Introduction to database

There are two definitions of a database. The first definition is that a database is an organized collection of information or data. A database also gives us a method of accessing and manipulating the data. So a database would allow us to order the cocktails, for instance, by page number and they would also allow us to only see the cocktails containing tequila.

SQL

Structured Query Language is the language we use to issue commands to the database. With it, we can create/insert data, read/select some data, update date, delete data, etc.

Terminology

- Database: contains one or more tables
- Relation (or table): contains tuples and attributes
- Attribute (also column or field) – one of possibly many elements of data corresponding to the object representated by the row.

Common Database Systems

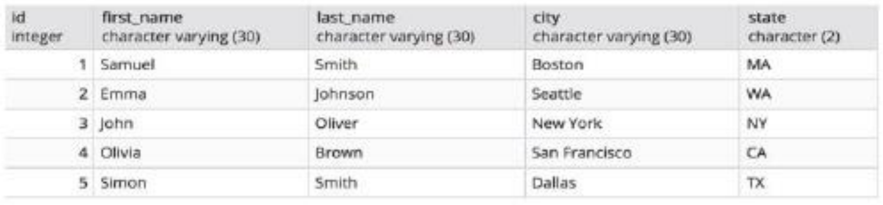
- Major Database management systems is wide use:
    - PostgreSQL: 100% open source, feature rich
    - Oracle: large commercial, enterprise-scale, very tweakable
    - MySQL: Fast and scalable: commercial open source
    - SqlServer: very nice – from Microsoft (also Access)
- Smaller projects: SQLite, HSQL…

Database Tables

Tables contain columns (fields) and rows of data (records). Each column has a defined data type which defines what type of data can be contained within than column. Each row of data should be unique. Each column should contain only one value per row.

In a relational database, tables can be linked together. Two tables are linked through primary keys and foreign keys.

This is an example of database tables.



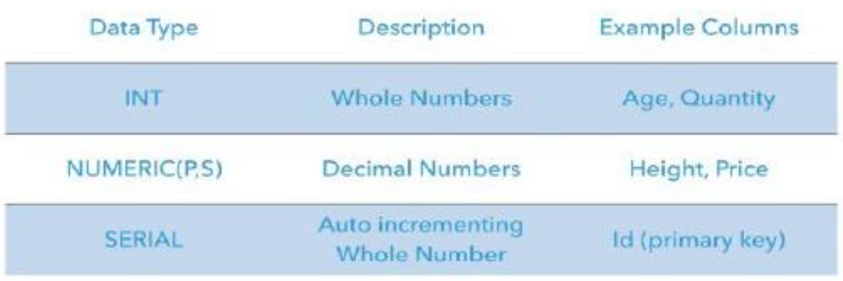
id	first_name	last_name	city	state
integer	character varying (30)	character varying (30)	character varying (30)	character (2)
1	Samuel	Smith	Boston	MA
2	Emma	Johnson	Seattle	WA
3	John	Oliver	New York	NY
4	Olivia	Brown	San Francisco	CA
5	Simon	Smith	Dallas	TX

PostgreSQL data type

Data Types PostgreSQL has a rich set of native data types available to users. Users can add new types to PostgreSQL using the CREATE TYPE command.

Numeric Types

Numeric types consist of two-, four-, and eight-byte integers, four- and eight-byte floating-point numbers, and selectable-precision decimals:



Data Type	Description	Example Columns
INT	Whole Numbers	Age, Quantity
NUMERIC(P,S)	Decimal Numbers	Height, Price
SERIAL	Auto incrementing Whole Number	Id (primary key)

Name	Storage Size	Description	Range
`smallint`	2 bytes	small-range integer	-32768 to +32767
`integer`	4 bytes	typical choice for integer	-2147483648 to +2147483647
`bigint`	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807

Name	Storage Size	Description	Range
`decimal`	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
`numeric`	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
`real`	4 bytes	variable-precision, inexact	6 decimal digits precision
`double precision`	8 bytes	variable-precision, inexact	15 decimal digits precision
`smallserial`	2 bytes	small autoincrementing integer	1 to 32767
`serial`	4 bytes	autoincrementing integer	1 to 2147483647
`bigserial`	8 bytes	large autoincrementing integer	1 to 9223372036854775807

Integer Types

The types smallint, integer, and bigint store whole numbers, that is, numbers without fractional components, of various ranges. Attempts to store values outside of the allowed range will result in an error. The type integer is the common choice, as it offers the best balance between range, storage size, and performance. The smallint type is generally only used if disk space is at a premium. The bigint type is designed to be used when the range of the integer type is insufficient. SQL only specifies the integer types integer (or int), smallint, and bigint. The type names int2, int4, and int8 are extensions, which are also used by some other SQL database systems

Arbitrary Precision Numbers

The type numeric can store numbers with a very large number of digits. It is especially recommended for storing monetary amounts and other quantities where exactness is required. Calculations with numeric values yield exact results where possible, e.g., addition, subtraction, multiplication. However, calculations on numeric values are very slow compared to the integer types, or to the floating-point types

We use the following terms below: The precision of a numeric is the total count of significant digits in the whole number, that is, the number of digits to both sides of the decimal point. The scale of a numeric is the count of decimal digits in the fractional part, to the right of the decimal point. So the number 23.5141 has a precision of 6 and a scale of 4. Integers can be considered to have a scale of zero. Both the maximum precision and the maximum scale of a numeric column can be configured.

The types decimal and numeric are equivalent. Both types are part of the SQL standard. When rounding values, the numeric type rounds ties away from zero, while (on most machines) the real and double precision types round ties to the nearest even number.

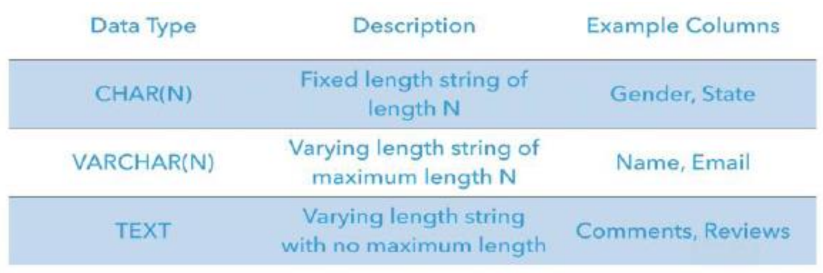
. Floating-Point Types

The data types real and double precision are inexact, variable-precision numeric types. Inexact means that some values cannot be converted exactly to the internal format and are stored as approximations, so that storing and retrieving a value might show slight discrepancies. Managing these errors and how they propagate through calculations is the subject of an entire branch of mathematics and computer science and will not be discussed here, except for the following points:

- If you require exact storage and calculations (such as for monetary amounts), use the numeric type instead.
- If you want to do complicated calculations with these types for anything important, especially if you rely on certain behavior in boundary cases (infinity, underflow), you should evaluate the implementation carefully.
- Comparing two floating-point values for equality might not always work as expected.

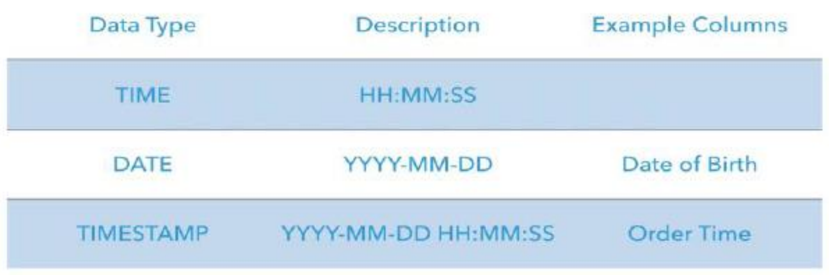
By default, floating point values are output in text form in their shortest precise decimal representation; the decimal value produced is closer to the true stored binary value than to any other value representable in the same binary precision. (However, the output value is currently never exactly midway between two representable values, in order to avoid a widespread bug where input routines do not properly respect the round-to-nearest-even rule.) This value will use at most 17 significant decimal digits for float8 values, and at most 9 digits for float4 values.

Character or string Types

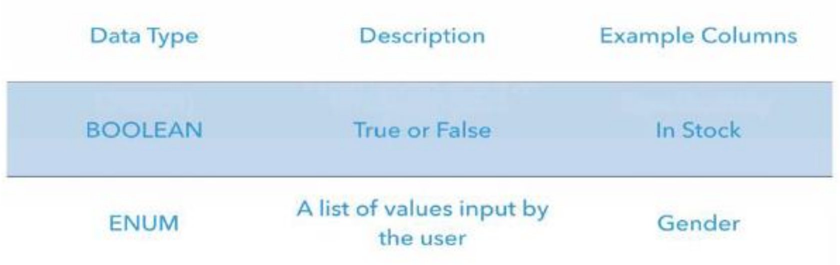


Data Type	Description	Example Columns
CHAR(N)	Fixed length string of length N	Gender, State
VARCHAR(N)	Varying length string of maximum length N	Name, Email
TEXT	Varying length string with no maximum length	Comments, Reviews

SQL defines two primary character types: character varying(n) and character(n), where n is a positive integer. Both of these types can store strings up to n characters (not bytes) in length. An attempt to store a longer string into a column of these types will result in an error, unless the excess characters are all spaces, in which case the string will be truncated to the maximum length. (This somewhat bizarre exception is required by the SQL standard.) If the string to be stored is shorter than the declared length, values of type character will be space-padded; values of type character varying will simply store the shorter string.



Data Type	Description	Example Columns
TIME	HH:MM:SS	
DATE	YYYY-MM-DD	Date of Birth
TIMESTAMP	YYYY-MM-DD HH:MM:SS	Order Time

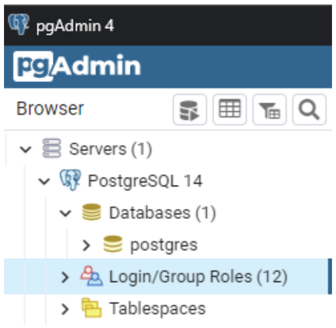


Data Type	Description	Example Columns
BOOLEAN	True or False	In Stock
ENUM	A list of values input by the user	Gender

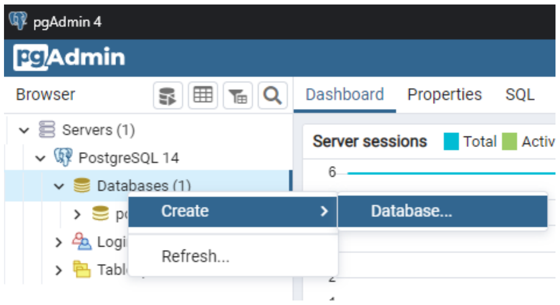
Creating database

The first test to see whether you can access the database server is to try to create a database. A running PostgreSQL server can manage many databases. Typically, a separate database is used for each project or for each user.

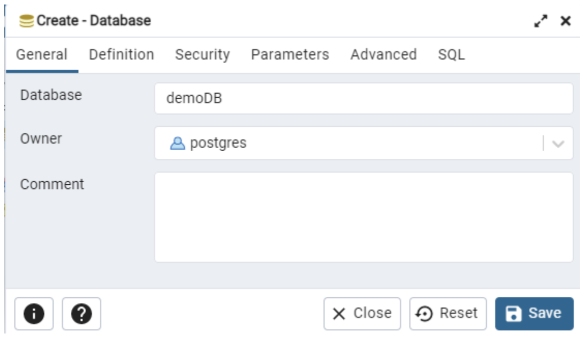
To create a new database, in this example named **demoDB**, we can create by login-in to PostgreSQL



From the left of the Browser column, go and right-click on Databases icon, move the mouse to hover the Create link and click on Database…



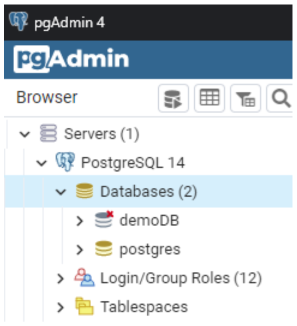
The Create-Database window appears, on General, you can fill up with the following info:


Create - Database

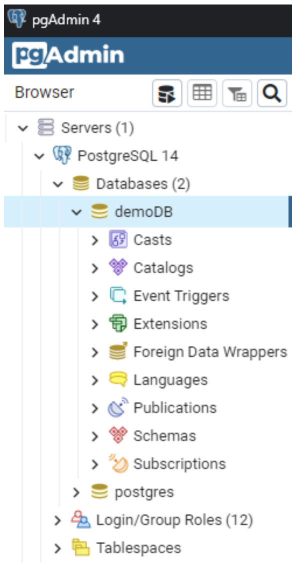
General | Definition | Security | Parameters | Advanced | SQL

Database	demoDB
Owner	postgres
Comment	

And click on Save. When you clicked on Save, the new database **demoDB** appears on the left column as a database:

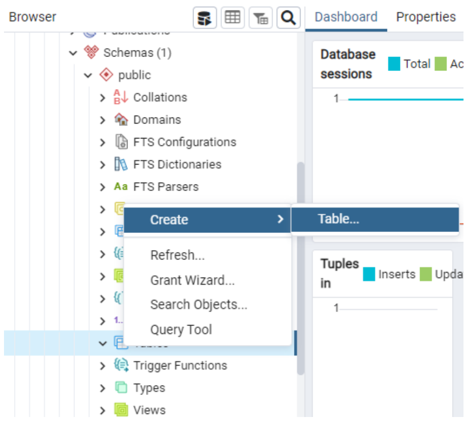


If you click on the new database demoDB, it expands and show the items that we can work with the database:

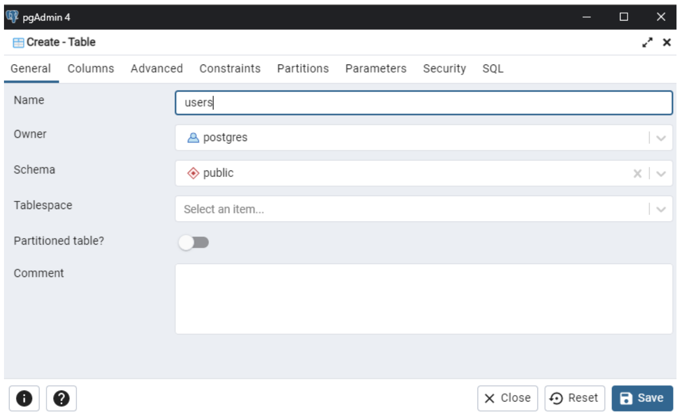


Create a table in PostgreSQL

Click on the Schema icon to open and dropdown the Schema list. Once in Schema, go to Tables and right-click and select new table:

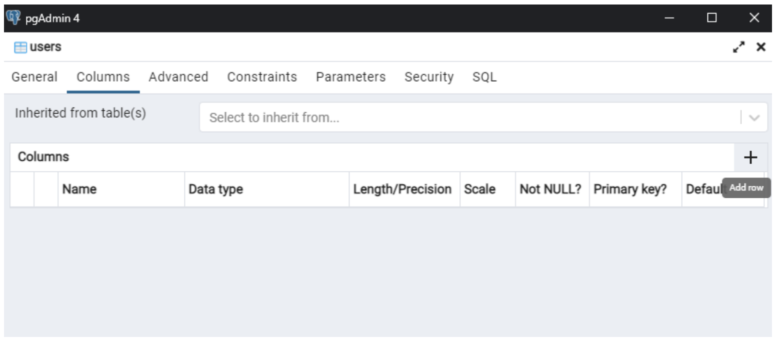


On the table, on General, we can name our table **users**:

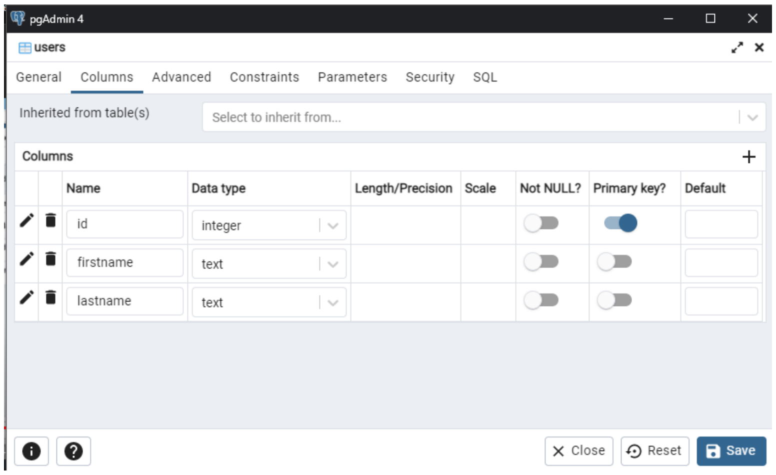


Name	users
Owner	postgres
Schema	public
Tablespace	Select an item...
Partitioned table?	
Comment	

We can also go the Columns tab and click on the plus icon to add columns to our table users. We create an ID column, as an integer value, and set as the primary key:



Let us to create those three column:



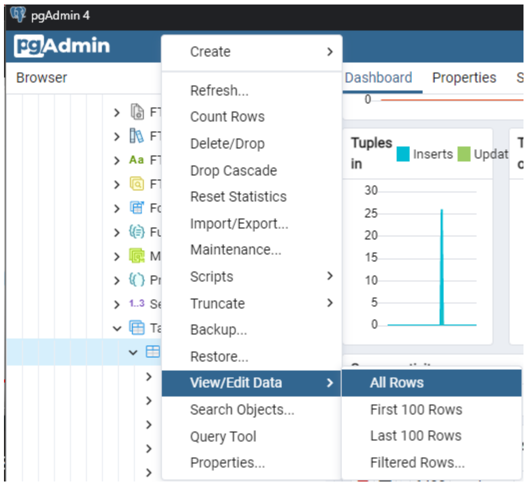
Inherited from table(s): Select to inherit from...

Columns

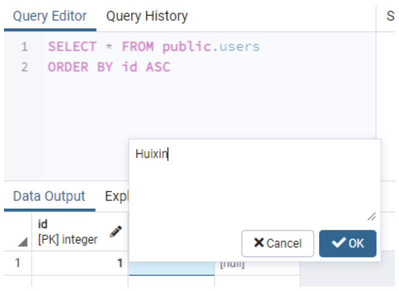
Name	Data type	Length/Precision	Scale	Not NULL?	Primary key?	Default
id	integer			No	Yes	
firstname	text			No	No	
lastname	text			No	No	

Once we finish we our table users, we can click on Save.

We can add data manually or import data from outside the database. To add data manually, we can right click on table users, go to View/Edit data, and click on All Rows:



When you do so, the table appears and on the bottom of the table, we can add data manually. To input the data to **firstname** and **lastname**, we double-click on the field and type a text in the dialog box :


```
SELECT * FROM public.users
ORDER BY id ASC
```

Using SQL statement to create and insert data into a table

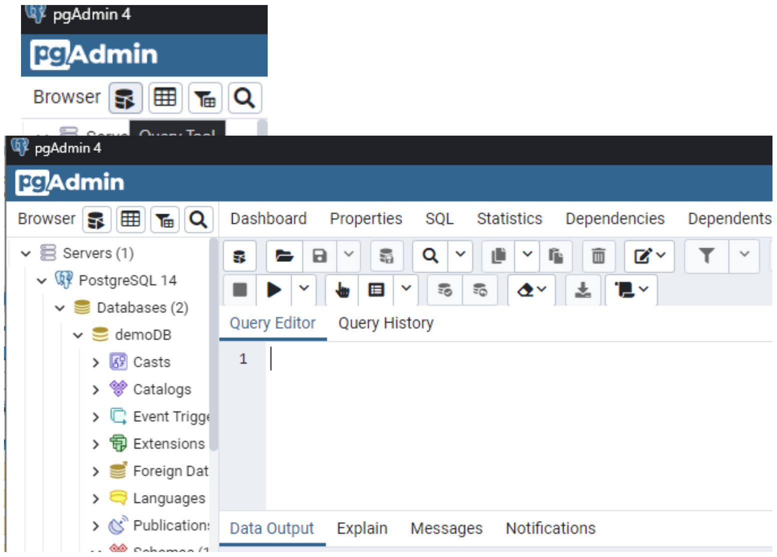
SQL CREATE TABLE

Create a table into our database

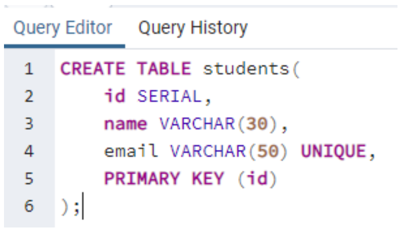
Syntax:

```
CREATE TABLE table's name(
        …
);
```

We can create a table using SQL statement. For this, we can go to the Query Tool to open the Query Editor

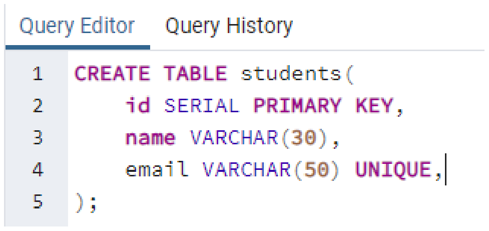


Let us create a table *students* with three columns: id, name, and email. The **`id`** is going to be the primary key and it will automatically increment; the name accepts strings up to 30 characters; and an email that will set as unique entry up to 50 characters.



```
CREATE TABLE students(
    id SERIAL,
    name VARCHAR(30),
    email VARCHAR(50) UNIQUE,
    PRIMARY KEY (id)
);
```

An alternative to create set the PRIMARY KEY is:



```
CREATE TABLE students(
    id SERIAL PRIMARY KEY,
    name VARCHAR(30),
    email VARCHAR(50) UNIQUE,
);
```

SQL INSERT

The INSERT statement inserts a row into a table

Syntax

INSERT INTO *table's name* (*column(s)'s name*) VALUES (*values to columns*);

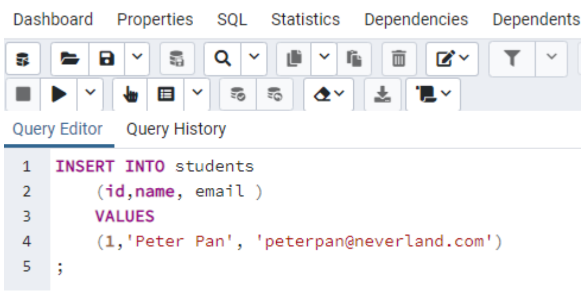
At the Query Editor, we can create manipulate our database data using SQL statement.

Let us create a table named **`students`** with **`id`**, **`name`**, and **`email address`**, as:

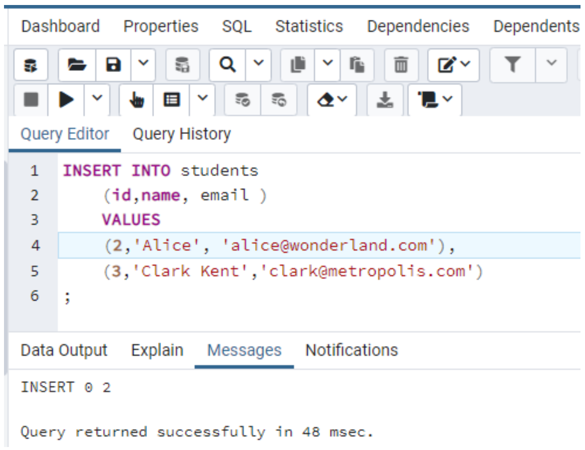
```
INSERT INTO students(id,name, email ) VALUES (1,'Peter Pan', peterpan@neverland.com');
```

Note: VARCHAR uses single quote for the values

To add multiple values, we have separate each set of values using a comma.



```
INSERT INTO students
    (id,name, email )
    VALUES
    (1,'Peter Pan', 'peterpan@neverland.com')
;
```

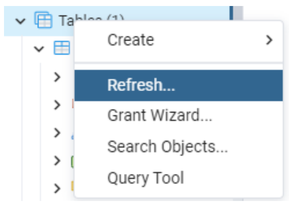


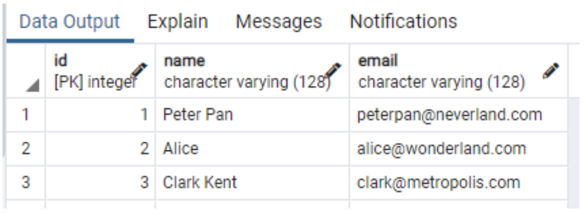
```
INSERT INTO students
    (id,name, email )
    VALUES
    (2,'Alice', 'alice@wonderland.com'),
    (3,'Clark Kent','clark@metropolis.com')
;
```

```
INSERT 0 2

Query returned successfully in 48 msec.
```

To view the table in our database, we can go to the left column, right-click on Table icons, and click on Refresh:





	id [PK] integer	name character varying (128)	email character varying (128)
1	1	Peter Pan	peterpan@neverland.com
2	2	Alice	alice@wonderland.com
3	3	Clark Kent	clark@metropolis.com

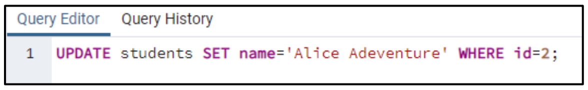
SQL UPDATE…SET… WHERE

UPDATE allows the updating of a field with a WHERE clause

Syntax

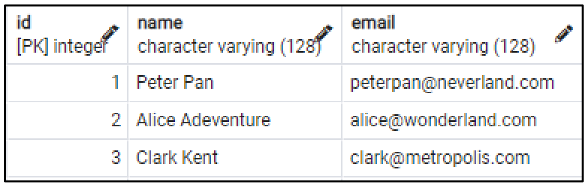
UPDATE *table's name* **SET** *column's name = 'values'* **WHERE** *column = value*

If we want to update one data from the table, we can use the statement below. For example, if we want to change/update the name Alice to Alice Adventure, we can write the line as:



```
UPDATE students SET name='Alice Adeventure' WHERE id=2;
```

Refreshing the table we will have:



id [PK] integer	name character varying (128)	email character varying (128)
1	Peter Pan	peterpan@neverland.com
2	Alice Adeventure	alice@wonderland.com
3	Clark Kent	clark@metropolis.com

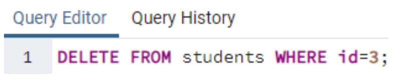
SQL DELETE

DELETE deletes a row in a table based on selection criteria

Syntax

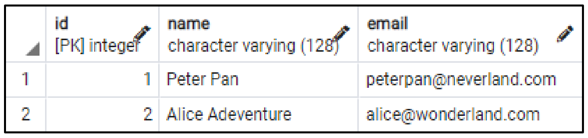
DELETE FROM *table's name* **WHERE** *column = value;*

For example, let us delete the third student of our table *students*



```
DELETE FROM students WHERE id=3;
```

If we refresh the table, this is we have:



	id [PK] integer	name character varying (128)	email character varying (128)
1	1	Peter Pan	peterpan@neverland.com
2	2	Alice Adeventure	alice@wonderland.com