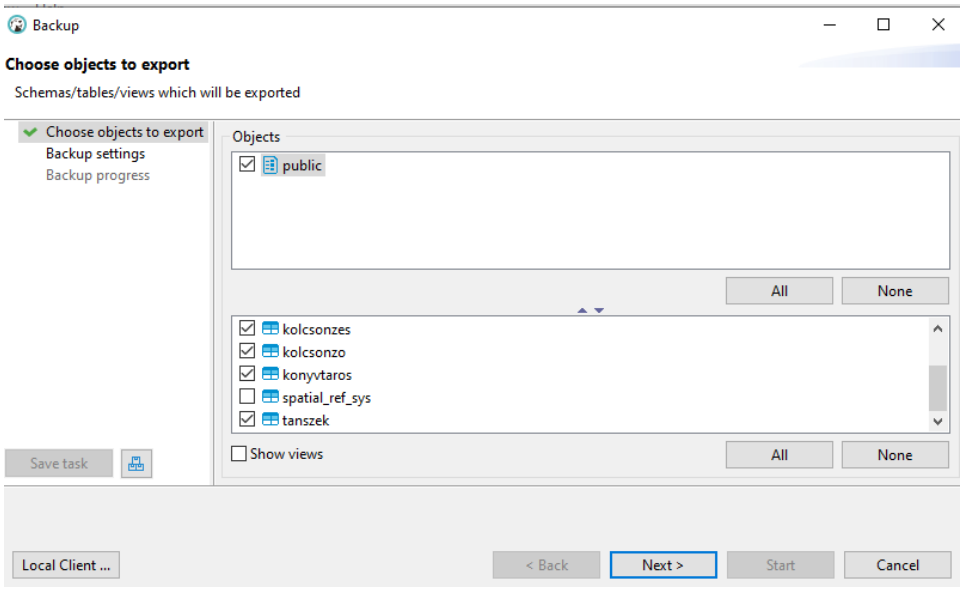
All other settings should be the same as in the previous example. Finally, check the updated table!

Creating a backup

If you want to back up your database or migrate it, use the Backup option.

Right-click on the **Database → Tools → Backup**.

Select the tables you want to include in the backup.



Recommended settings:

- *Format: Plain*. (can be edited/viewed in a text editor).
- *Encoding*: UTF-8

- *Use SQL INSERT instead of copy of rows*: Inserts the rows as SQL statements rather than copying raw data
- *Output folder*: Select the destination folder

→ Click **Start** to begin the backup process.

Upload a table from an another one.

Create a new empty table named 'newtable'. Add the following columns: `n_id serial`, `n_name varchar` and a `n_geom geometry` columns. The primary key is `n_id`. SAVE the changes.

```
create table public.newtable ();

alter table public.newtable add n_id serial NOT NULL;
alter table public.newtable add n_name varchar NULL;
alter table public.newtable add n_geom public.geometry NULL;
```

Insert data from the geobox table, only include rows where the geometry intersects with the county named Fejér megye.

Use: an INSERT INTO + SELECT SQL Statement:

```
insert into newtable (n_name, n_geom) select gl_full_name, gl_geom from geobox
where ST_Intersects(gl_geom, (select co_geom from county where co_name='Fejér
megye'))
```

Data conversion

You can export geometry to GeoJSON format using the `ST_AsGeoJSON` function. This function can convert individual geometries or full rows (with geometry).

Example – convert only the geometry column:

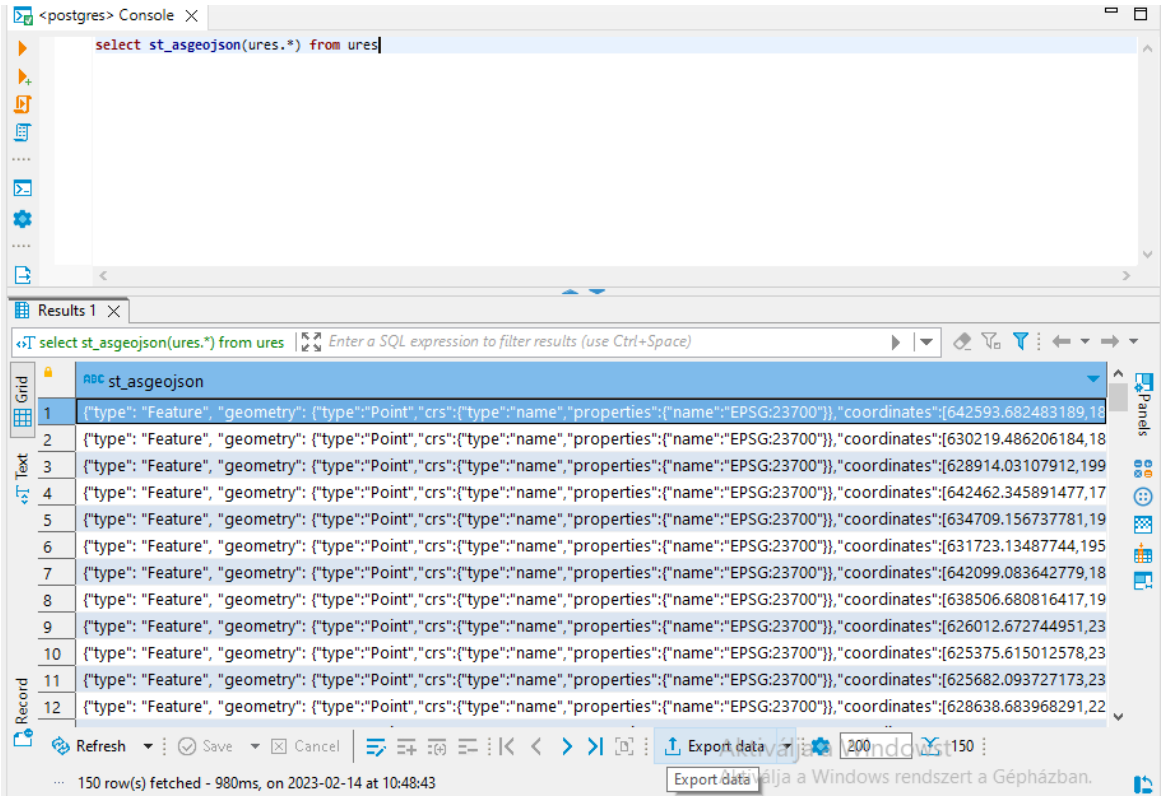
```
select emp_name, ST_AsGeoJSON(emp_geom) from empty_table
```

Example – convert entire records (including all attributes):

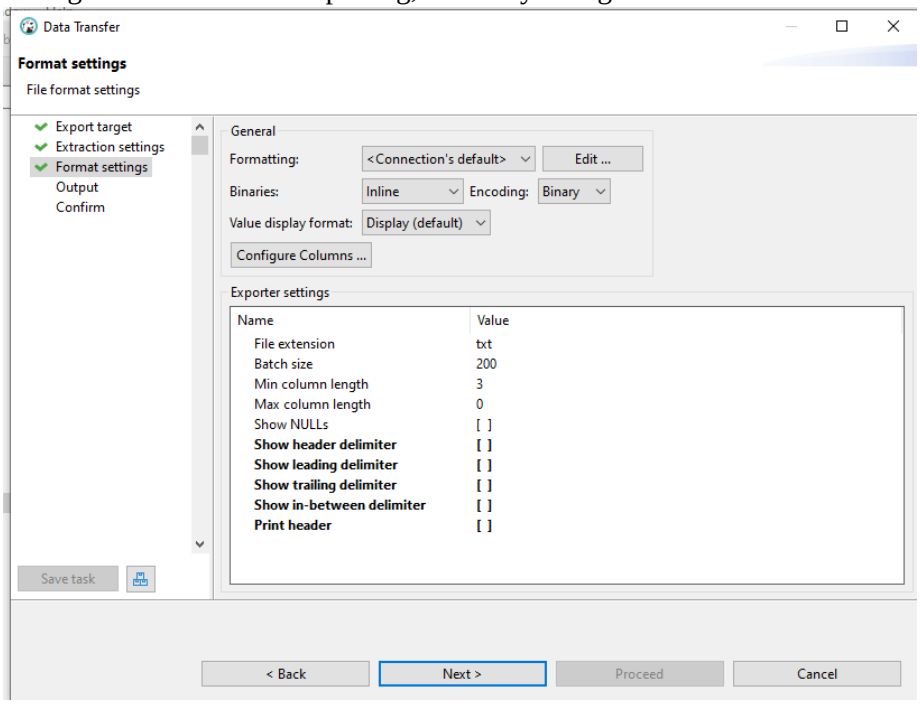
```
select ST_AsGeoJSON(empty_table.*) from empty_table
```

Exporting to a GeoJSON File

Run the query. Below the result table, click on the **Export Data** button.



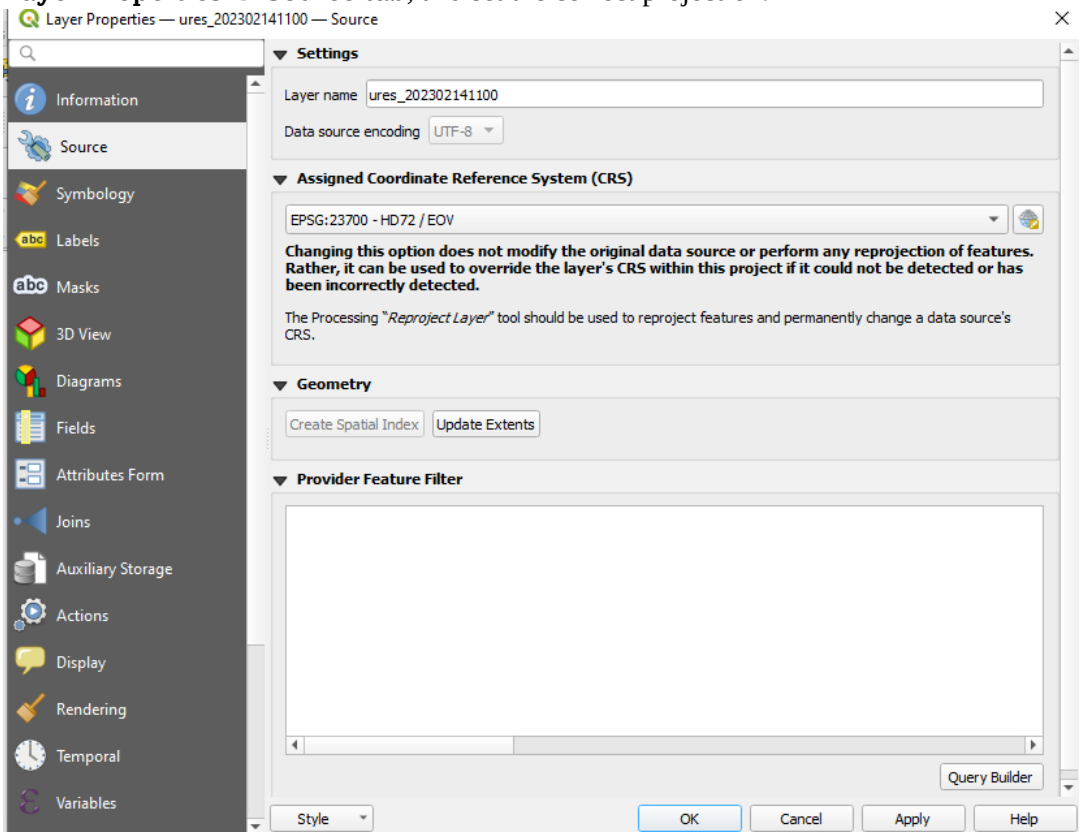
Choose **TXT** as the export format. **Disable delimiters** (refer to the image). Leave the remaining settings as default. After exporting, manually change the file extension to GeoJSON.




```
{"type": "Feature", "geometry":
{"type":"Point","crs":{"type":"name","properties":{"name":"EPSG:23700"}},"coordinates":[642593.682
483189,180088.555515703]}, "properties": {"ur_id": 1, "ur_name": "Dunaújvárosi Madár Liget"}}
{"type": "Feature", "geometry":
{"type":"Point","crs":{"type":"name","properties":{"name":"EPSG:23700"}},"coordinates":[630219.486
206184,189545.599921337]}, "properties": {"ur id": 2, "ur name": "Kastélykutató"}}
```

This file can be used in GIS software such as **QGIS**. Be careful with the coordinate reference system (CRS): GeoJSON expects **EPSG:4326** by default.
In this case, the correct CRS is **EPSG:23700**, so after importing the file into QGIS, go to:

Layer Properties → Source tab, and set the correct projection.



Exporting to KML Format

PostGIS also provides an `ST_AsKML` function. However, it only works with individual geometry columns—it does not support full table rows (i.e., table_name.* is not allowed). This means, you can convert geometries to KML, but attribute data will not be included.

```
select ST_AsKML(emp_geom) from empty_table
```

The Geography data type

The `geography` data type does not use planar (flat) coordinates—it works with spherical coordinates: geographic latitude and longitude.

If you want to convert a geometry to `geography`, you first need to use the `ST_Transform` function to project the geometry into EPSG:4326 (WGS84), which uses geographic coordinates.

The most significant differences between geometric and geographic coordinates appear when calculating distances using functions like `ST_Distance` and `ST_DistanceSphere`.

The following article explains these differences in detail:
http://postgis.net/workshops/postgis-intro/geography.html

Unless strictly necessary, it's often better to avoid using the `geography` type and stick with `geometry`, unless you're working with data covering the entire globe.

Add a `geography` Column to the `empty_table` Table:

Create a new column in the `ures` table with the `geography` data type:

```
alter table public.empty_table add emp_geog public.geography NULL;
```

Populate the new column using transformed geometry data:

```
update empty_table set emp_geog  = ST_Transform(enp_geom,4326);
```

or

```
update empty_table set emp_geog  = ST_Transform(emp_geom,4326)::geography;
```

Which geoboxes (geocaches) in Fejér County are within a 10 km radius of Lake Velence?

```
select emp_name from empty_table where ST_Intersects(emp_geog,(select
ST_Transform(ST_Buffer(la_geom, 10000),4326) from lake where la_name='Velencei-
tó'))
```

Distance Calculations Example

Let's calculate the distance between the geocache "Madár Liget in Dunaújváros" (`emp_id = 1`) and "Bakonykúti" (`emp_id = 28`) using different methods.

When using `ST_Distance` on **geometry**, you get a **planar (projected)** distance. When using `ST_Distance` on **geography**, you get a **distance over the ellipsoid** (WGS84).

When using `ST_DistanceSphere` (after transforming to EPSG:4326), you get the **spherical distance** in meters. The assumed Earth radius is **6,371,008 meters**.

```
select ST_Distance((select emp_geom from empty_table where emp_id=1),(select
emp_geom from empty_table where emp_id=28)),ST_Distance((select emp_geog from
empty_table where emp_id=1),(select emp_geog from empty_table where emp_id=28)) →
65 527.41 m and 65 531.6 m
```

```
select ST_DistanceSphere((select ST_Transform(emp_geom,4326) from empty_table
where emp_id=1),(select ST_Transform(emp_geom,4326) from empty_table where
emp_id=28)),ST_Distance((select emp_geog from empty_table where emp_id=1),(select
emp_geog from empty_table where emp_id=28)) → 65 387.86 m and 65 531.6 m
```

Expected results (approximate):

- Planar vs. ellipsoidal: **65,527.41 m** vs. **65,531.6 m**
- Spherical vs. ellipsoidal: **65,387.86 m** vs. **65,531.6 m**

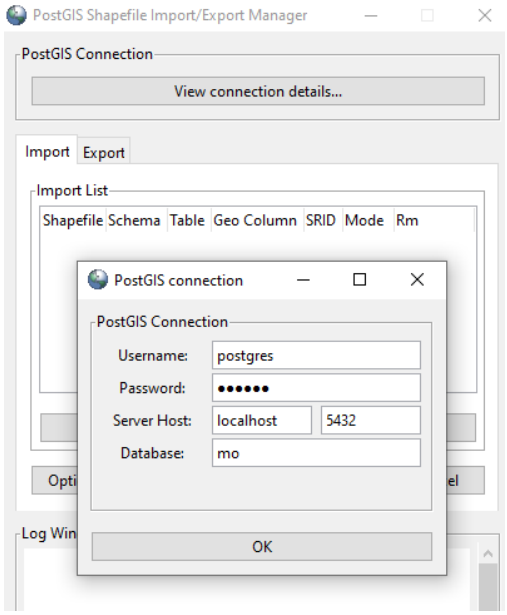
Shapefile import with PostGIS Bundle

The PostGIS Bundle is available on Windows. (On macOS and Linux/Unix systems, the necessary tools are available, but there is no graphical user interface—you must use the command line.)

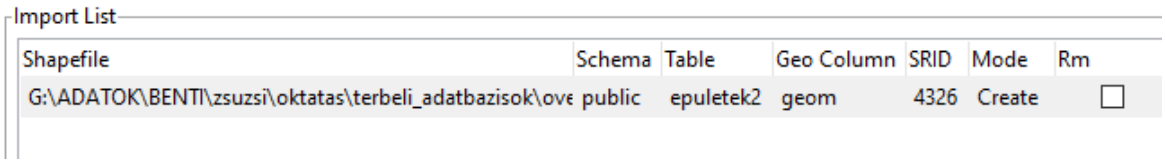
Start the PostGIS Bundle from the Start menu. This tool allows you to import and export Shapefiles to and from a PostgreSQL + PostGIS database.

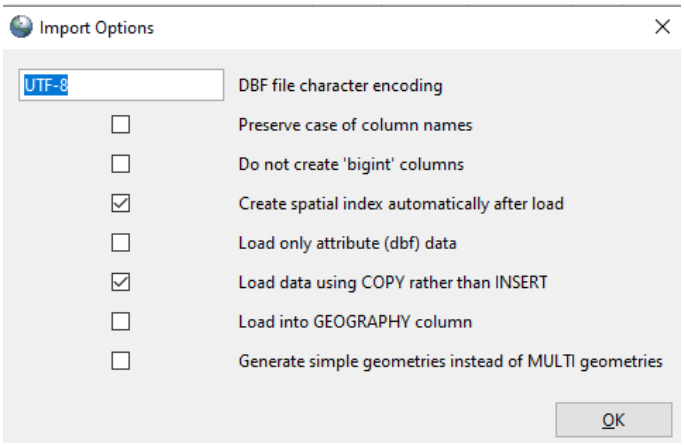
First step is to set the connection details. (PostgreSQL user name+password and port number, and the database name)

Try using the *stops.shp* file to test the import process.



Click **Add File** to select the Shapefile. Here, you can define: the table name, the geometry column name and the SRID (Spatial Reference Identifier). In the Import options, you can also set the character encoding of the DBF file, and enable the creation of a spatial index, etc



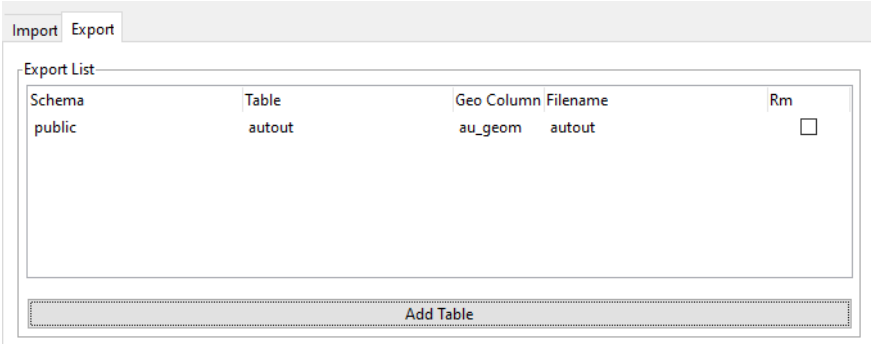


Once everything is configured, click the **Import** button.

Check the Log Window to confirm whether the import was successful.

Exporting Shapefiles

You can also export Shapefiles from the database. To do this, select the table, then specify the geometry column and set the output file name.



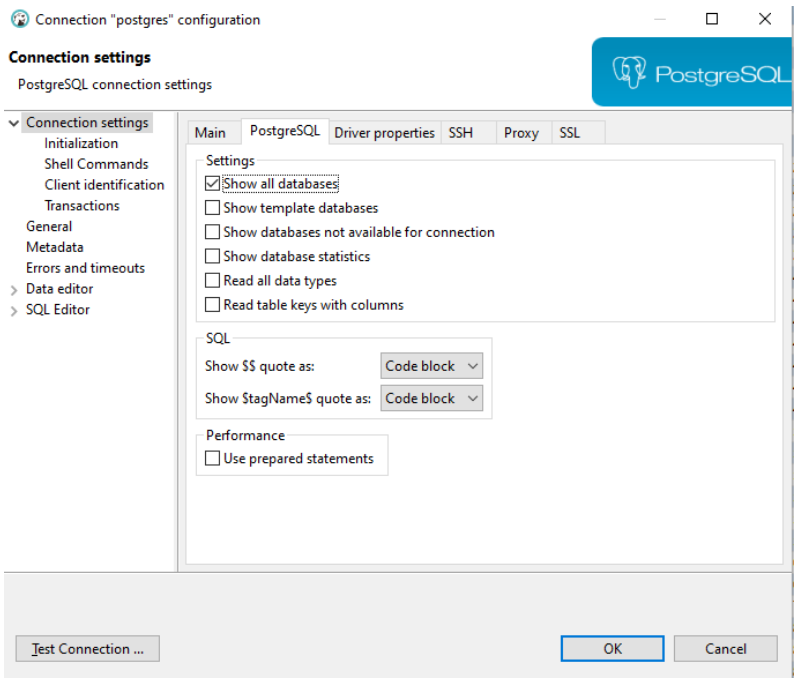
Troubleshooting FAQ

Here are solutions to common issues:

Issue: I can't see the database, even though I created it.

Right-click on the username (e.g., postgres) → **Edit connection**

Go to the **PostgreSQL tab** → check the box **Show all databases**.



Issue: The font in the window is too small. I want to increase it.

Navigate to: **Window → Preferences → User Interface → Colors and Fonts → Basic → Text Font → Click Edit** to increase the font size.

Issue: A long SQL query appears in one line in the SQL Console. I want to format it.

Use the shortcut:
CTRL + SHIFT + F (Auto-format)

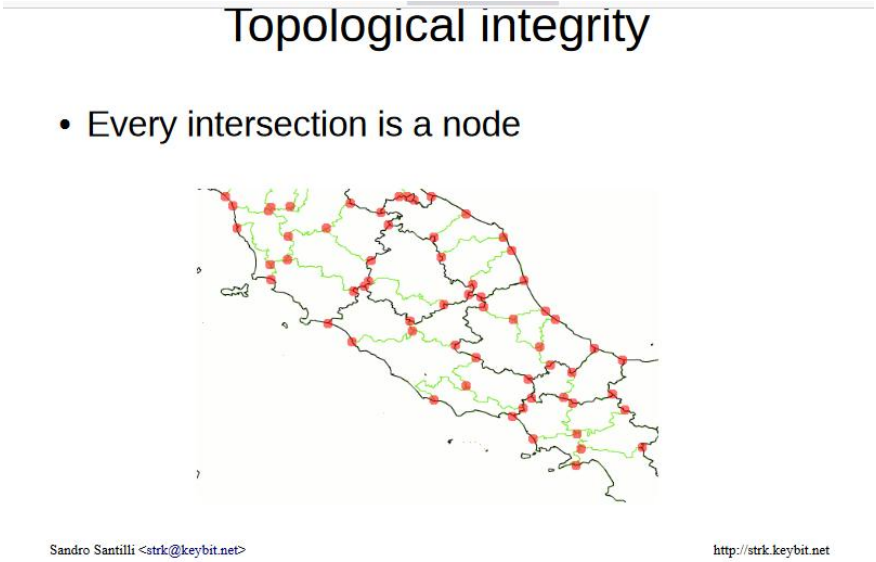
CHAPTER 10: POSTGIS Topology

PostGIS allows us to built topological data structure and work with them. To use these features, add the PostGIS Topology Extension. A detailed description of this module can be found here:

https://postgis.net/docs/Topology.html

Why use a topological data structure instead of standard geometry?

- To store spatial relationships explicitly
- Reduced storage size – every edge is stored once → more compact geometry representation
- Ensures topological integrity: every intersection becomes a node
- Shared edges are stored once, rather than being duplicated for each polygon



Source of image: http://strk.kbt.io/projects/postgis/Paris2011_TopologyWithPostGIS_2_0.pdf

Topological data structure uses four data type for storing data:

- nodes (marked with red dots in the image above)
- edges (lines between nodes, marked green for shared edges and black individual ones.
- faces (entities built from edges, similar to polygons in geometry)
- relation (define relationships between topological elements)

When should you use topological data structures?

Topological structures are especially useful in certain cases. The example below demonstrates such a case. The hungary database contains a county table with data suited for a map scale of 1:300,000–500,000. Our goal is to **generalize** these county borders using a **line simplification** algorithm to make the resulting map suitable for a **1:2,000,000** scale. Among various generalization algorithms, the most commonly used is the **Douglas–Peucker** (or Ramer–Douglas–Peucker) algorithm. This recursive algorithm reduces the number of nodes while preserving the vertices that contribute to the line's overall shape.

If you apply a simplification algorithm directly to geometries:

- Every feature is simplified **independently**
- The algorithm **does not account for shared borders**
- This can cause **topological errors**, like **gaps or overlaps** between polygons

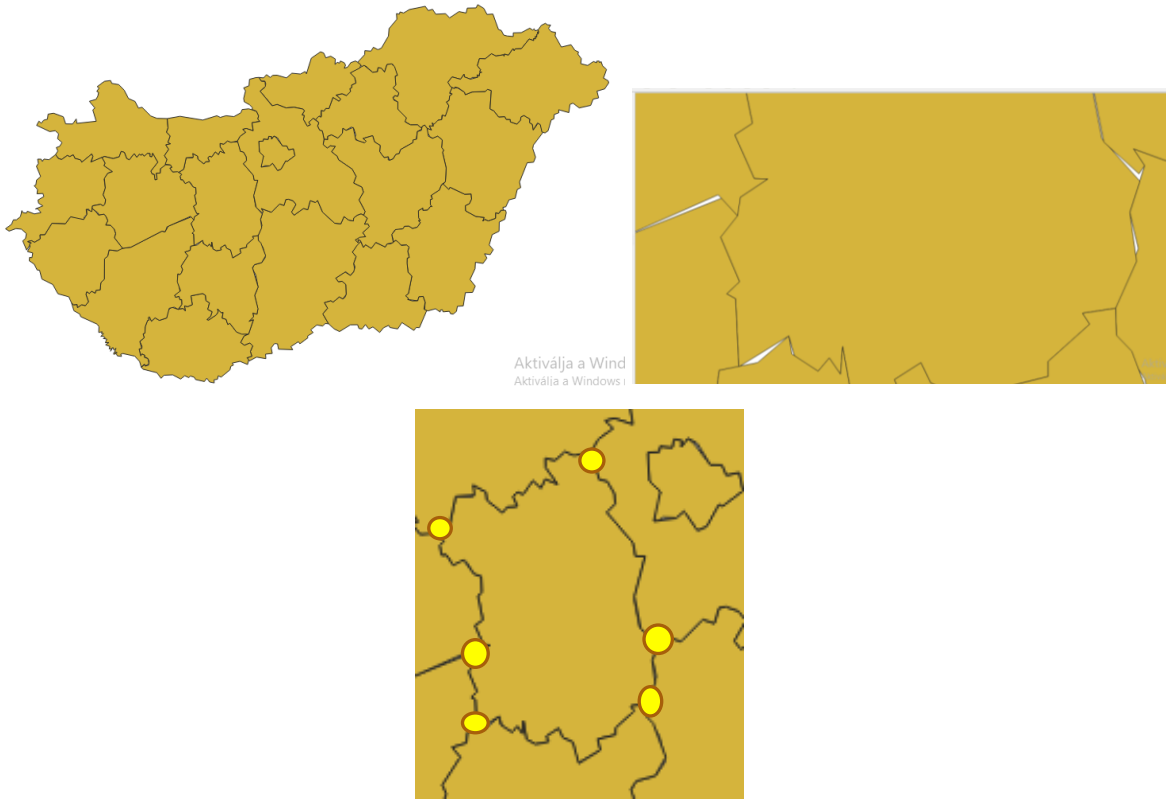
So, **geometry simplification is not suitable** for small-scale maps if shared borders must remain consistent.

Try it yourself:

Even when using ST_SimplifyPreserveTopology, shared borders can still break:

```
select co_name, ST_Simplify(co_geom, 1000) from county
```

```
select co_name, ST_SimplifyPreserveTopology(co_geom, 1000) from county
```



Upper-left: Original layer: counties of Hungary. Upper-right: Topological errors of the feature simplification. Bottom-center image: yellow dots show the nodes of topology

The solution is to introduce the topological data structure. In this case, you will simplify the edges (single or shared shared edges are stored only once, along with which **faces** they belong to. The result maintains **topological correctness.**

How to do this with PostGIS?

Outline

- Create an empty topology table and populate it with the transformed geometry from county table
- Create one more empty topology table. Call the Douglas–Peucker algorithm on the previous table, and upload the simplified topology to this table.
- Convert the simplified topolpgy back to geometry.

Douglas–Peucker algorithm will be described at the end of this lesson. **Now, let's do the task step by step.**

Add a topology Schema and the postgis.topology extension to the database.

```
create schema IF NOT EXISTS topology;

create extension IF NOT EXISTS postgis_topology SCHEMA topology;
```

Copy the original county table and add a new field for a simplified geometry.

```
create table new_county as (select co_id, co_name, co_geom from county);

alter table new_county add column simplified_county geometry(POLYGON, 23700);
```

Create an empty topology table.

```
select topology.CreateTopology('topo1',23700)
```

Populate the empty topology table with the transformed data from county table.

```
select ST_CreateTopoGeo('topo1', ST_Collect(co_geom)) from county
```

Create another empty topology for the simplified result.

```
select topology.CreateTopology('topo2',23700);
```

Simplify the edges with Douglas–Peucker algorithm and populate the second table.

```
select ST_CreateTopoGeo('topo2', geom) from (select
ST_Collect(ST_SimplifyPreserveTopology(geom, 1000)) as geom from topo1.edge_data)
As foo;
```

Reconstruct the simplified geometry from the simplified topology. Use `with` clause.

```
with simple_face As (select ST_GetFaceGeometry('topo2',face_id) as geom from
topo2.face where face_id>0) update new_county d set simplified_county=sf.geom from
simple_face sf where ST_Intersects(d.co_geom, sf.geom) and
ST_Area(ST_Intersection(sf.geom,d.co_geom))/ST_Area(sf.geom)>0.5;
```

Detailed explanation: an Update statement uses the with clausule to reconstruct the polygons from faces. `ST_GetFaceGeometry` returns the polygon in the given topology with the specified face id. Builds the polygon from the edges making up the face.

```
Select ST_GetFaceGeometry('topo2',face_id) as geom from topo2.face where face_id>0
```

The `simplified_county` table is populated from the result of select statement. The new – simplified – geometry is inherited from

```
update new_county d set simplified_county=sf.geom from simple_face sf where
ST_Intersects(d.co_geom, sf.geom) and
ST_Area(ST_Intersection(sf.geom,d.co_geom))/ST_Area(sf.geom)>0.5;
```

To restore the geometry, the original geometry of the new county and the polygons reconstructed from the faces must intersect, and the ratio of the area of their intersection to the area of the reconstructed polygons must be greater than 0.5. This condition is included because we did not store descriptive data for the topologies. Therefore, I identify the simplified county polygons based on which original polygon

they significantly overlap with (after simplification, there may be minor overlaps with neighboring counties, which must be excluded).

Other funtions in the task:

`ST_Collect()` → create geometry collection from other data types

`ST_CreateTopoGeo()` → convert topology from geometry

`ST_GetFaceGeometry()`→ create faces from edges with similar id-s

How **WITH** clause works:

A **CTE** (Common Table Expression) is a temporary, named result set in SQL that you define at the start of a query using the WITH clause. You can use it just like a table or subquery in the main part of your SQL statement. In PostgreSQL, the WITH clause is used to define **Common Table Expressions (CTEs)**. A CTE is a temporary result set that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement. It helps structure complex queries and improves readability.

Syntax:

```
with temporaryTable (averageValue) as
(select Avg(Attr1)
from table)
select Attr1
from Table, temporaryTable
where table.Attr1 > temporaryTable.averageValue;
```

Lets see an example from *library* database. Which salaries are bigger than the average salary?

```
with libr(avgsalary) as
    (select Avg(li_salary)
    from librarian)
    select li_job from librarian, libr
    where librarian.li_salary > libr.avgsalary;
```

One more example: the average salary in which position is bigger than the average salary of librarian table?

```
with  libr(avgsalary) as
(select Avg(li_salary)
    from librarian),
   job(avgsalary, job) as (select Avg(li_salary), li_job from librarian group by
li_job)
    select job.job from job,libr where job.avgsalary>libr.avgsalary;
```

Other tutorials about PostGIS Topology:

http://strk.kbt.io/projects/postgis/Paris2011_TopologyWithPostGIS_2_0.pdf

https://trac.osgeo.org/postgis/wiki/UsersWikiPostgisTopology

What is `ST_Simplify` and `ST_SimplifyPreserveTopology`?

The most geoinformatics software offers algorithm(s) for line generalization. The most popular algorithm is the Douglas–Peucker, which can be found in every software with different naming. This algorithm was find out to reduce the number of nodes of a line, but it can be applied for polygons as well. In this case, it simplify the ring (exterior and interior) of the polygon. If you simplify a polygon layer, be careful: do not simplify the features one by one, because you will get topological errors (gaps or overlaps). Use a topology data structure instead as in this chapter you have seen.

How it works (step-by-step):

Input:

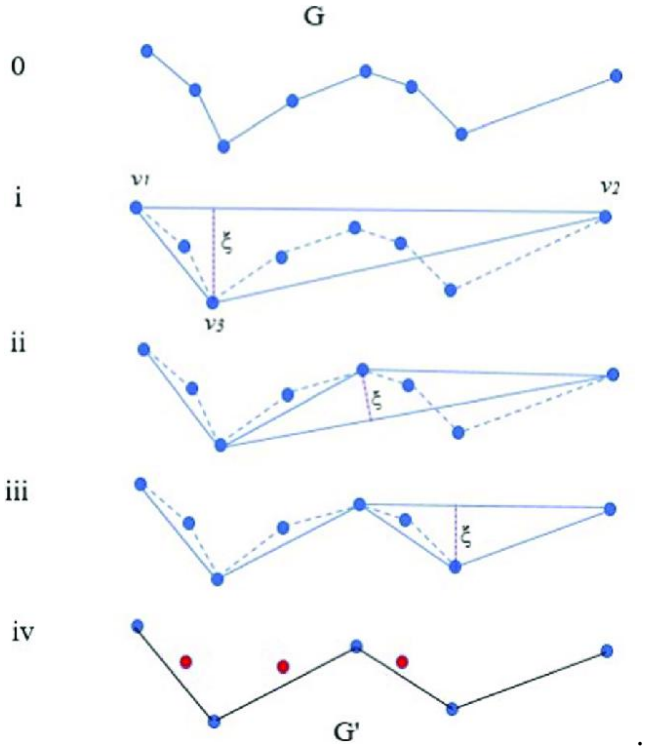
- A curve made up of a series of ordered points.
- A tolerance (ε) — the maximum allowed deviation between the original and simplified curve.

Basic Steps:

- Start with the first and last point of the polyline. These will always be part of the simplified result.
- Find the point farthest from the line segment between the first and last points. This is the point that causes the most "bending."
- If this distance is greater than ε:
    - Keep this point.
    - Recursively apply the same process to the two segments:
    - From the first point to the farthest point.
    - From the farthest point to the last point.
- If the maximum distance ≤ ε, remove all intermediate points — the entire curve segment can be approximated by the straight line.

Recursive Nature:

The algorithm uses divide and conquer, splitting the polyline wherever the maximum deviation is too large, until all segments are within tolerance.



Source of image: Wikipédia. (https://it.wikipedia.org/wiki/Algoritmo_Ramer-Douglas-Peucker)

Chapter 10: Appendix – databases and files in these lecture notes

The CSV (Comma Separated Values) files stores data in a table-like structure as a plain text. Each line represents a database record, and the columns are separated by a delimiter (such as a comma, semicolon, tab, etc.).

europa.csv

This file contains the names, area, polulation, population density and the capital city of European countries.

SQL

.sql scripts are used for backing up and the migrating databases. They contains the whole table structure and data as well. Be careful, MySQL and PostgreSQL scripts are incompatible with each other.

library.sql

It contains the *library* database. The tables are *catalogue, publication, publication_group, borrowing, borrower, librarian, department.*

hungary.sql

Table structure of *hungary* database. EPSG/SRID 23700 for all tables, except *castles*.

th tables of dataset hungary		**Field name**	**Data type**	
	county	co_id	serial4	Primary key
		co_name	varchar(254)	Name of county (=vármegye)
		co_county_code	int4	A custom code for the county
		co_geom	geometry (MultiPolygon)	Geometry field
	district	d_id	serial4	Primary key
		d_name	varchar(254)	District name (járás)
		d_county_code	int4	the code of the county, which contains the district
		d_district_code	int4	A custom code for district
		d_geom	geometry (MultiPolygon)	Geometry field
	settlement	s_id	serial4	Primary key
		s_name	varchar(254)	Name of the settlement
		s_name_de	varchar(254)	german name
		s_jaras_kod	int4	the code of the district , which contains the settlement
		s_geom	geometry (MultiPolygon)	Geometry field

	settlement_centroid	sc_id	int4	Primary key
		sc_name	varchar(254)	Name of the settlement
		sc_geom	geometry	Geometry field (the centorid of built up areas)
	builtup_area	ba_id	int4	Primary key
		ba_name	varchar(254)	Name of the settlement
		ba_geom	geometry (MultiPolygon)	Geometry field (only the built up areas)
	motorway	mo_id	int4	Primary key
		mo _wayid	varchar(254)	OSM identifier
		mo _hu_ed_dire	varchar(254)	direction of the motorway from Budapest
		mo _alt_name	varchar(254)	Old name of the motorway (in the most cases is NULL)
		mo_alt_bridge	varchar(254)	is it a bridge?
		mo _highway	varchar(254)	type: motorway
		mo _int_ref	varchar(254)	Other number of the road
		mo _lanes	varchar(254)	Number of lanes
		mo _maxspeed	varchar(254)	Maximum speed
		mo _oneway	varchar(254)	Is it a one way road?
		mo _ref	varchar(254)	**Number of the road**
		mo _surface	varchar(254)	covering material
		mo _toll	varchar(254)	is there a toll?
		mo _geom	geometry(multilinestring)	Geometry field
	highway	hi_id	int4	Primary key
		hi _wayid	varchar(254)	OSM identifier
		hi _hu_ed_dire	varchar(254)	direction of the road from Budapest
		hi _bridge	varchar(254)	is it a bridge?
		hi _foot	varchar(254)	can people walk on road?
		hi _highway	varchar(254)	road type
		hi _horse	varchar(254)	Can people use horse carriage?
		hi _int_ref	varchar(254)	Other road numbering
		hi _maxspeed	varchar(254)	Maximum speed

		hi _name	varchar(254)	road name
		hi _oneway	varchar(254)	One way?
		hi _ref	varchar(254)	**Number of the road**
		hi _surface	varchar(254)	covering material
		hi _toll	varchar(254)	is there a toll?
		hi _tunnel	varchar(254)	is it a tunnel?
		hi _geom	geometry(multilinestring)	Geometry field
	primary highway	ph_id	int4	Primary key
		ph _wayid	varchar(254)	OSM identifier
		ph _hu_ed_dire	varchar(254)	Road direction from Budapest
		ph_bicycle	varchar(254)	Can peole ride a bicycle?
		ph_bridge	varchar(254)	Is it a bridge?
		ph_foot	varchar(254)	Can people walk there?
		ph_highway	varchar(254)	road type
		ph_horse	varchar(254)	can people ride a horse carriage?
		ph_int_ref	varchar(254)	other road numbering
		ph_maxspeed	varchar(254)	maximum speed
		ph_name	varchar(254)	Name of the road section
		ph_oneway	varchar(254)	One way road?
		ph_ref	varchar(254)	**Number of the primary road**
		ph_surface	varchar(254)	covering material
		ph_toll	varchar(254)	Is there a toll?
		ph_tunnel	varchar(254)	Is it a tunnel?
		ph_geom	geometry(multilinestring)	Geometry field
	secondary highway	sh_id	int4	Primary key
		sh_wayid	varchar(254)	OSM identifier
		sh_bicycle	varchar(254)	can people ride a bicycle?
		sh_bridge	varchar(254)	is it a bridge?
		sh_foot	varchar(254)	can people walk there?
		sh_highway	varchar(254)	road type
		sh_horse	varchar(254)	can people ride a horse carriage?

		sh_maxspeed	varchar(254)	maximum speed
		sh_name	varchar(254)	name of the road section
		sh_oneway	varchar(254)	one way?
		sh_ref	varchar(254)	**Number of the secondary road**
		sh_surface	varchar(254)	covering material
		sh_toll	varchar(254)	is there a toll?
		sh_tunnel	varchar(254)	is there a tunnel?
		sh_geom	geometry(multilinestring)	Geometry field
	tertiary highway	th_id	int4	Primary key
		th_wayid	varchar(254)	OSM identifier
		th_bicycle	varchar(254)	can people ride a bicycle?
		th_bridge	varchar(254)	is it a bridge?
		th_foot	varchar(254)	can people walk there?
		th_highway	varchar(254)	road type
		th_horse	varchar(254)	can people ride a horse carriage?
		th_maxspeed	varchar(254)	maximum speed
		th_name	varchar(254)	name of the road section
		th_oneway	varchar(254)	one way?
		th_ref	varchar(254)	**Number of the tertiary road**
		th_surface	varchar(254)	coverint material
		th_toll	varchar(254)	is there a toll?
		th_tunnel	varchar(254)	is it a tunnel?
		th_geom	geometry(multilinestring)	Geometry field
	railway	r_id	serial4	Primary key
		r_name	varchar(48)	Nam of the section, if there is
		r_type	varchar(16)	type
		r_linenumber	varchar(10)	Number of the line
		r_status	varchar(15)	Current status of the section.
		r_agency	varchar(5)	Agency name
		r_geom	geometry(multilinestring)	Geometry field

	river_polygon	rp_id	int4	Primary key
		rp_name	varchar(254)	Name
		rp_geom	geometry(multipolygon)	Geometry field
	river_line	rl_id	int4	Primary key
		rl_name	varchar(254)	River name
		rl_category	varchar(50)	River category
		rl_geology	varchar(50)	The geological environment of the river
		rl_steepness	varchar(50)	Steepness
		rl_crossing	varchar(6)	Crossing identifier
		rl_desc	varchar(250)	Detaild description
		rl_geom	geometry(multilinestring)	Geometry field
	river	ri_id	int4	Primary key
		ri_name	varchar(254)	Primary name of the river
		ri_name_hu	varchar(254)	Hungarian name
		ri_geom	geometry(multilinestring)	Geometry field
	creek	cr_id	int4	Primary key
		cr_name	varchar(254)	Name
		cr_geom	geometry(multilinestring)	Geometry field
	lake	la_id	serial4	Primary key
		la_name	varchar(100)	Lak name
		la_category	varchar(30)	Lake category
		la_elevation	numeric	Elevation above sea level
		la_general	varchar(50)	general environment of the lake
		la_depth	numeric	lake depth
		la_desc	varchar(150)	description
		la_geom	geometry(multipolygon)	Geometry field
	hungary_border	mo_id	int4	Primary key
		mo_name	varchar(254)	Name of the county
		mo_geom	geometry(multipolygon)	Geometry field
	geobox	gid	serial(4)	Primary key
		gl_id	float(8)	other id
		gl_lat_d	float(8)	latitude in degree

		gl_lat_m	numeric	latitude in minute
		gl_lon_d	float(8)	longitude in degree
		gl_lon_m	numeric	longitude in minute
		gl_elevation	numeric	elevation above sea level
		gl_full_name	varchar(254)	name of the geobox
		gl_type	varchar(254)	type of the geobox
		gl_county_country	varchar(254)	Name of the county and country where it can be found
		gl_terep	numeric	difficulty of the road
		gl_waypoint	varchar(254)	short id
		gl_description	varchar(254)	geobox description
		gl_felhaszn	varchar(254)	owner
		gl_email	varchar(254)	owner email
		gl_field_30	numeric	latitude in degrees
		gl_field_31	numeric	longitude in degress
		gl_geom	geometry(point(23700))	Geometry field
	nature_protection	np_id	in(4)	Primary key
		np_name	varchar(254)	Name of the protected area
		np_type	varchar(50)	National park (nemzeti park) or local protected area (tájvédelmi körzet)
		np_geom	geometry(MultiPolygon)	Geometry field
	castles	ca_id	int(4)	Primary key
		ca_name	varchar(254)	Name of the castle
		ca_geom	geometry(geometry,4326)	**Geometry field (EPSG: 4326)**
	spatial_ref_sys			**Built in PostGIS table, it stores the projections and spatial reference system.**