https://www.halvorsen.blog

PostgreSQL

Hans-Petter Halvorsen

Contents

PostgreSQL

Introduction

Hans-Petter Halvorsen

Table of Contents

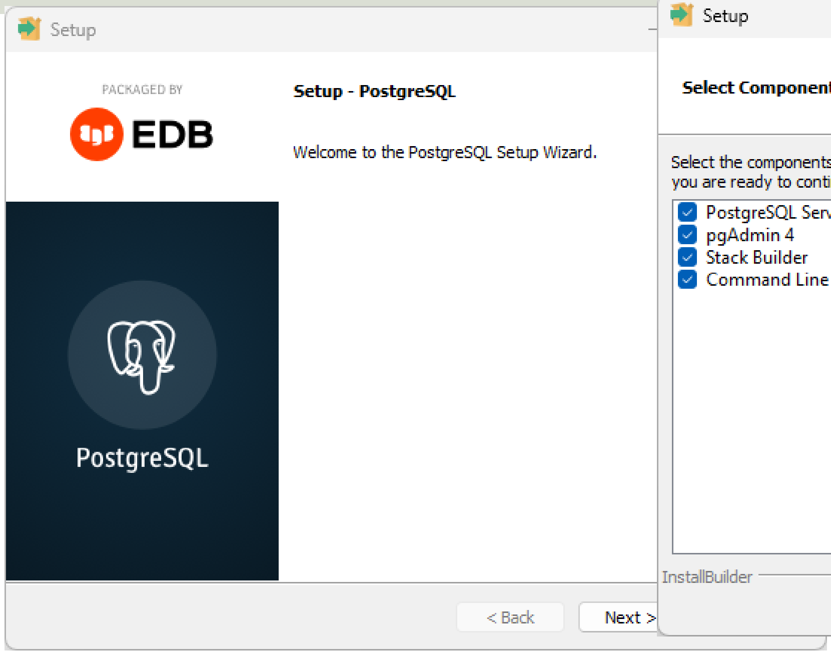
Introduction

- PostgreSQL is an open-source object-relational database system.
- Many other SQL databases exists like SQL Server, MySQL, MariaDB etc.
- We will focus on PostgreSQL in this Tutorial.

PostgreSQL

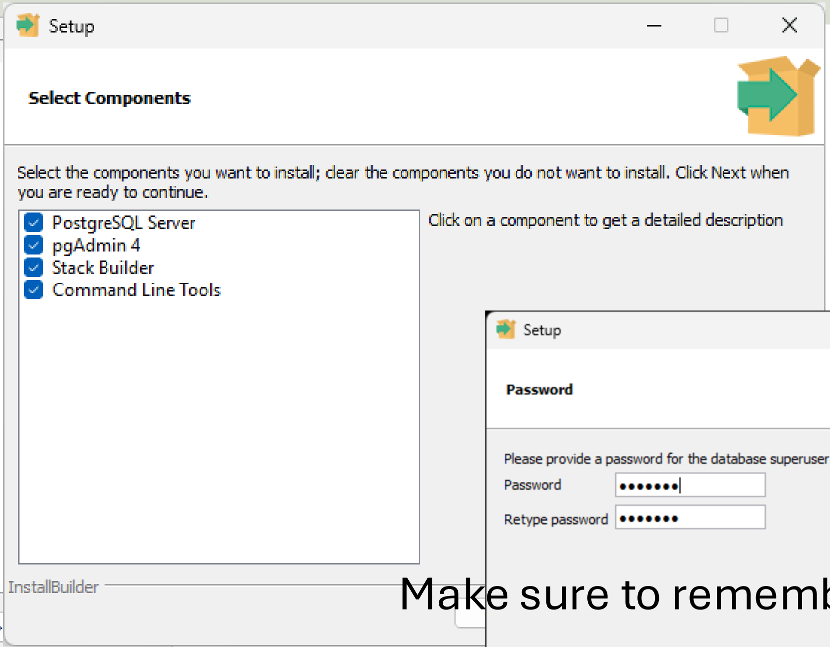
- PostgreSQL is an open-source object-relational database system.
- PostgreSQL exists for Windows, macOS and Linux.
- Homepage: https://www.postgresql.org
- EnterpriseDB (EDB) is the company that is one of the largest contributor to PostgreSQL and responsible for the installer.
- EDB offer paid services for enterprises, but PostgreSQL itself is free.
- ERD Download Page: https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

Installation



Setup - PostgreSQL

Welcome to the PostgreSQL Setup Wizard.

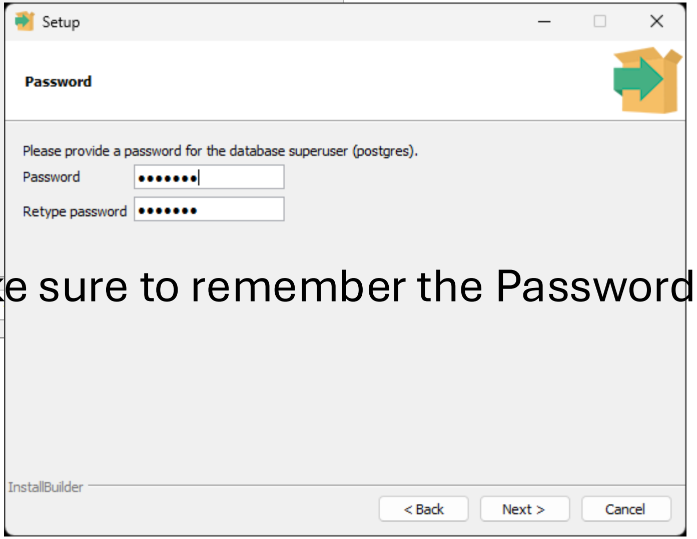


Select Components

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] PostgreSQL Server
- [✓] pgAdmin 4
- [✓] Stack Builder
- [✓] Command Line Tools

Click on a component to get a detailed description



Password

Please provide a password for the database superuser (postgres).

Password •••••••

Retype password •••••••

Make sure to remember the Password!

I just use the default installation setup. In addition, you need to create a password for the database superuser that you need to remember for later.

https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

pgAdmin

- pgAdmin is graphical tool for managing your PostgreSQL database.
- pgAdmin is part of the installer from EDB.
- If you prefer, you can also use “SQL Shell (psql)”, which is a terminal based program where you can write and execute SQL syntax in the command-line terminal.

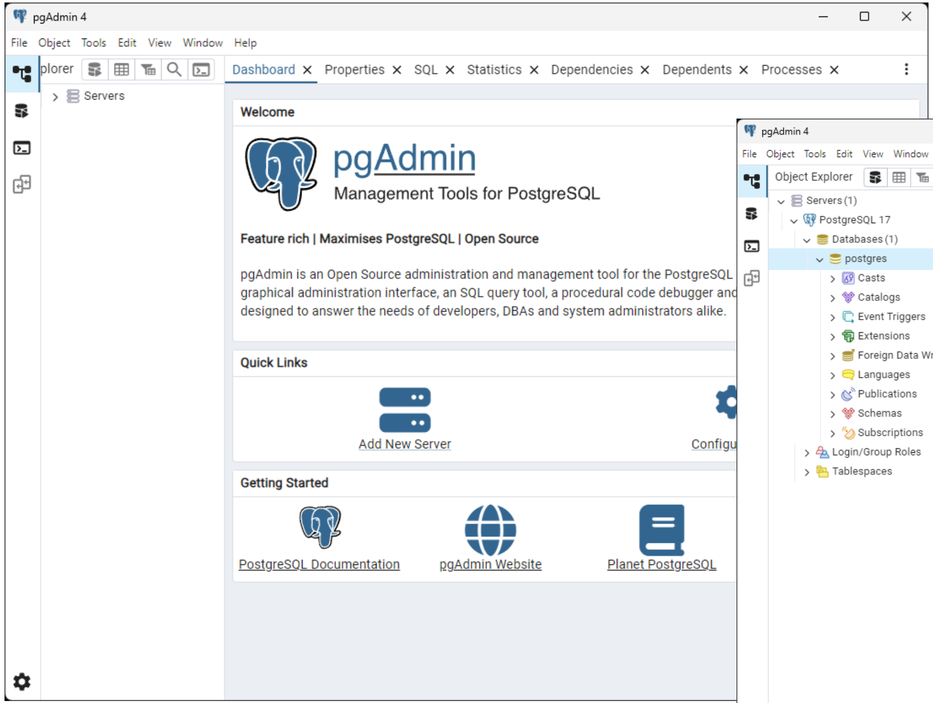
PostgreSQL

Getting Started with PostgreSQL

Hans-Petter Halvorsen

Table of Contents

pgAdmin

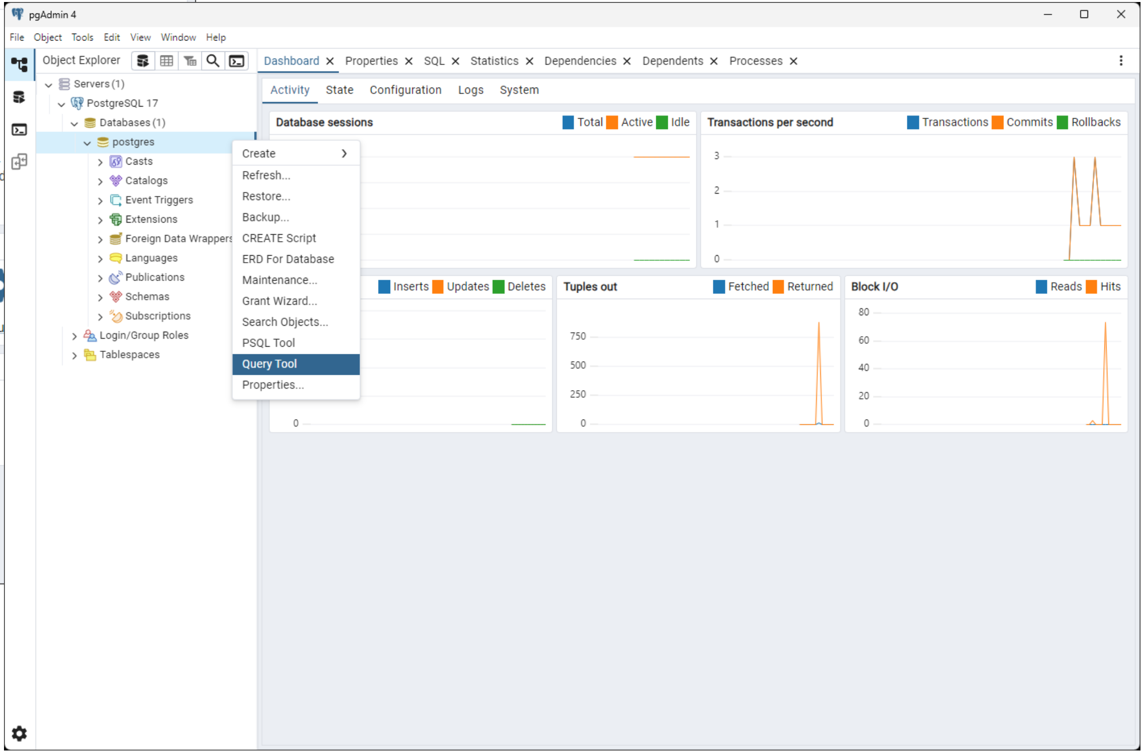

Welcome

pgAdmin

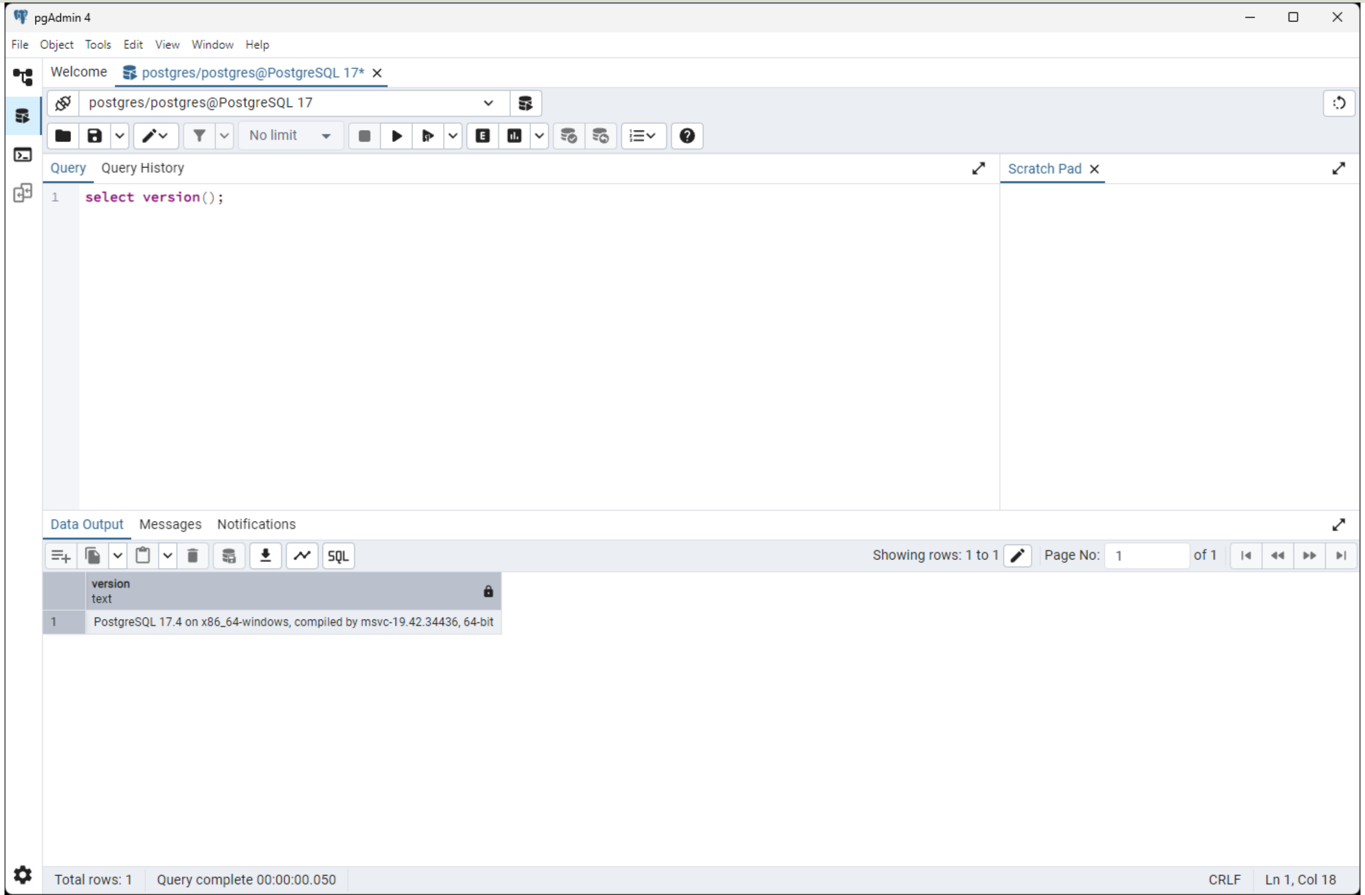
Management Tools for PostgreSQL

Feature rich | Maximises PostgreSQL | Open Source

pgAdmin is an Open Source administration and management tool for the PostgreSQL
graphical administration interface, an SQL query tool, a procedural code debugger and
designed to answer the needs of developers, DBAs and system administrators alike.



Query Tool



pgAdmin 4
File Object Tools Edit View Window Help
Welcome
postgres/postgres@PostgreSQL 17*
postgres/postgres@PostgreSQL 17
No limit
Query
Query History
Scratch Pad
Data Output
Messages
Notifications
SQL

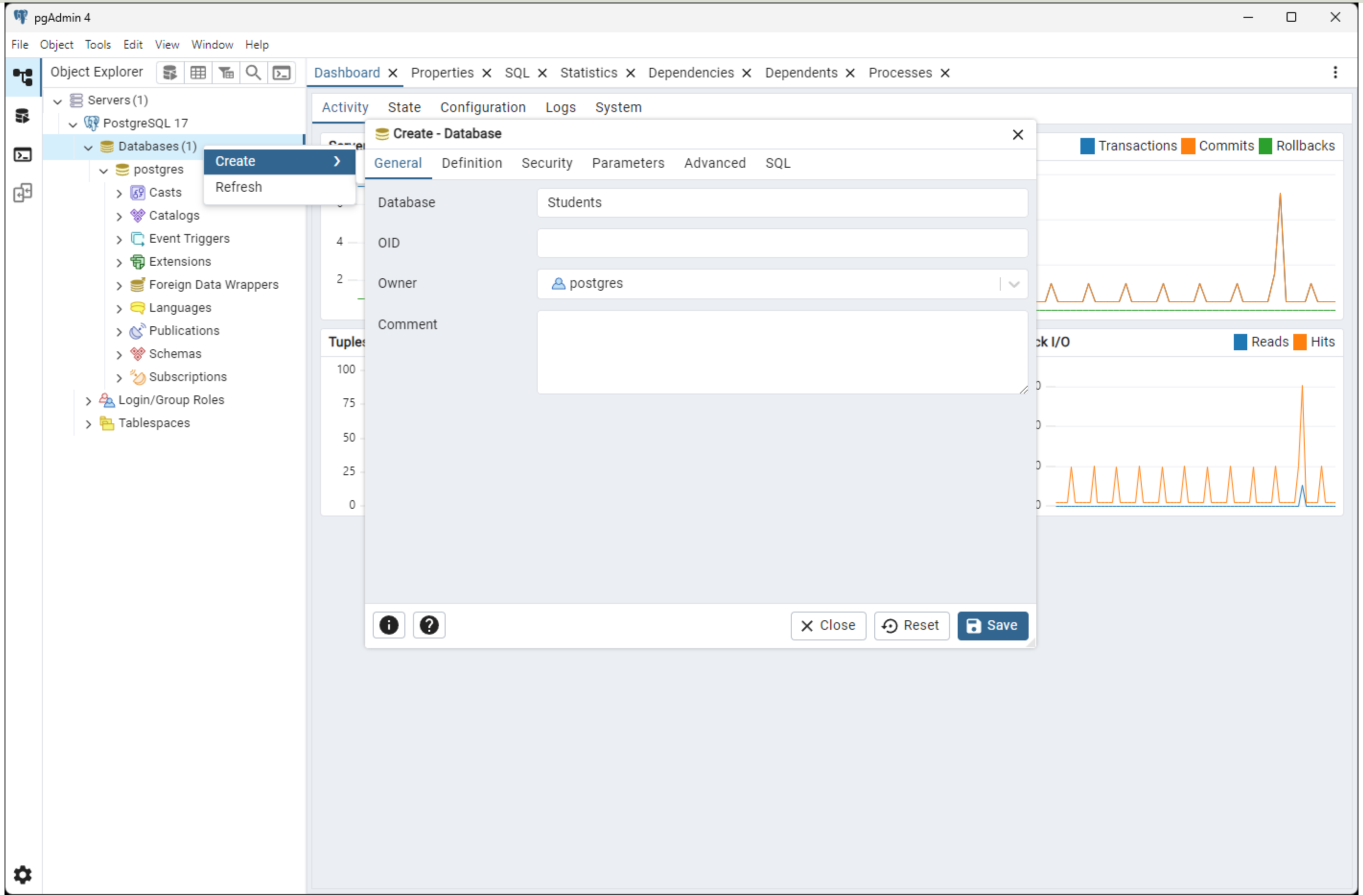
```
select version();
```

Showing rows: 1 to 1 Page No: 1 of 1

	version text
1	PostgreSQL 17.4 on x86_64-windows, compiled by msvc-19.42.34436, 64-bit

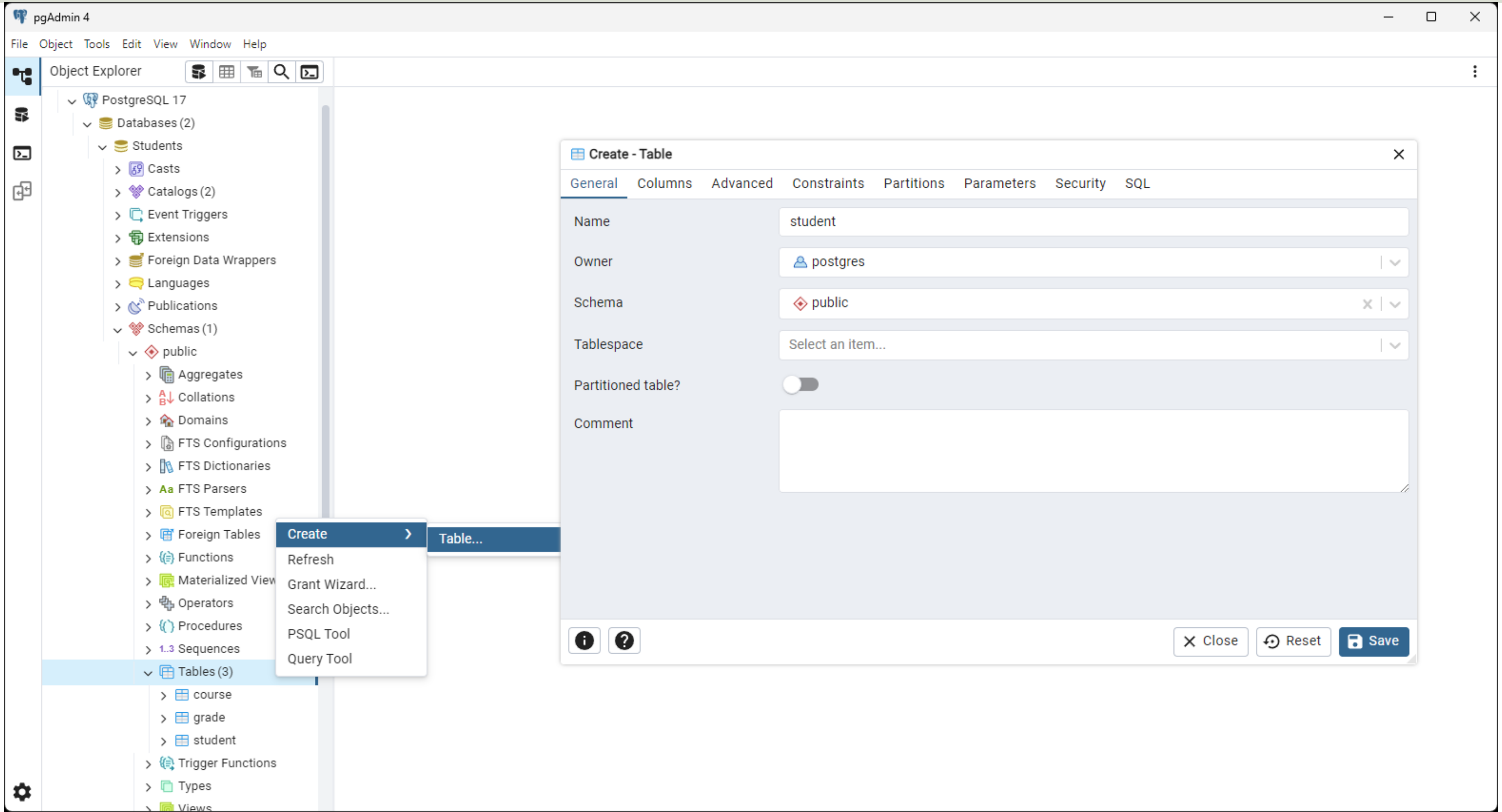
Total rows: 1 Query complete 00:00:00.050 CRLF Ln 1, Col 18

Create new Database



Database	Students
OID	
Owner	postgres
Comment	

Create new Tables

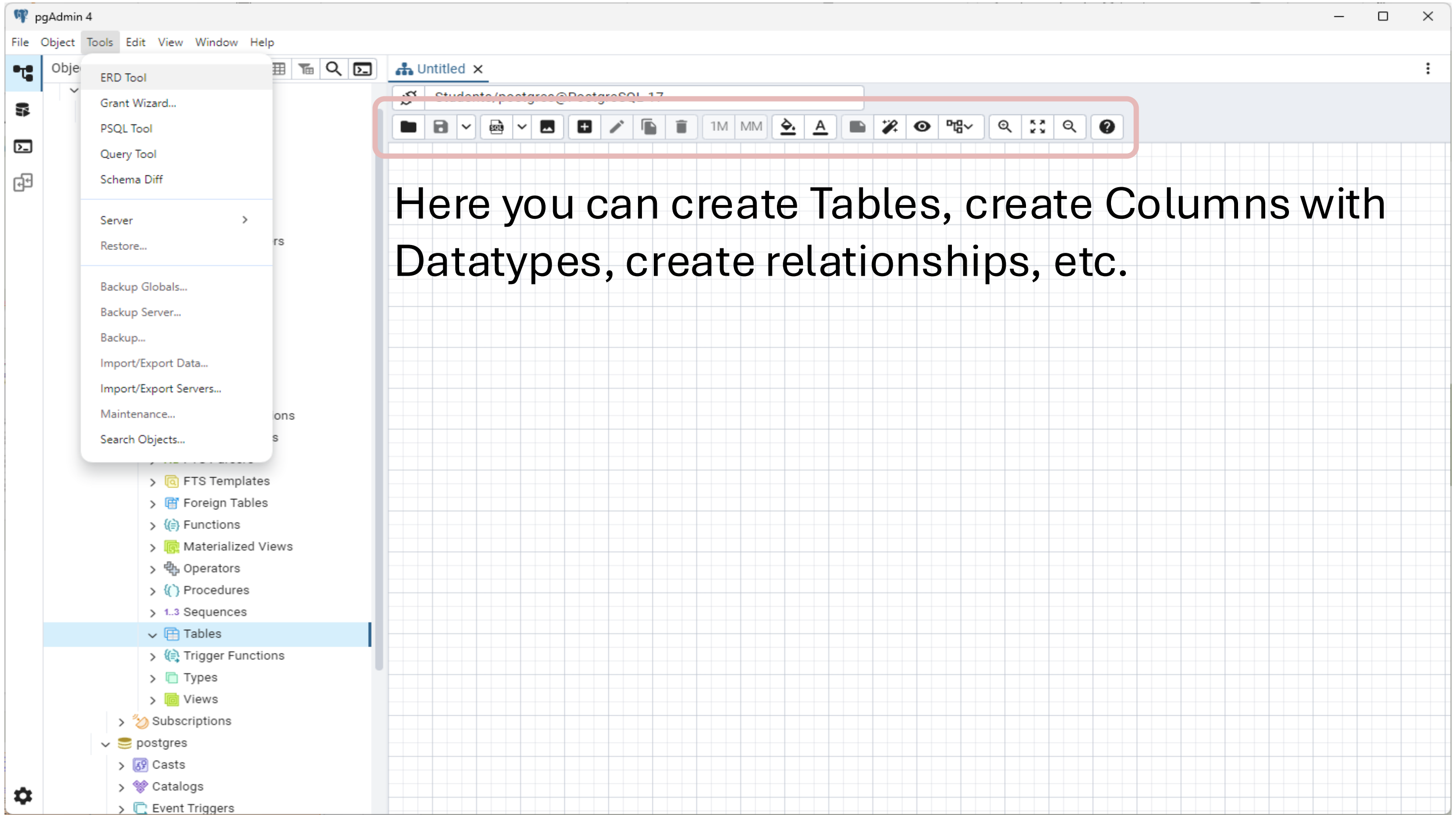


Name	student
Owner	postgres
Schema	public
Tablespace	Select an item...
Partitioned table?	
Comment	

ERD Tool in pgAdmin

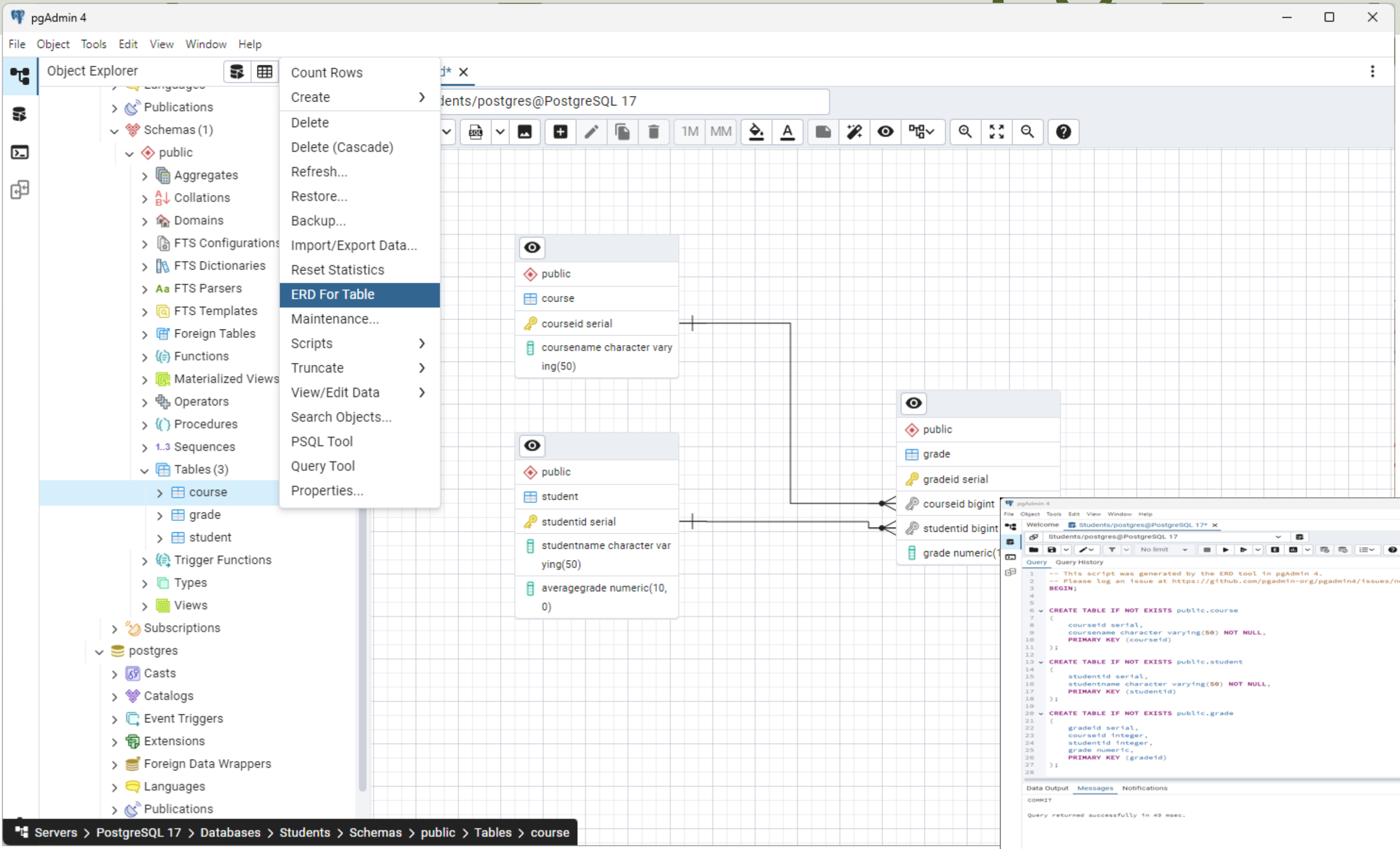
We can also use the ERD tool for creating the Tables from scratch

Entity Relationship Diagram (ERD)

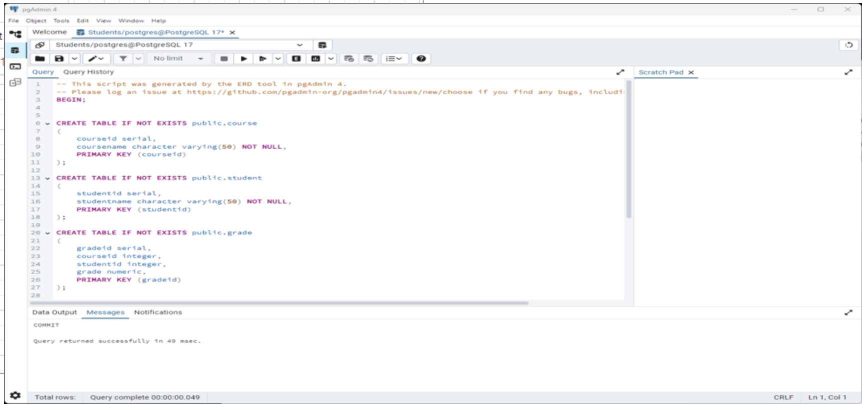


Here you can create Tables, create Columns with Datatypes, create relationships, etc.

ERD Tool in pgAdmin



You can then generate a SQL Script:


pgAdmin 4
File Object Tools Edit View Window Help
Welcome
Students/postgres@PostgreSQL 17*
Students/postgres@PostgreSQL 17
No limit
Query
Query History
Scratch Pad

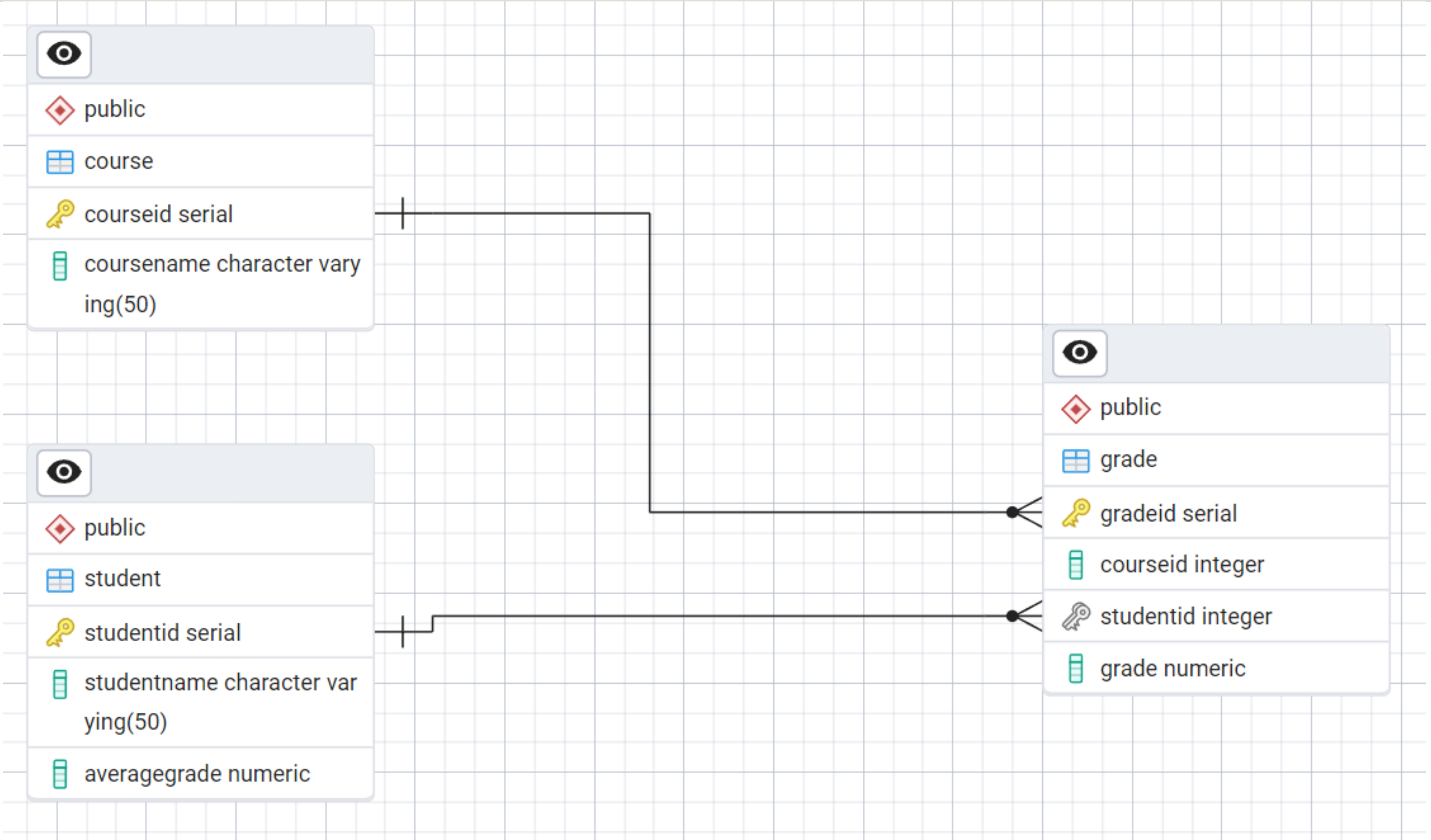
```
-- This script was generated by the ERD tool in pgAdmin 4.
-- Please log an issue at https://github.com/pgadmin-org/pgadmin4/issues/new/choose if you find any bugs, includi
BEGIN;


CREATE TABLE IF NOT EXISTS public.course
(
    courseid serial,
    coursename character varying(50) NOT NULL,
    PRIMARY KEY (courseid)
);

CREATE TABLE IF NOT EXISTS public.student
(
    studentid serial,
    studentname character varying(50) NOT NULL,
    PRIMARY KEY (studentid)
);

CREATE TABLE IF NOT EXISTS public.grade
(
    gradeid serial,
    courseid integer,
    studentid integer,
    grade numeric,
    PRIMARY KEY (gradeid)
);

```

Data Output Messages Notifications

```
COMMIT

Query returned successfully in 49 msec.
```

Total rows: Query complete 00:00:00.049 CRLF Ln 1, Col 1

ERD Diagram


public
course
courseid serial
coursename character varying(50)
public
student
studentid serial
studentname character varying(50)
averagegrade numeric
public
grade
gradeid serial
courseid integer
studentid integer
grade numeric

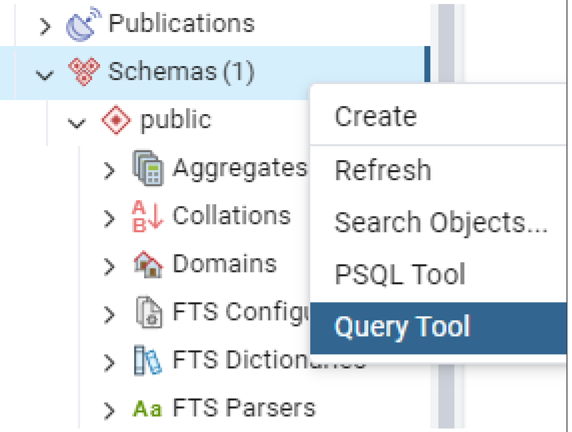
SQL Table Script

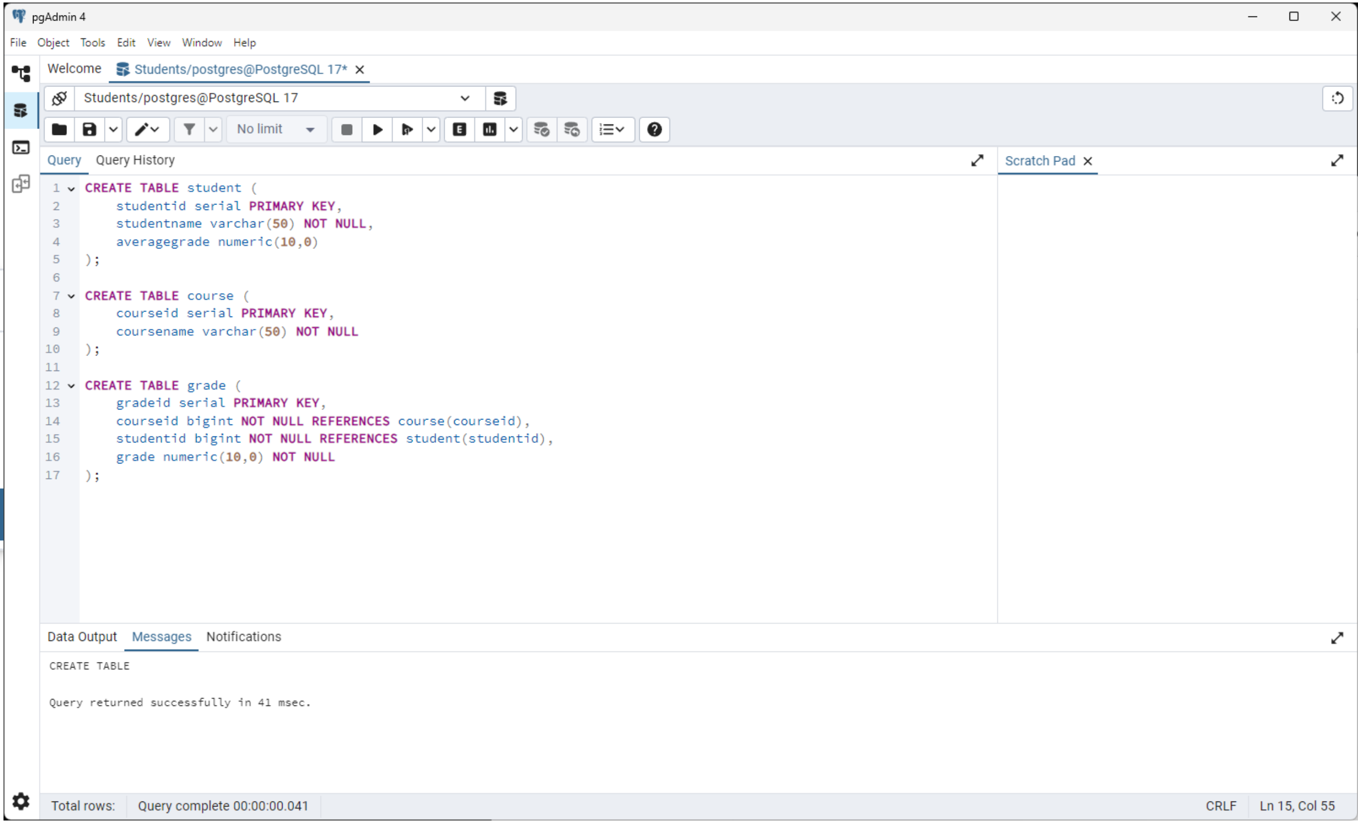
```
CREATE TABLE student (
studentid serial PRIMARY KEY,
studentname varchar(50) NOT NULL,
averagegrade numeric(10,0)
);

CREATE TABLE course (
courseid serial PRIMARY KEY,
coursename varchar(50) NOT NULL
);

CREATE TABLE grade (
gradeid serial PRIMARY KEY,
courseid bigint NOT NULL REFERENCES course(courseid),
studentid bigint NOT NULL REFERENCES student(studentid),
grade numeric(10,0) NOT NULL
);
```

From the ERD Tool, we get a SQL script like this. We can also of course create this script from scratch.

Create Tables from Script

We use the **Query Tool:**




pgAdmin 4
File Object Tools Edit View Window Help
Welcome
Students/postgres@PostgreSQL 17*
Students/postgres@PostgreSQL 17
No limit
Query
Query History
Scratch Pad
Data Output
Messages
Notifications

```
CREATE TABLE student (
    studentid serial PRIMARY KEY,
    studentname varchar(50) NOT NULL,
    averagegrade numeric(10,0)
);

CREATE TABLE course (
    courseid serial PRIMARY KEY,
    coursename varchar(50) NOT NULL
);

CREATE TABLE grade (
    gradeid serial PRIMARY KEY,
    courseid bigint NOT NULL REFERENCES course(courseid),
    studentid bigint NOT NULL REFERENCES student(studentid),
    grade numeric(10,0) NOT NULL
);
```

```
CREATE TABLE

Query returned successfully in 41 msec.
```

Total rows: Query complete 00:00:00.041 CRLF Ln 15, Col 55

PostgreSQL

SQL Queries

Hans-Petter Halvorsen

We have the following main SQL Queries:

- INSERT
- SELECT
- UPDATE
- DELETE

CRUD operations, CRUD = Create (Insert), Read (Select), Update and Delete

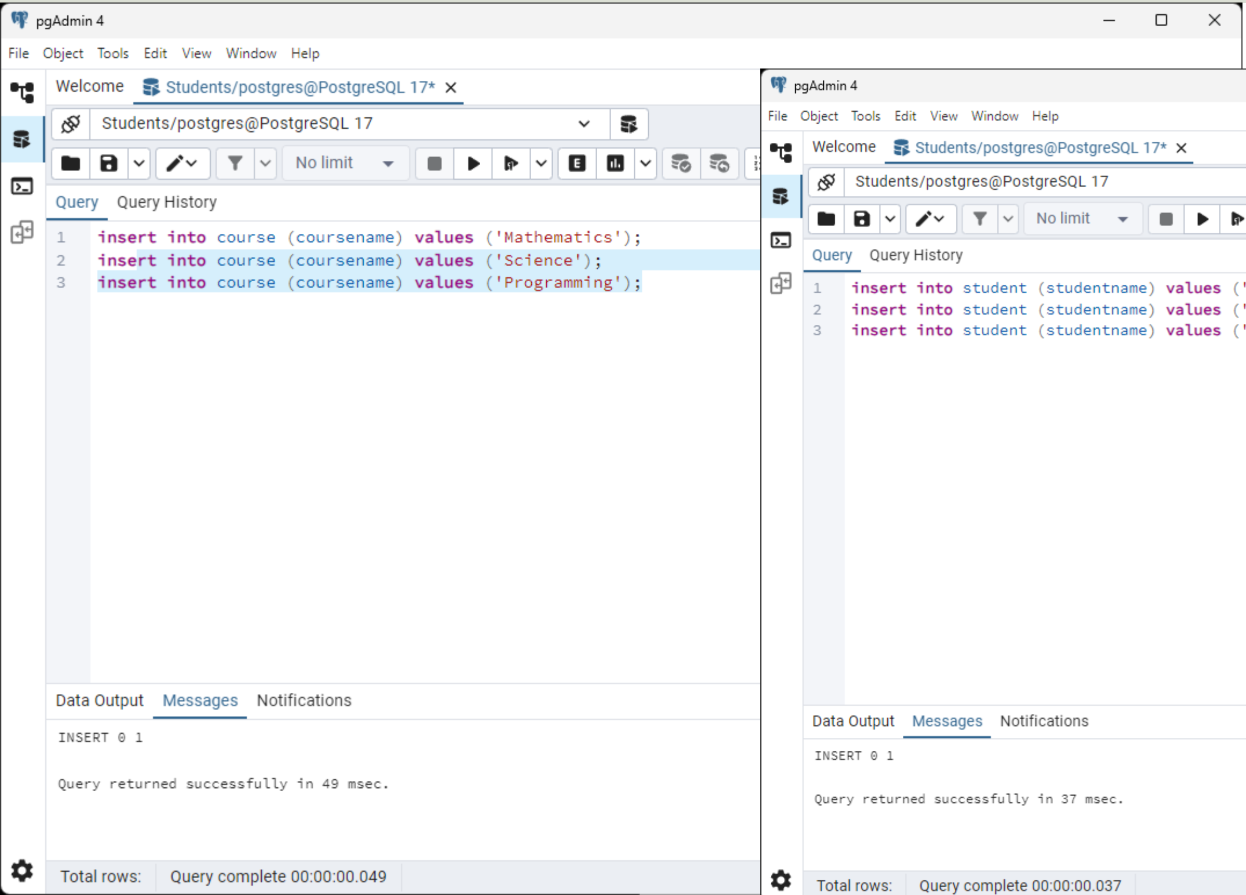
INSERT Courses and Students

Let's create some default data in our tables:

```
insert into course (coursename) values ('Mathematics');
insert into course (coursename) values ('Science');
insert into course (coursename) values ('Programming');
```

```
insert into student (studentname) values ('Elvis Presley');
insert into student (studentname) values ('John Wayne');
insert into student (studentname) values ('John Statham');
```

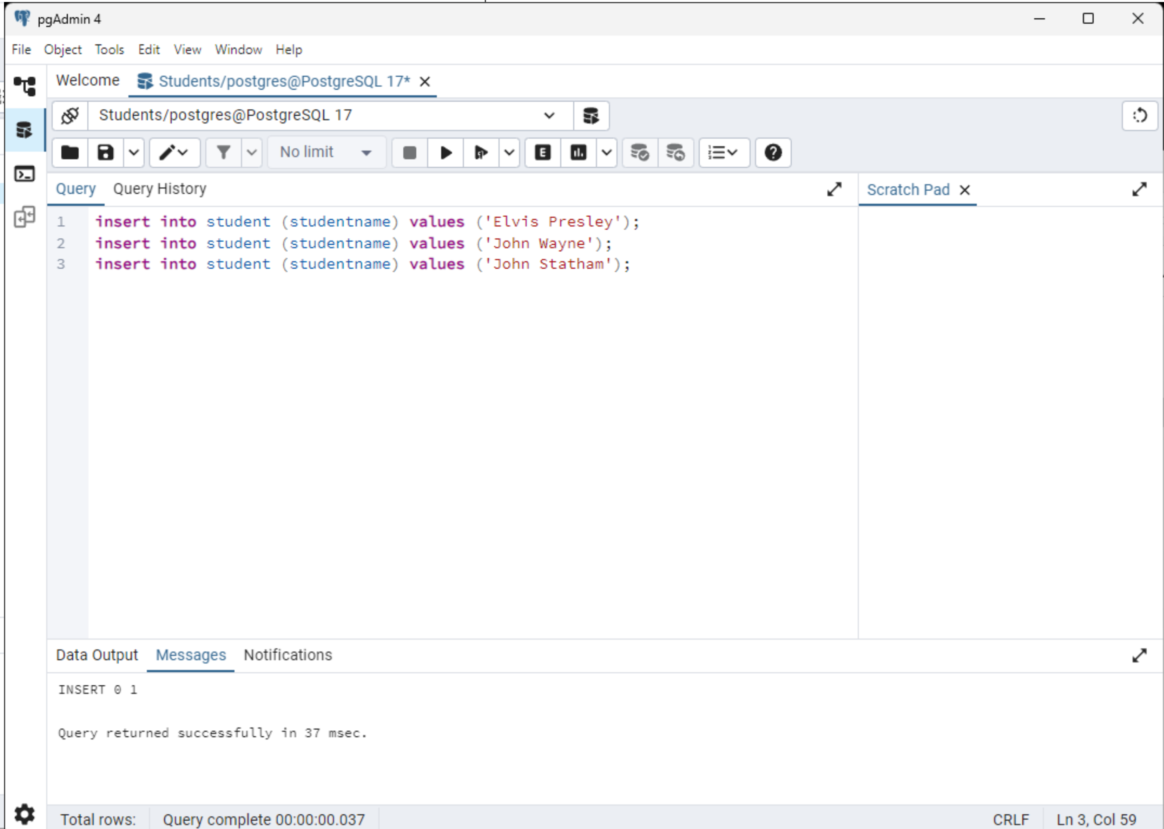
Using the Query Tool


pgAdmin 4
File Object Tools Edit View Window Help
Welcome
Students/postgres@PostgreSQL 17*
Students/postgres@PostgreSQL 17
No limit
Query Query History
Data Output Messages Notifications

```sql
insert into course (coursename) values ('Mathematics');
insert into course (coursename) values ('Science');
insert into course (coursename) values ('Programming');
```

```
INSERT 0 1

Query returned successfully in 49 msec.
```

Total rows: Query complete 00:00:00.049


pgAdmin 4
File Object Tools Edit View Window Help
Welcome
Students/postgres@PostgreSQL 17*
Students/postgres@PostgreSQL 17
No limit
Query Query History
Scratch Pad
Data Output Messages Notifications

```sql
insert into student (studentname) values ('Elvis Presley');
insert into student (studentname) values ('John Wayne');
insert into student (studentname) values ('John Statham');
```

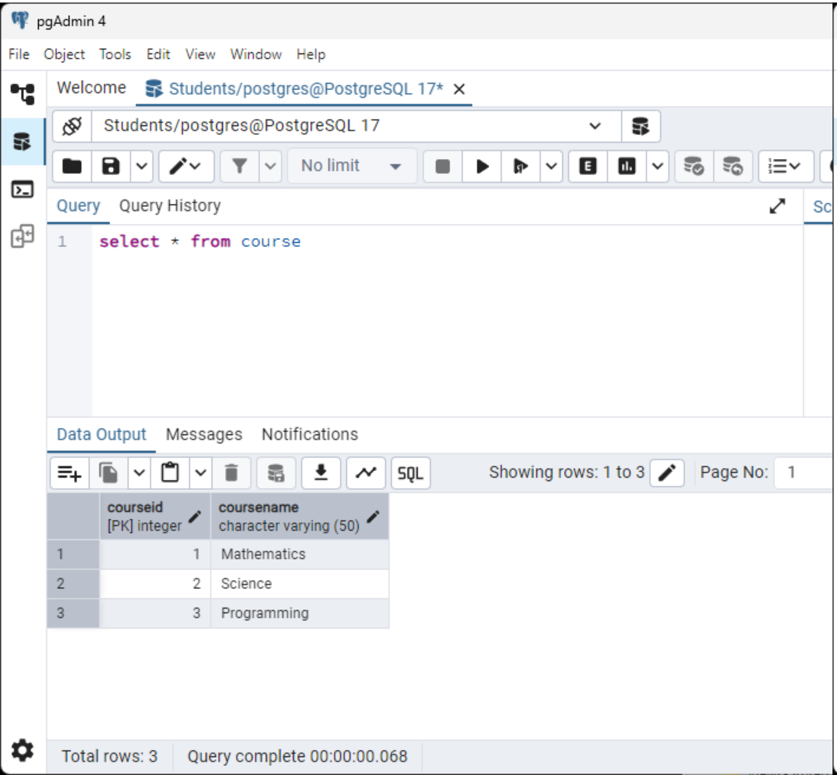
```
INSERT 0 1

Query returned successfully in 37 msec.
```

Total rows: Query complete 00:00:00.037 CRLF Ln 3, Col 59

SELECT

```
select * from course
```

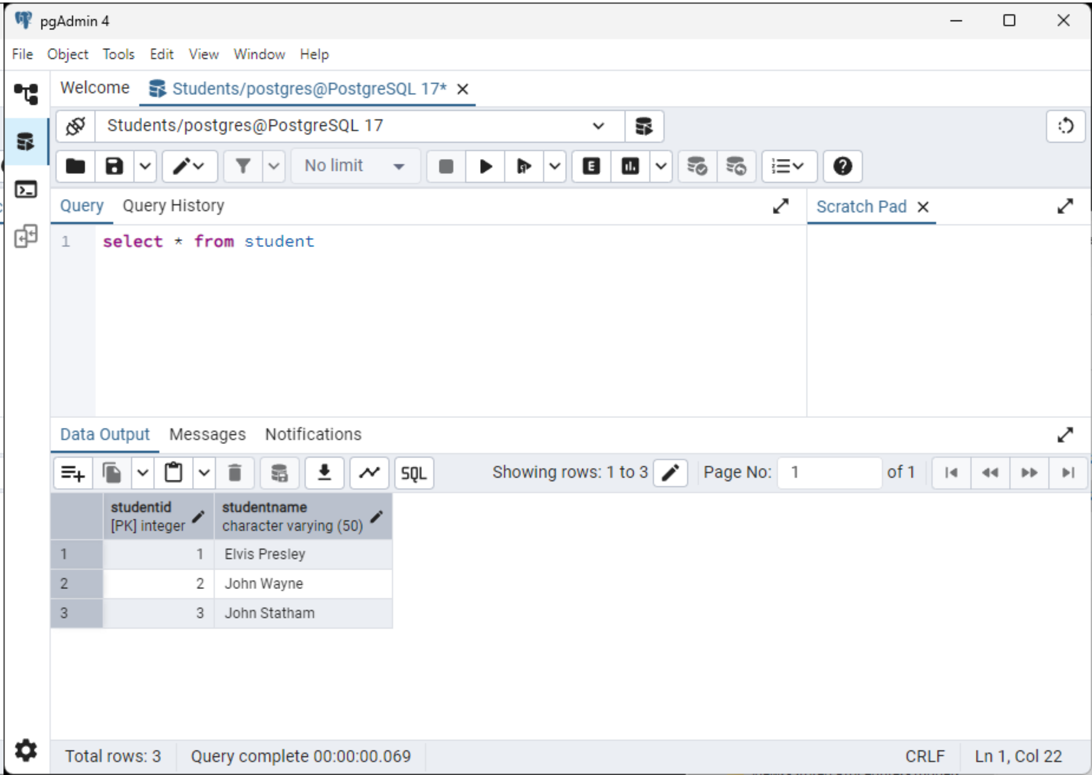
```
select * from student
```



```
select * from course
```

	courseid [PK] integer	coursename character varying (50)
1	1	Mathematics
2	2	Science
3	3	Programming

Total rows: 3 Query complete 00:00:00.068



```
select * from student
```

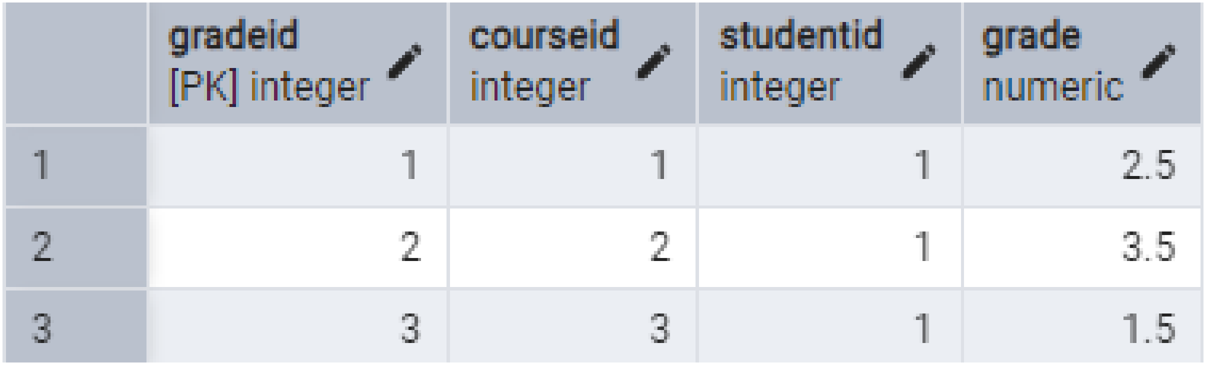
	studentid [PK] integer	studentname character varying (50)
1	1	Elvis Presley
2	2	John Wayne
3	3	John Statham

Total rows: 3 Query complete 00:00:00.069

Insert Grades

```
insert into grade (courseid, studentid, grade) values (1, 1, 2.5);
insert into grade (courseid, studentid, grade) values (2, 1, 3.5);
insert into grade (courseid, studentid, grade) values (3, 1, 1.5);
```

```
select * from grade
```



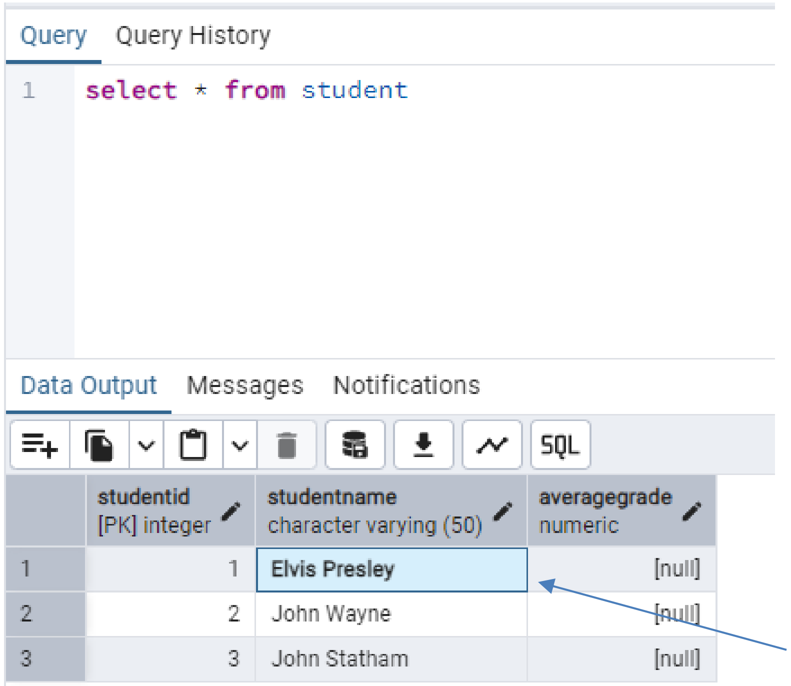
	gradeid [PK] integer	courseid integer	studentid integer	grade numeric
1	1	1	1	2.5
2	2	2	1	3.5
3	3	3	1	1.5

Here student “Elvis Presley” (StudentId=1) gets the following grades in the different courses:

- “Mathematics” (CourseId=1) => Grade = 2.5
- “Science” (CourseId=2) => Grade = 3.5
- “Programming” (CourseId=3) => Grade = 1.5

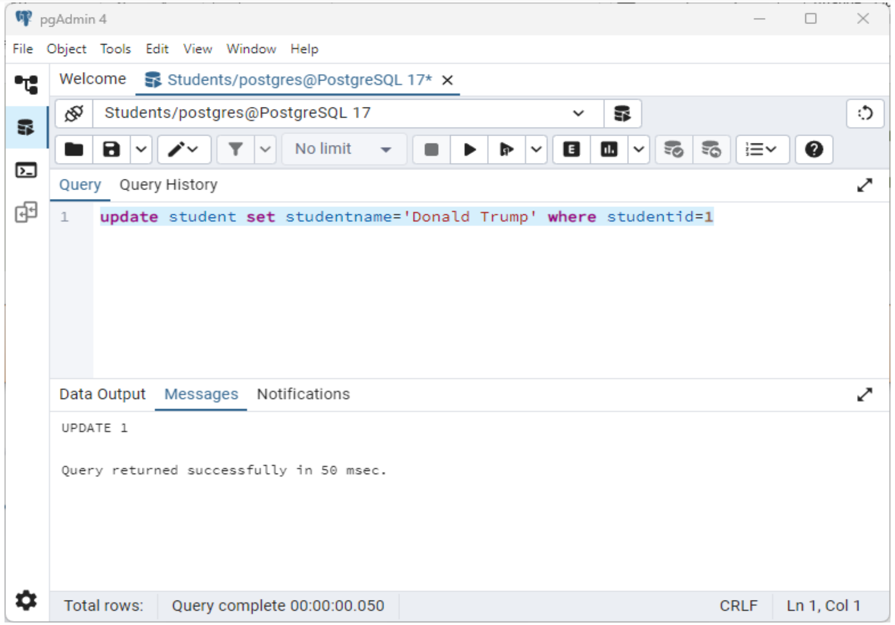
UPDATE

```
update student set studentname = 'Donald Trump' where studentid = 1
```



```
select * from student
```

	studentid [PK] integer	studentname character varying (50)	averagegrade numeric
1	1	Elvis Presley	[null]
2	2	John Wayne	[null]
3	3	John Statham	[null]



```
update student set studentname='Donald Trump' where studentid=1
```

```
UPDATE 1

Query returned successfully in 50 msec.
```

Total rows: Query complete 00:00:00.050 CRLF Ln 1, Col 1

You can also double-click to edit/update the data directly in the “Data Output” panel.

DELETE

```
delete from tablename where column = ...
```

When using DELETE it is important to include a **where** statement, unless you want to delete all the data in that table.

Example:

```
delete from student where studentid = 3
```

This query will only delete the specific student where studentid =3

Or like this:

```
delete from student where studentname = 'John Statham'
```

PostgreSQL

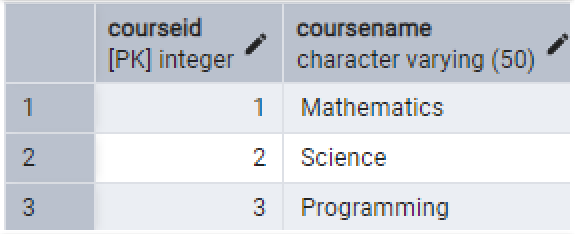
Views

Hans-Petter Halvorsen

Problem Description

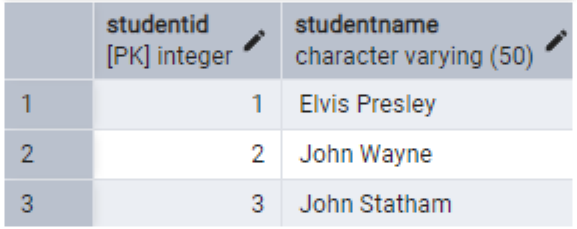
We need to use 3 different SQL queries to get information:

```
select * from course
```



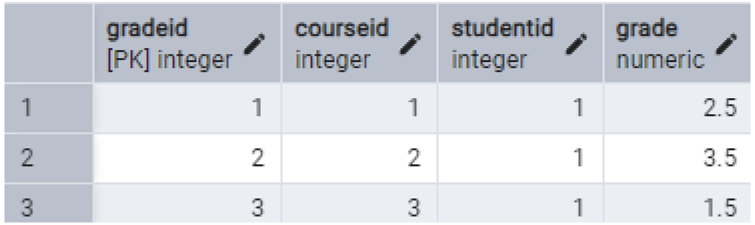
	courseid [PK] integer	coursename character varying (50)
1	1	Mathematics
2	2	Science
3	3	Programming

```
select * from student
```



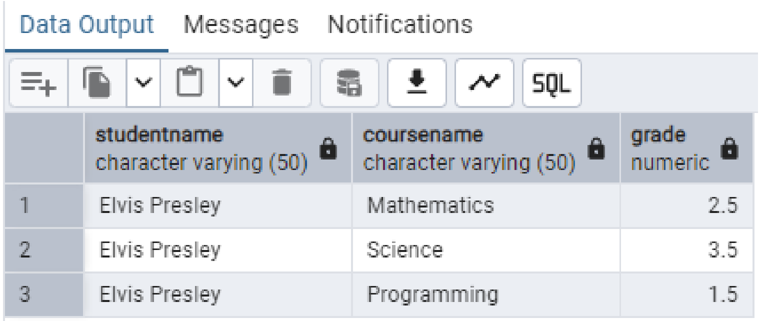
	studentid [PK] integer	studentname character varying (50)
1	1	Elvis Presley
2	2	John Wayne
3	3	John Statham

```
select * from grade
```



	gradeid [PK] integer	courseid integer	studentid integer	grade numeric
1	1	1	1	2.5
2	2	2	1	3.5
3	3	3	1	1.5

But we want to get information like this:



	studentname character varying (50)	coursename character varying (50)	grade numeric
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5

But it is not possible because the information is stored in 3 different tables.

=> The solution is to create and use a **View**.

Views

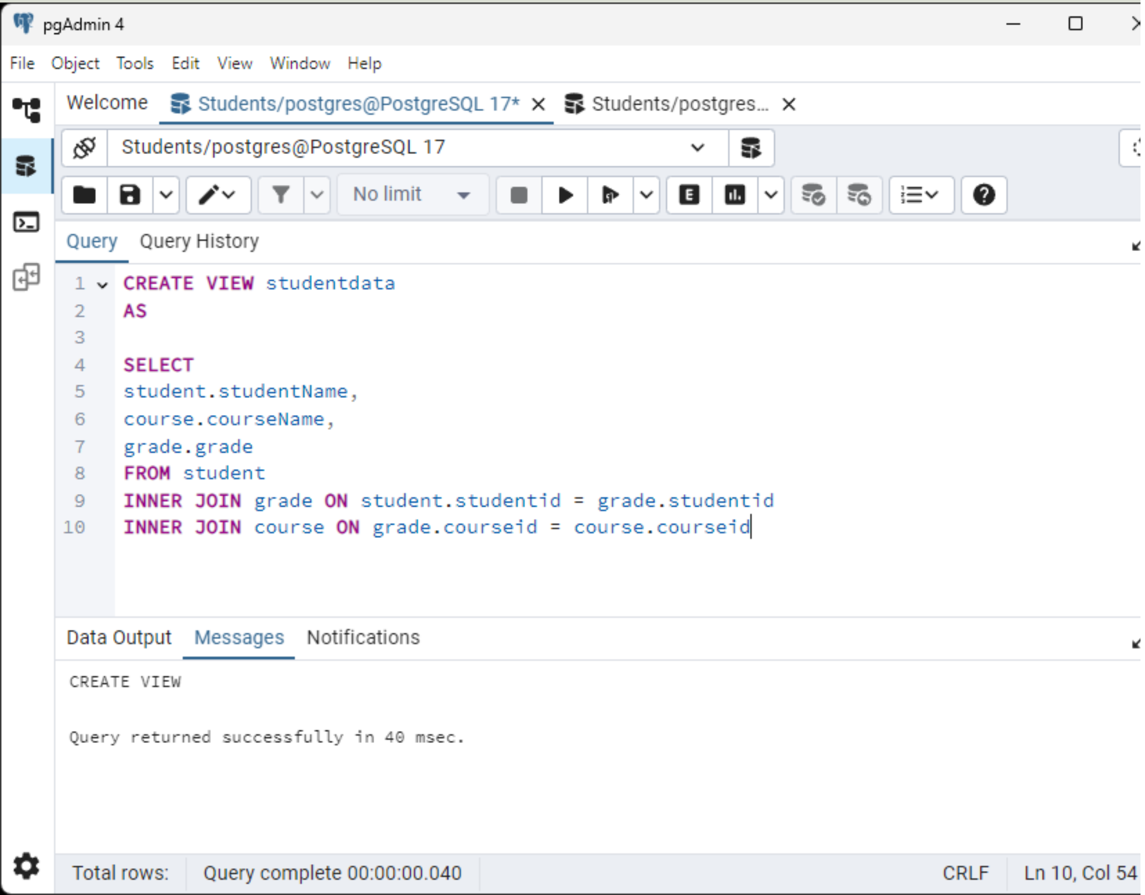
- A View is a “virtual” table that can contain data from multiple tables.
- Basically, a View is a SQL query that links 2 or more tables together making it possible to get data from these tables in a single query.

View Example

```
CREATE OR REPLACE VIEW studentdata
AS

SELECT
student.studentName,
course.courseName,
grade.grade
FROM student
INNER JOIN grade ON student.studentid = grade.studentid
INNER JOIN course ON grade.courseid = course.courseid
```

In a View we typically use "**INNER JOIN**" to join information stored in different Tables.

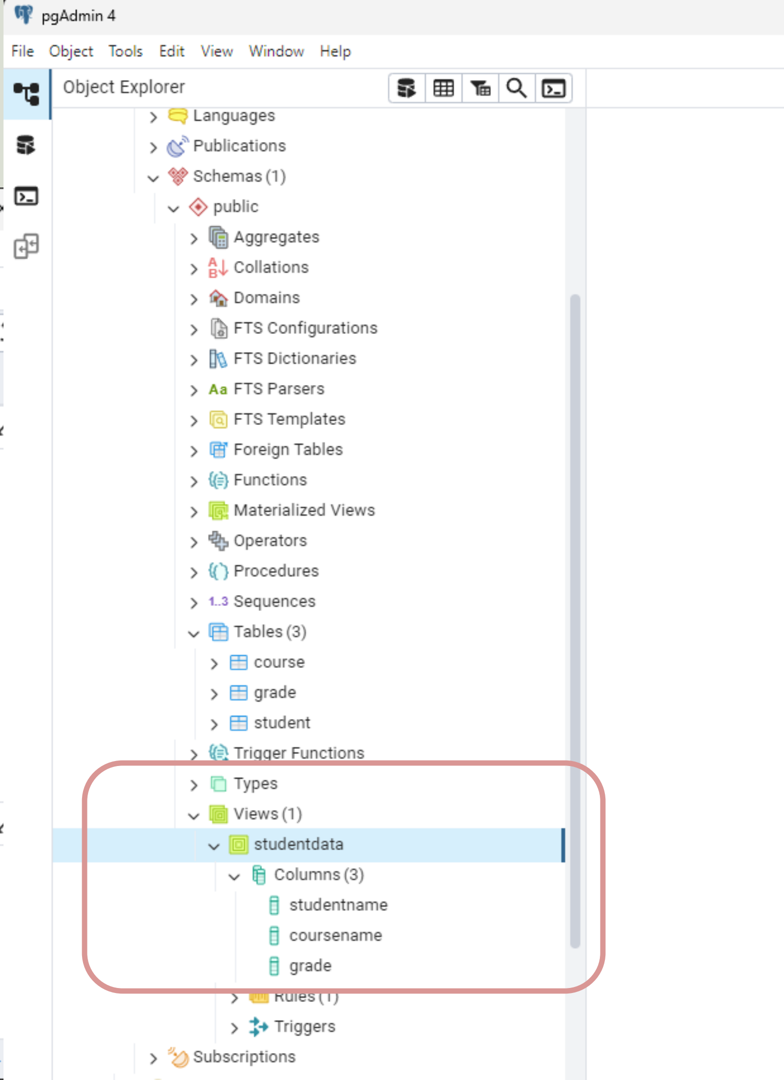
Create the View


pgAdmin 4
File Object Tools Edit View Window Help
Welcome
Students/postgres@PostgreSQL 17*
Students/postgres...
Students/postgres@PostgreSQL 17
No limit
Query
Query History
Data Output
Messages
Notifications

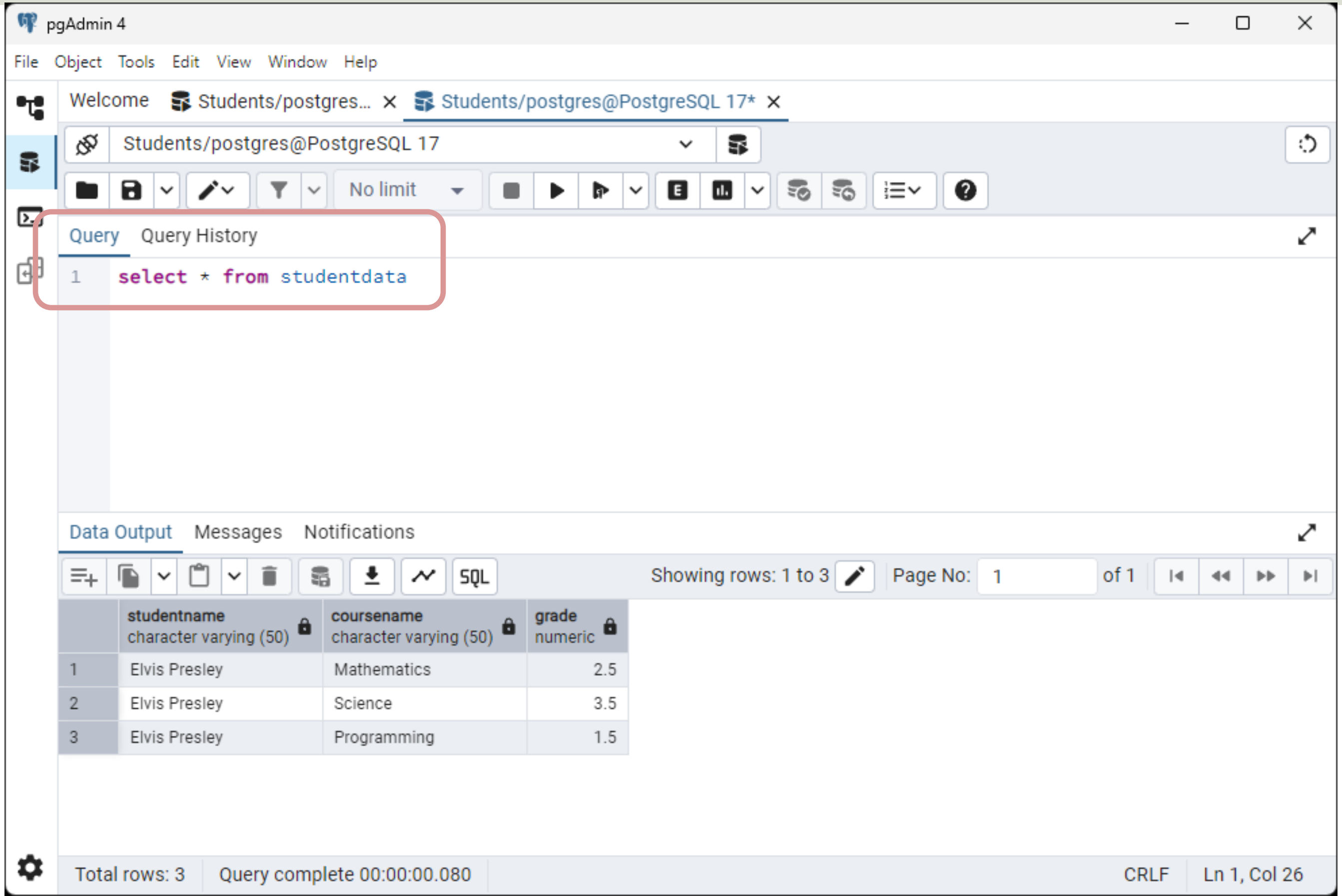
```
CREATE VIEW studentdata
AS

SELECT
student.studentName,
course.courseName,
grade.grade
FROM student
INNER JOIN grade ON student.studentid = grade.studentid
INNER JOIN course ON grade.courseid = course.courseid
```

```
CREATE VIEW

Query returned successfully in 40 msec.
```

Total rows: Query complete 00:00:00.040 CRLF Ln 10, Col 54



Using the View



```
select * from studentdata
```

	studentname character varying (50)	coursename character varying (50)	grade numeric
1	Elvis Presley	Mathematics	2.5
2	Elvis Presley	Science	3.5
3	Elvis Presley	Programming	1.5

Total rows: 3 Query complete 00:00:00.080 CRLF Ln 1, Col 26

Views Queries Examples

You can use Views almost as you use Tables. Here are some basic examples:

```
select * from studentdata

select coursename, grade from studentdata where studentname = 'Elvis Presley'

select studentname, grade from studentdata where coursename = 'Mathematics'

select avg(grade) as avgrade from studentdata where studentname = 'Elvis Presley'

..
```

PostgreSQL

Stored Procedures

Hans-Petter Halvorsen

Problem Description

To create/insert Grades we need to create and execute queries like this:

```
insert into GRADE (CourseId, StudentId, Grade) values (1, 1, 2.5)

insert into GRADE (CourseId, StudentId, Grade) values (2, 1, 3.5)

insert into GRADE (CourseId, StudentId, Grade) values (3, 1, 1.5)
```

The “drawback” is that we need to remember the CourseIds and the StudentIds, typically we only remember and want to use their names.

=> The solution is to create and use a **Stored Procedure**.

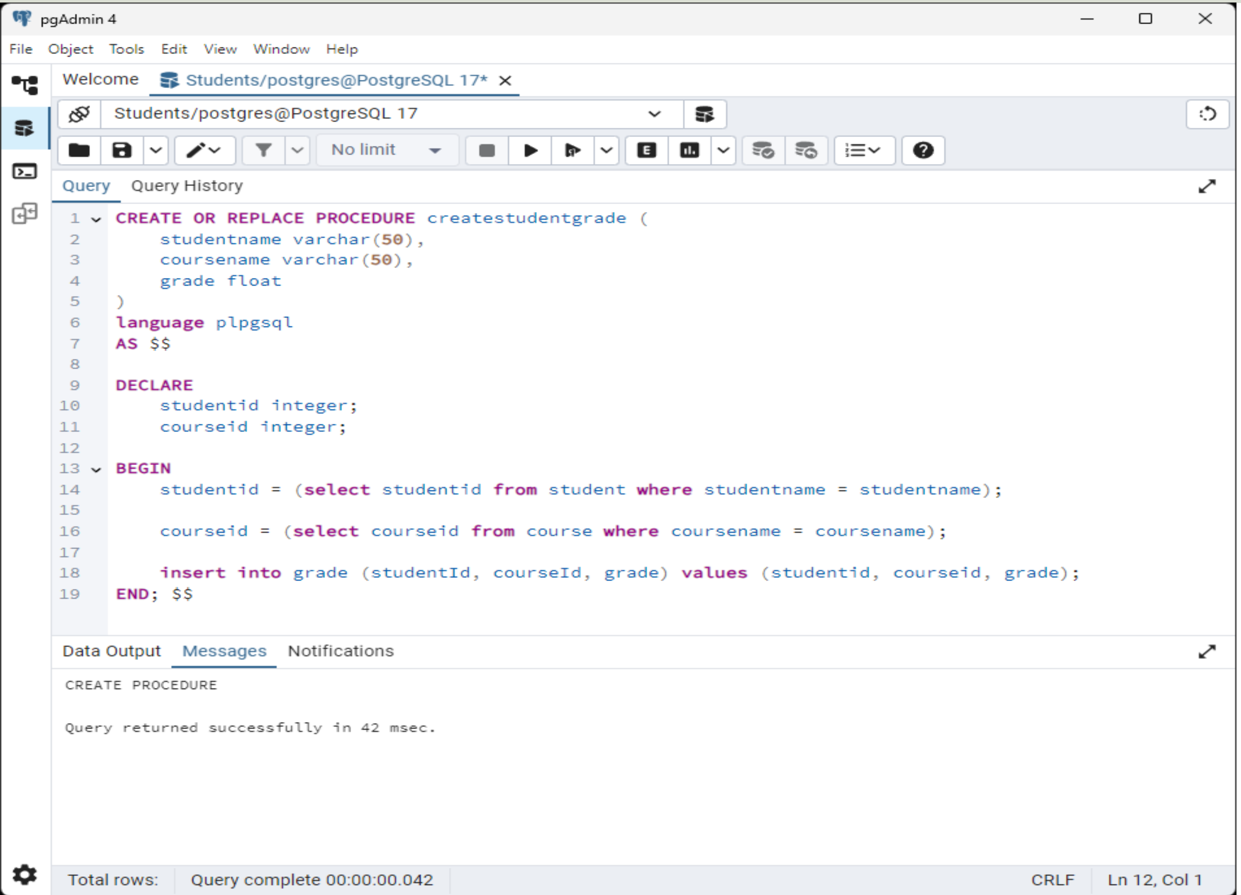
Stored Procedures

- A Stored Procedure is very similar as a Method/Function in C# or Python - it is a piece of code with SQL commands that do a specific task – and you can reuse it.
- A Stored Procedure can have Input Arguments and Return values (just like a Method/Function).
- It also adds a layer of security, because you can do a lot of harm by creating the wrong queries. In that way you can create a set of Stored Procedures that is well implemented and tested properly.
- Stored Procedures can also prevent “SQL Injection” used by “hackers”, etc.

Stored Procedure Example

```
CREATE OR REPLACE PROCEDURE createstudentgrade (
           studentname_in varchar(50),
           coursename_in varchar(50),      Input Arguments
           grade float
)
language plpgsql
AS $$

DECLARE
           studentid_var integer;          Internal variables
           courseid_var integer;

BEGIN
           studentid_var = (select studentid from student where studentname = studentname_in);

           courseid_var = (select courseid from course where coursename = coursename_in);

           insert into grade (studentId, courseId, grade) values (studentid_var, courseid_var, grade);
END; $$
```

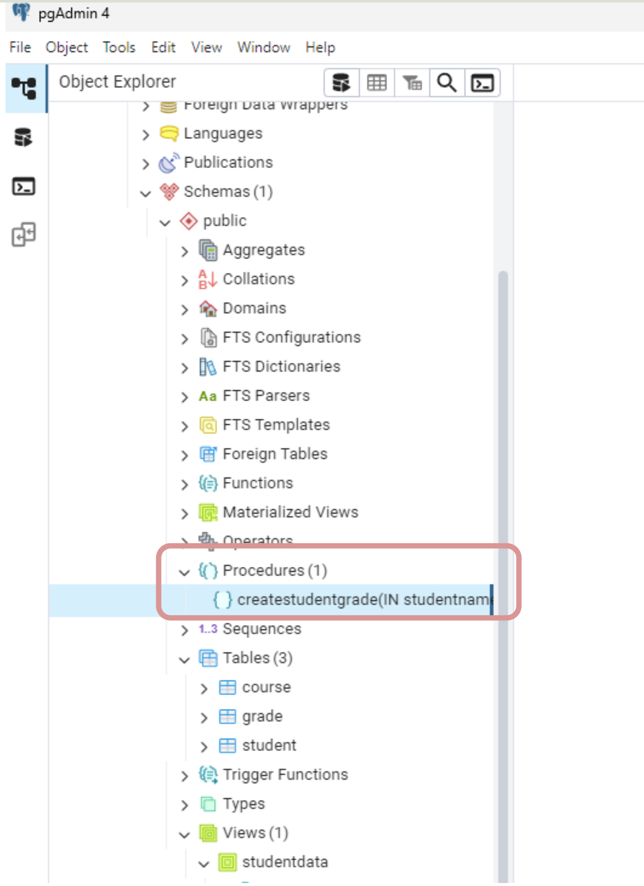
Create the Stored Procedure



pgAdmin 4
File Object Tools Edit View Window Help
Welcome
Students/postgres@PostgreSQL 17*
Students/postgres@PostgreSQL 17
No limit
Query
Query History

```sql
CREATE OR REPLACE PROCEDURE createstudentgrade (
    studentname varchar(50),
    coursename varchar(50),
    grade float
)
language plpgsql
AS $$

DECLARE
    studentid integer;
    courseid integer;

BEGIN
    studentid = (select studentid from student where studentname = studentname);

    courseid = (select courseid from course where coursename = coursename);

    insert into grade (studentId, courseId, grade) values (studentid, courseid, grade);
END; $$
```

```
CREATE PROCEDURE

Query returned successfully in 42 msec.
```

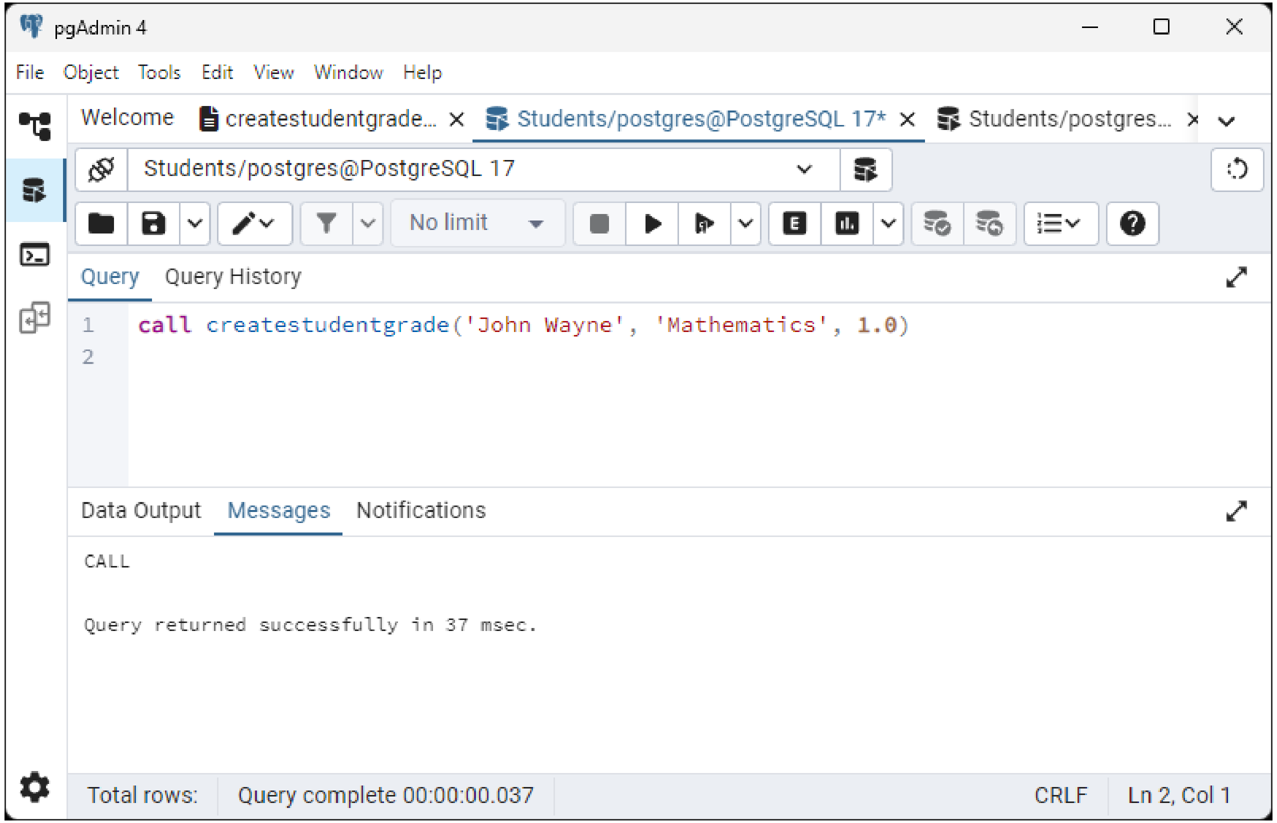


Using the Stored Procedure

```
insert into GRADE (CourseId, StudentId, Grade) values (1, 1, 2.5)

insert into GRADE (CourseId, StudentId, Grade) values (2, 1, 3.5)

insert into GRADE (CourseId, StudentId, Grade) values (3, 1, 1.5)
```

```
call createstudentgrade('John Wayne', 'Mathematics', 1.0)

call createstudentgrade('John Wayne', 'Science’, 2.0)

call createstudentgrade('John Wayne', 'Programming’, 3.0)
```

Using the Stored Procedure



```sql
call createstudentgrade('John Wayne', 'Mathematics', 1.0)
```

```
CALL

Query returned successfully in 37 msec.
```

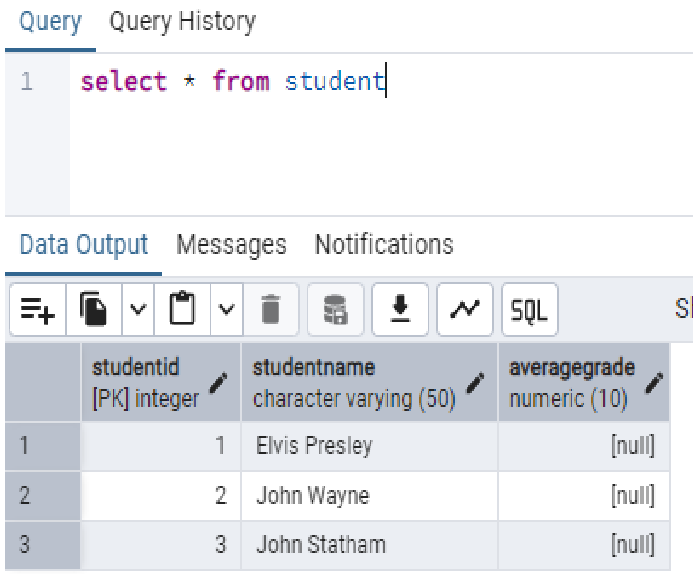
Total rows: Query complete 00:00:00.037 CRLF Ln 2, Col 1

PostgreSQL

Triggers

Hans-Petter Halvorsen

Problem Description


Query Query History

```
select * from student
```

Data Output Messages Notifications

	studentid [PK] integer	studentname character varying (50)	averagegrade numeric (10)
1	1	Elvis Presley	[null]
2	2	John Wayne	[null]
3	3	John Statham	[null]

```
call createstudentgrade('John Wayne', 'Mathematics', 1.0)

call createstudentgrade('John Wayne', 'Science’, 2.0)

call createstudentgrade('John Wayne', 'Programming’, 3.0)
```

We want to automatically update the “averagegrade” for each student when inserting, updating or deleting Grades for a specific Student in a specific Course.
=> The solution is to create and use a **Trigger.**

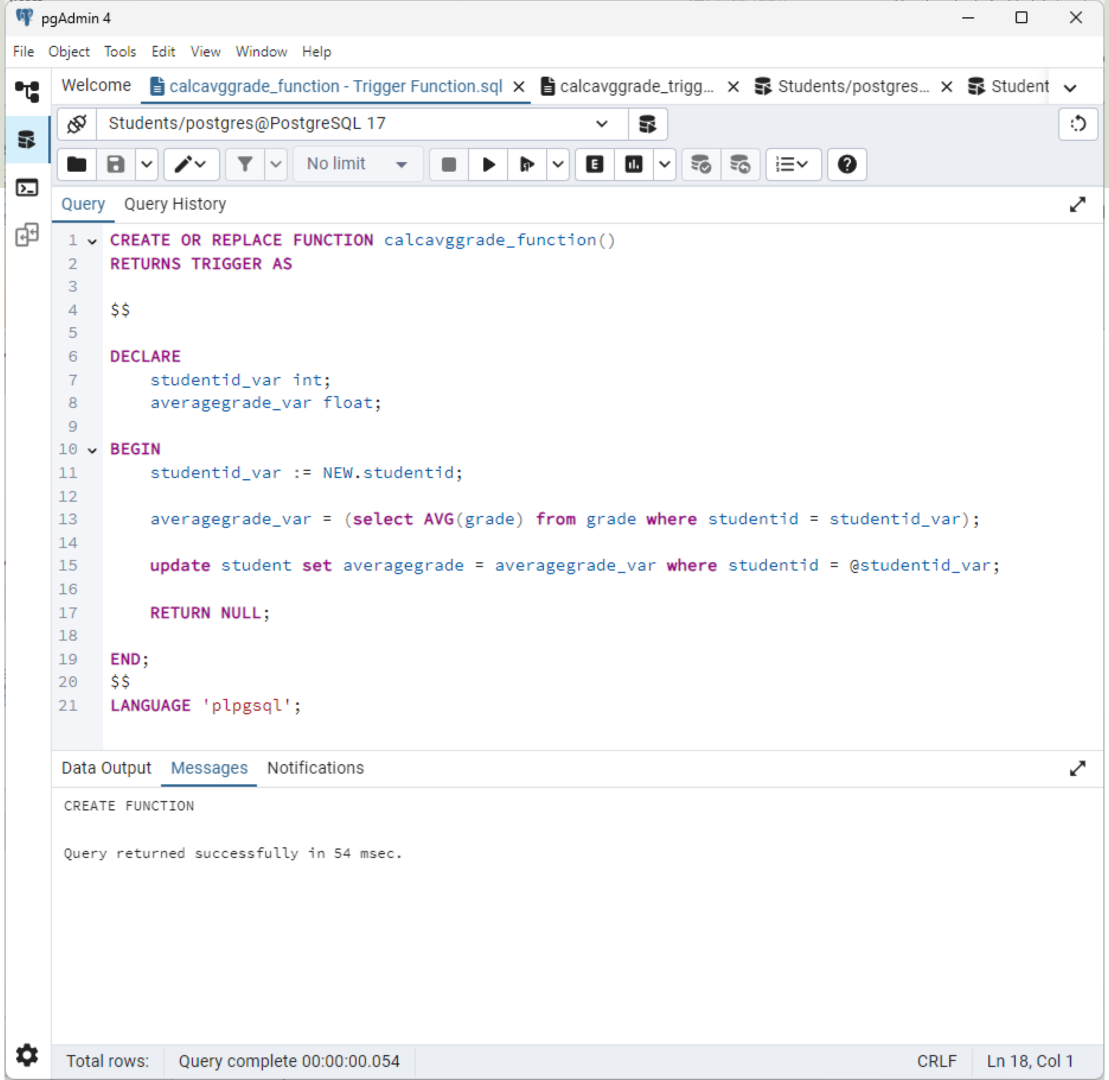
Triggers

- A Trigger is executed when you insert, update or delete data in a Table specified in the Trigger.
- A trigger is a stored procedure in a database that automatically invokes whenever a special event in the database occurs.
- A Trigger is attached to a specific Table.
- You can use a Trigger to change data in the same table or in other tables.
- We typically first make a Trigger Function then we make the Trigger itself that is attached to a specific Table, this Trigger then basically executes the Trigger Function.

Trigger Function Example

```
CREATE OR REPLACE FUNCTION calcavggrade_function()
RETURNS TRIGGER AS

$$

DECLARE
studentid_var int;
averagegrade_var float;

BEGIN
studentid_var := NEW.studentid;

averagegrade_var = (select AVG(grade) from grade where studentid = studentid_var);

update student set averagegrade = averagegrade_var where studentid = @studentid_var;

RETURN NULL;

END;
$$
LANGUAGE 'plpgsql';
```

Note! "NEW" is a temporarily table containing the latest inserted data, and it is very handy to use inside a Trigger Function.



```sql
CREATE OR REPLACE FUNCTION calcavggrade_function()
RETURNS TRIGGER AS

$$

DECLARE
    studentid_var int;
    averagegrade_var float;

BEGIN
    studentid_var := NEW.studentid;

    averagegrade_var = (select AVG(grade) from grade where studentid = studentid_var);

    update student set averagegrade = averagegrade_var where studentid = @studentid_var;

    RETURN NULL;

END;
$$
LANGUAGE 'plpgsql';
```

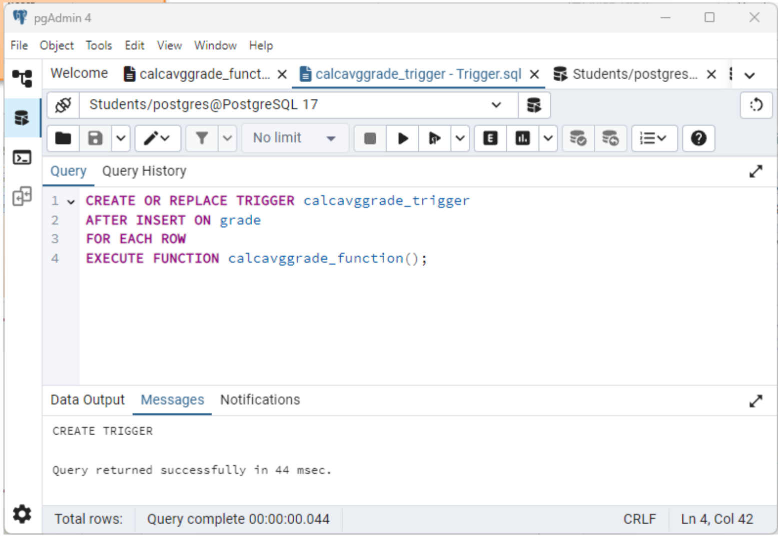
```
CREATE FUNCTION

Query returned successfully in 54 msec.
```

Trigger Example

The Trigger basically just call/execute the Trigger Function calcavggrade_function()

```
CREATE OR REPLACE TRIGGER calcavggrade_trigger
AFTER INSERT ON grade
FOR EACH ROW
EXECUTE FUNCTION calcavggrade_function();
```

You need to specify which Table the Trigger shall be attached to.

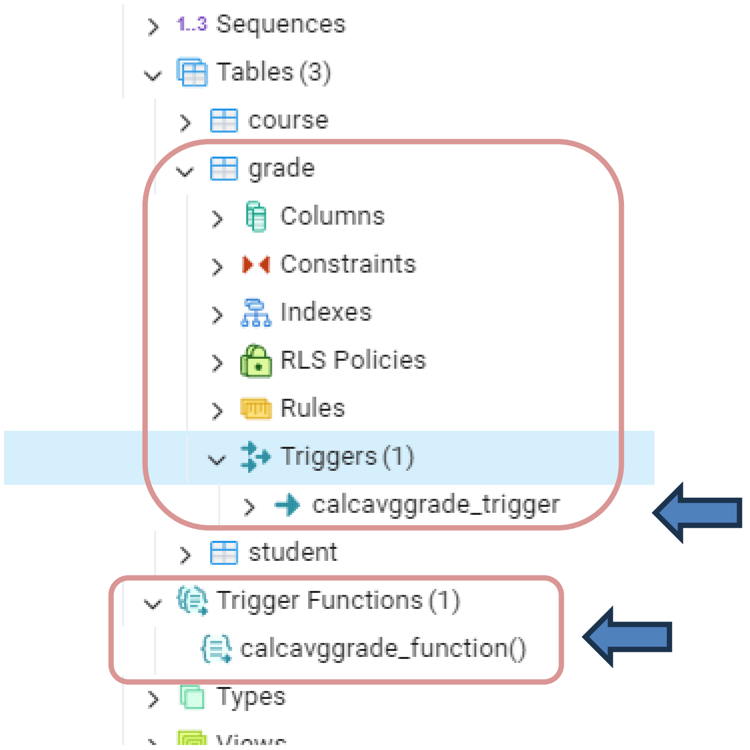


```
CREATE OR REPLACE TRIGGER calcavggrade_trigger
AFTER INSERT ON grade
FOR EACH ROW
EXECUTE FUNCTION calcavggrade_function();
```

```
CREATE TRIGGER

Query returned successfully in 44 msec.
```

Total rows: Query complete 00:00:00.044 CRLF Ln 4, Col 42

Trigger Function + Trigger



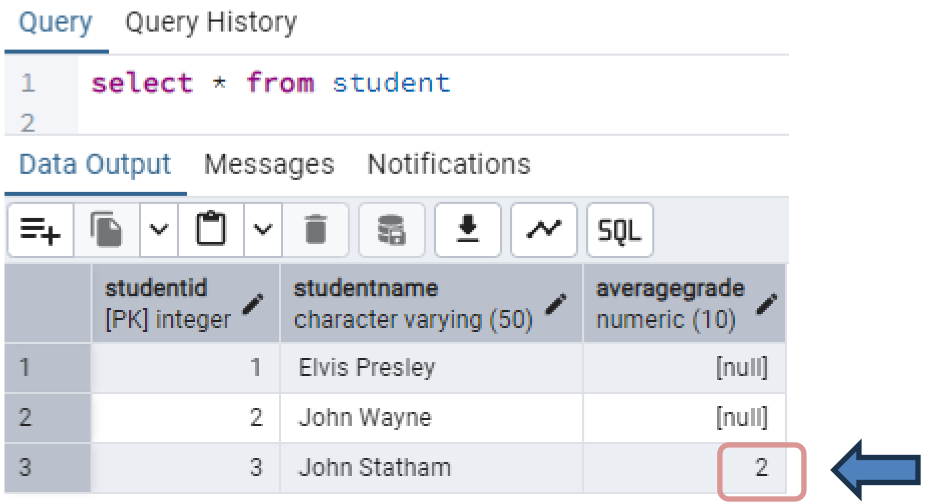
Insert Grades

We use the Stored Procedure created earlier:

```
call createstudentgrade('John Statham', 'Mathematics', 2.0)
```

```
call createstudentgrade('John Statham', ‘Science’, 3.0)
```

```
call createstudentgrade('John Statham’, Programming’, 1.0)
```



```
select * from student
```

	studentid [PK] integer	studentname character varying (50)	averagegrade numeric (10)
1	1	Elvis Presley	[null]
2	2	John Wayne	[null]
3	3	John Statham	2

Hans-Petter Halvorsen

University of South-Eastern Norway

www.usn.no

E-mail: hans.p.halvorsen@usn.no

Web: https://www.halvorsen.blog