Data Management and Analysis

Unit 2

SQL

Dott. Franco Liberati
liberati@di.uniroma1.it

SQL

General

- **SQL** stands for **Structured Query Language**
- SQL is used in order to access and manipulate databases
- SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987

SQL

General

SQL can:

- create new databases
- create new tables in a database
- insert records in a database
- update records in a database
- delete records from a database
- retrieve data from a database
  - execute queries against a database
- create stored procedures in a database
- create views in a database
- set permissions on tables, procedures, and views

SQL

General

Data Definition Language (DDL): database schema and data description

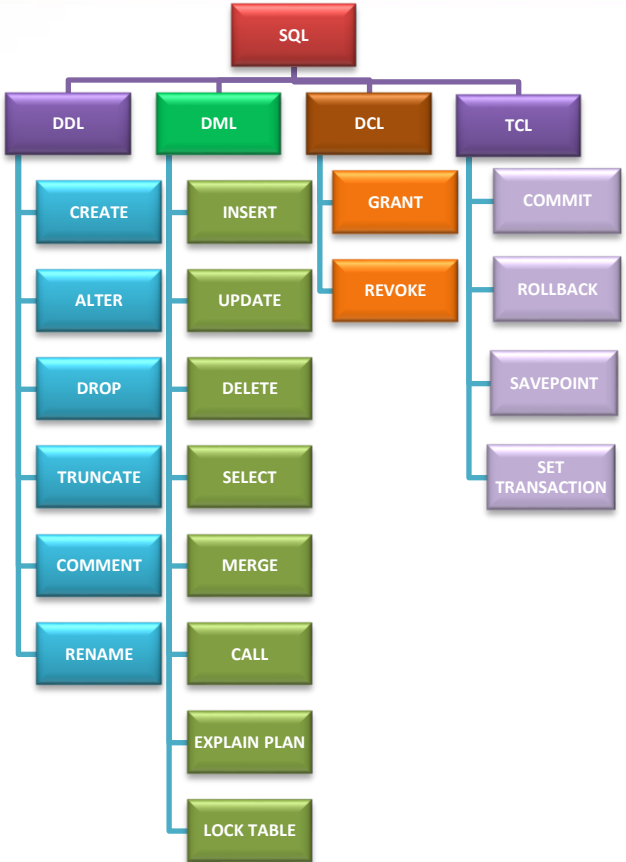
Data Manipulation Language (DML): used for storing, querying, modifying and deleting data

Data Control Language (DCL): deals with the management of permissions, rights and other forms of database control

Transactions Control Language (TCL): deals with the management of transactions

SQL

General


SQL
DDL
DML
DCL
TCL
CREATE
ALTER
DROP
TRUNCATE
COMMENT
RENAME
INSERT
UPDATE
DELETE
SELECT
MERGE
CALL
EXPLAIN PLAN
LOCK TABLE
GRANT
REVOKE
COMMIT
ROLLBACK
SAVEPOINT
SET TRANSACTION

SQL

General

To use SQL, you will need an **RDBMS database program** (i.e. MS Access, SQL Server, MySQL, PostreSQL)

The **data** in RDBMS is stored in database objects called **tables**.

A **table** is a collection of related data entries and it consists of columns and rows

Every table is broken up into smaller entities called **fields**

A **field** is a column in a table that is designed to maintain specific information about every record in the table

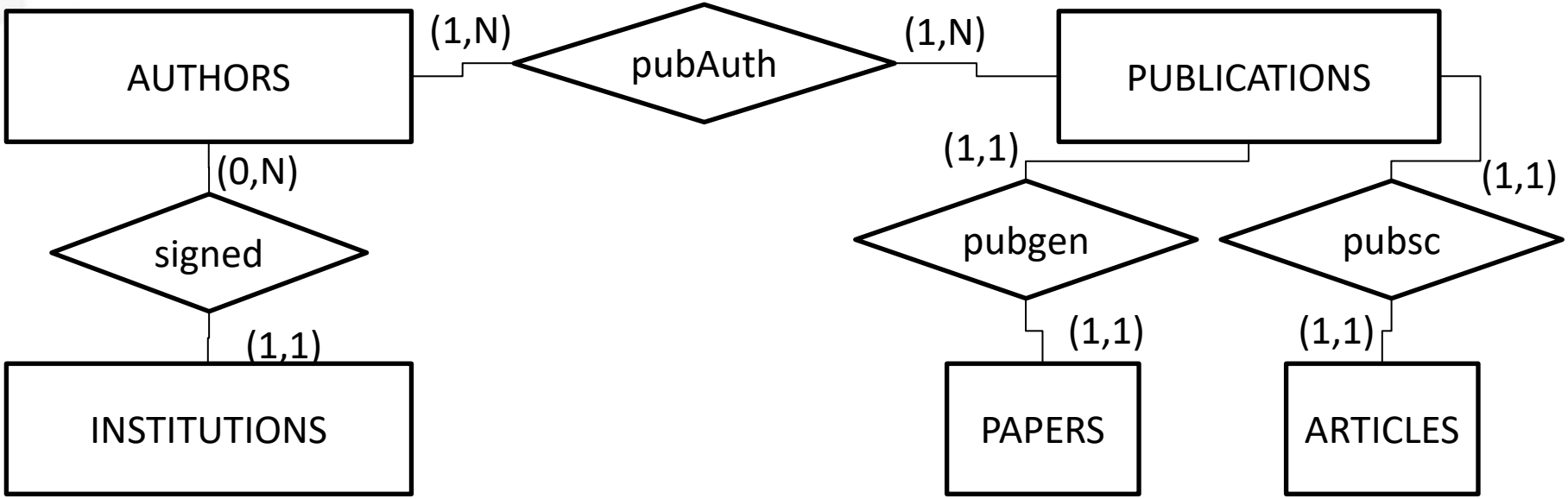
A **record**, also called a row, is each individual entry that exists in a table

A **column** is a vertical entity in a table that contains all information associated with a specific field in a table

SQL

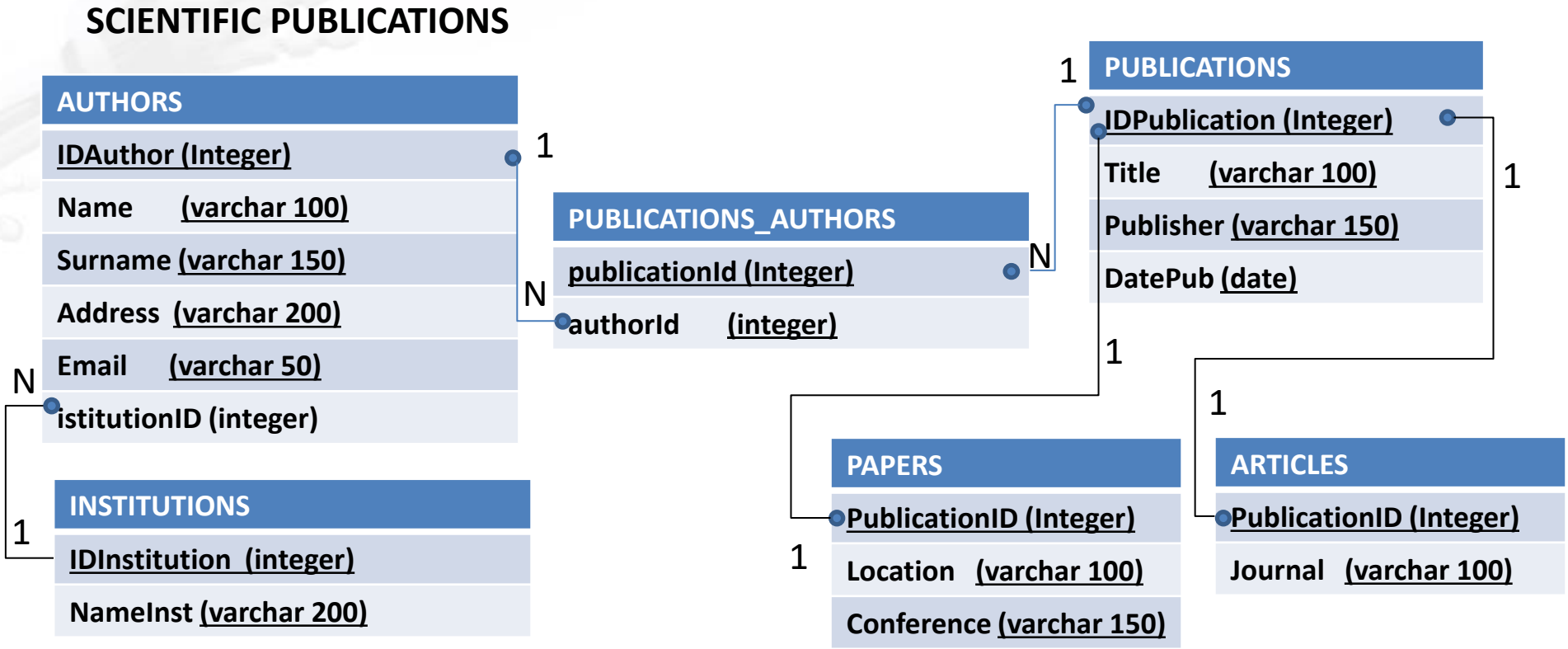
Sample Database: E R Model

SCIENTIFIC PUBLICATIONS



SQL

Sample Database: Relational Model



SQL DATABASE

PostgreSQL

PostgreSQL

General

PostgreSQL is a powerful, open source object-relational database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads

The origins of PostgreSQL date back to 1986 as part of the **POSTGRES** project at the University of California at Berkeley and has more than 35 years of active development on the core platform.

PostgreSQL

General

PostgreSQL comes with **many features** aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset

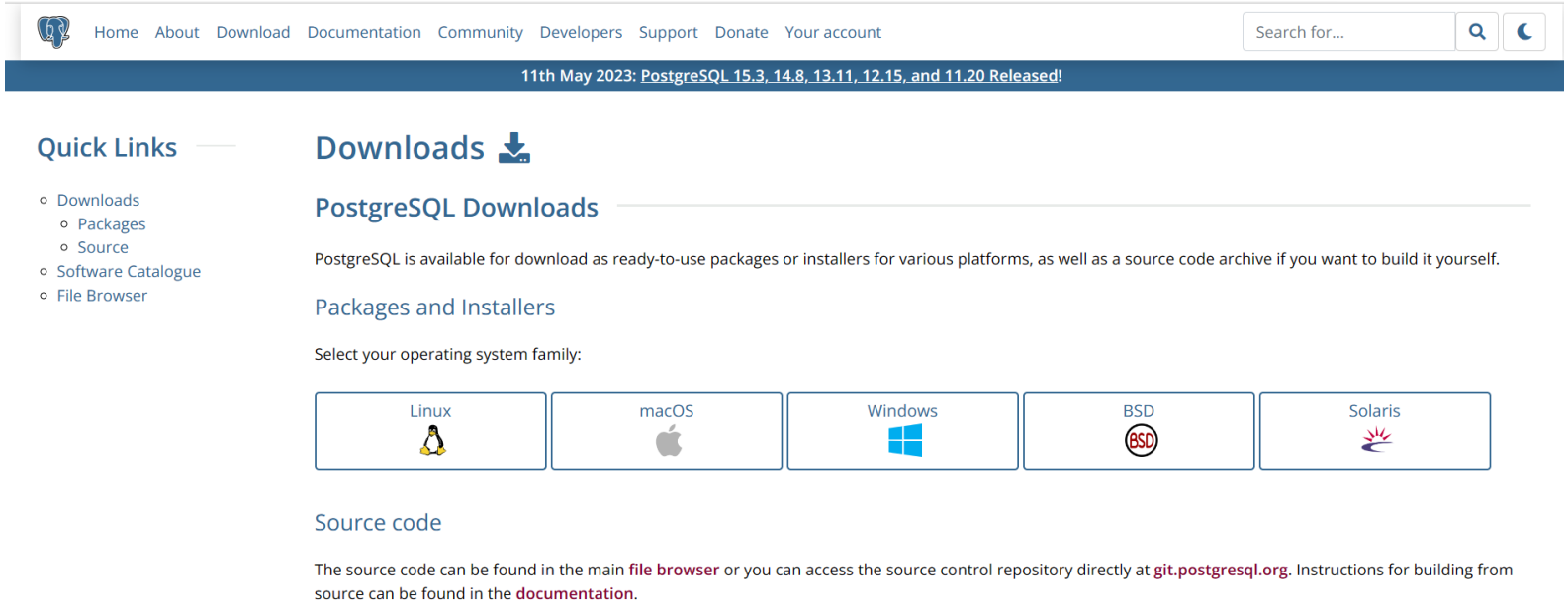
In addition to being **free and open source**, PostgreSQL is highly extensible. For example, you can define your own data types, build out custom functions, even write code from different programming languages without recompiling your database

PostgreSQL

DOWNLOAD

OFFICIAL WEBSITE:

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



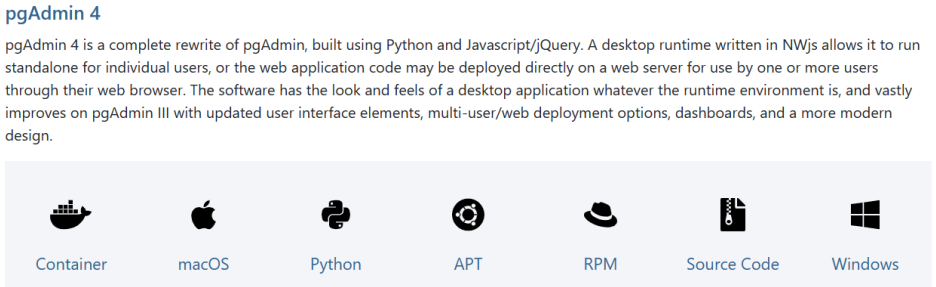
PostgreSQL

DOWNLOAD

PgAdmin is a management tool for PostgreSQL and derivative relational databases such as EnterpriseDB's EDB Advanced Server. It may be run either as a web or desktop application.

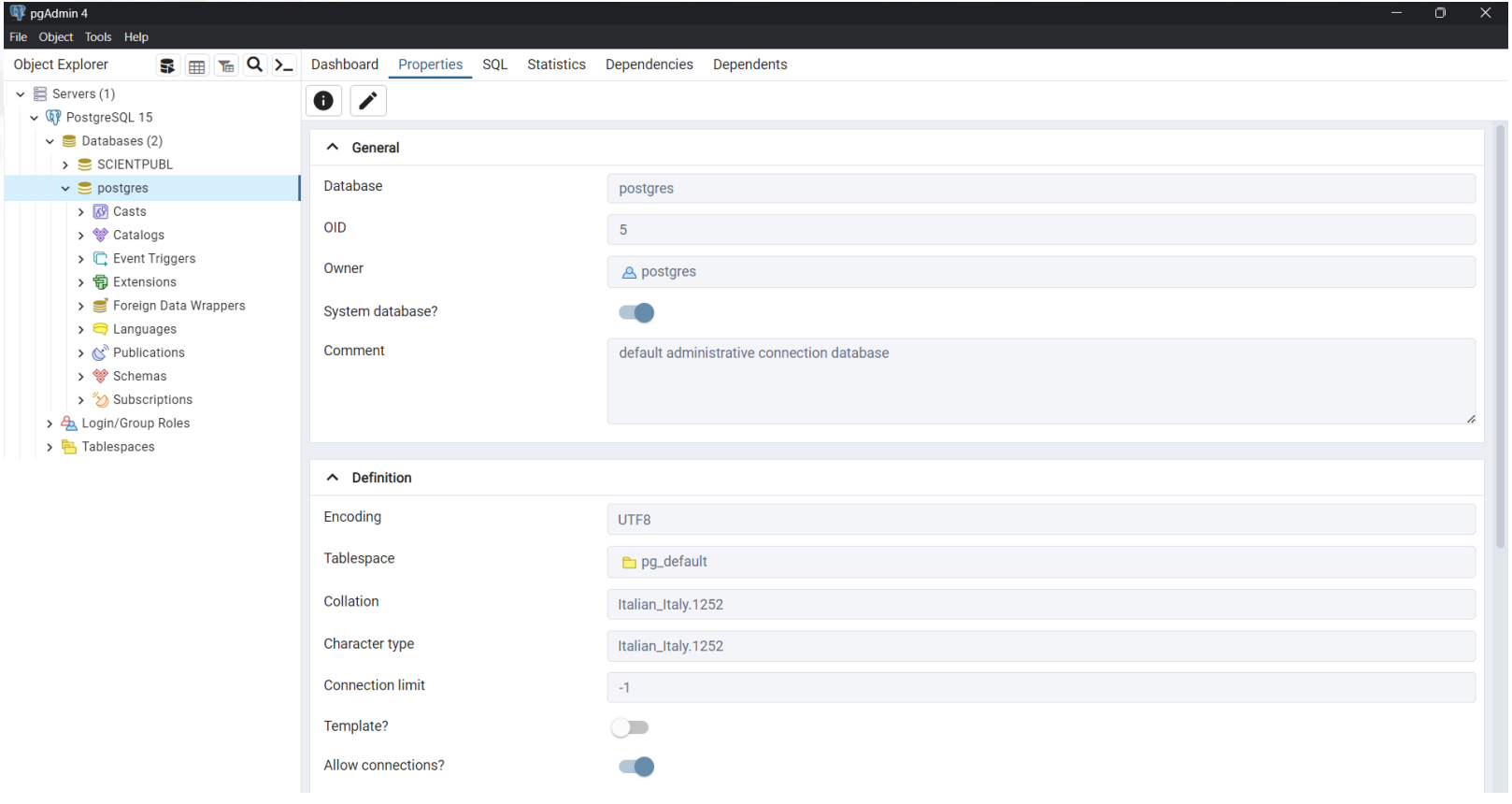
OFFICIAL WEBSITE:

https://www.pgadmin.org/download/



PostgreSQL

pgAdmin



SQL DATABASE

Data Definition Language

DDL

General

Data Definition Language (DDL) is a syntax for creating and modifying database objects such as tables, indices, and users

DDL statements are similar to a computer programming language for defining data structures, especially database schemas

DDL

CREATE DATABASE

The **CREATE DATABASE** statement is used to create a new SQL database

SINTAX

```
CREATE DATABASE databasename;
```

DDL

CREATE DATABASE POSTGRESQL

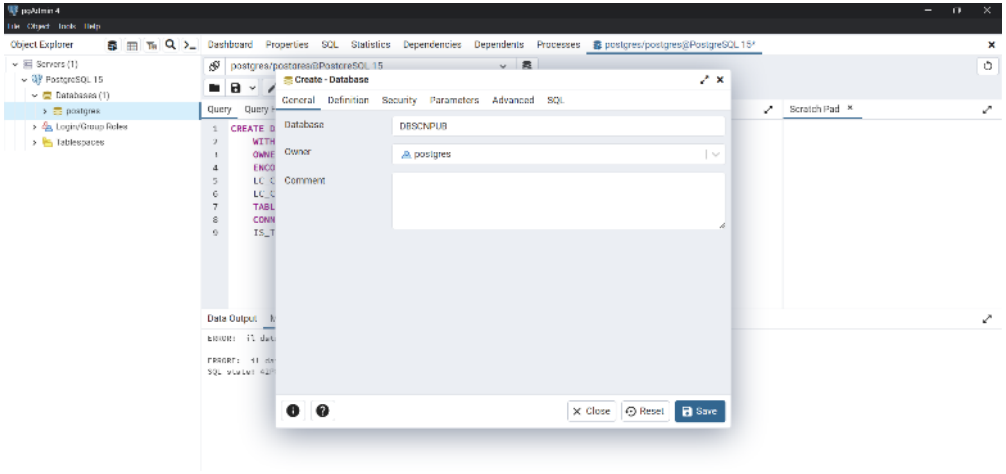
CREATE DATABASE statements creates a new database

```
CREATE DATABASE name
    [ WITH ] [ OWNER [=] user_name ]
           [ TEMPLATE [=] template ]
           [ ENCODING [=] encoding ]
           [ STRATEGY [=] strategy ] ]
           [ LOCALE [=] locale ]
           [ LC_COLLATE [=] lc_collate ]
           [ LC_CTYPE [=] lc_ctype ]
           [ ICU_LOCALE [=] icu_locale ]
           [ LOCALE_PROVIDER [=] locale_provider ]
           [ COLLATION_VERSION = collation_version ]
           [ TABLESPACE [=] tablespace_name ]
           [ ALLOW_CONNECTIONS [=] allowconn ]
           [ CONNECTION LIMIT [=] connlimit ]
           [ IS_TEMPLATE [=] istemplate ]
           [ OID [=] oid ]
```

DDL

CREATE DATABASE POSTGRESQL

```
CREATE DATABASE "DBSCNPUB"
    WITH
    OWNER = postgres
    ENCODING = 'UTF8'
    LC_COLLATE = 'Italian_Italy.1252'
    LC_CTYPE = 'Italian_Italy.1252'
    TABLESPACE = pg_default
    CONNECTION LIMIT = -1
    IS_TEMPLATE = False;
```



Only the database owner or a superuser can create a database; non-superuser owners must also have the CREATEDB privilege

DDL

DROP DATABASE

The **DROP DATABASE** statement is used to drop an existing SQL database

SINTAX

```
DROP DATABASE databasename;
```

DDL

DROP DATABASE POSTGRESQL

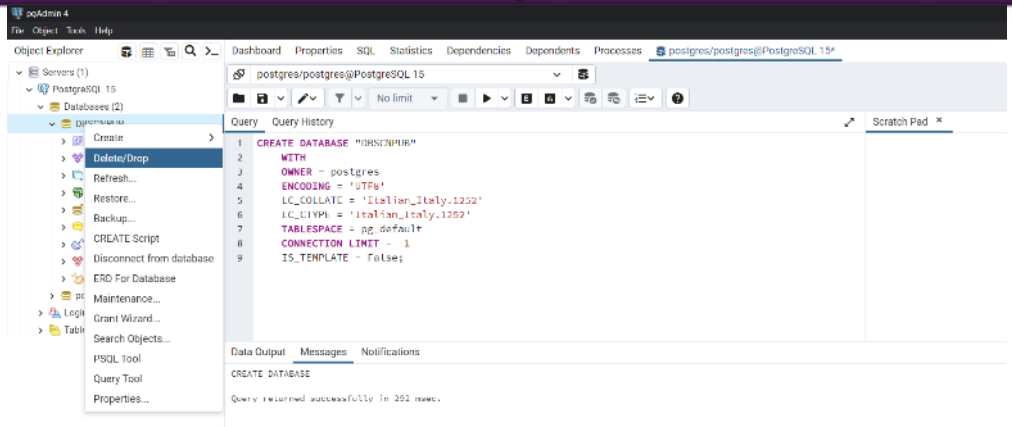
DROP DATABASE statements removes a database

```
DROP DATABASE [ IF EXISTS ] name [ [ WITH ] ( option [, ...] ) ]

where option can be:
    FORCE
```

DDL

DROP DATABASE POSTGRESQL

```
DROP DATABASE "DBSCNPUB"
```



Only the database owner or a superuser can create a database; non-superuser owners must also have the CREATEDB privilege

DDL

ALTER DATABASE

The **ALTER DATABASE** statement changes the attributes of a database

SINTAX

```
ALTER {DATABASE | SCHEMA} [db_name] alter_option ...

alter_option: {
[DEFAULT] CHARACTER SET [=] charset_name |
[DEFAULT] COLLATE [=] collation_name |
[DEFAULT] ENCRYPTION [=] {'Y' | 'N'} |
READ ONLY [=] {DEFAULT | 0 | 1}
}
```

DDL

ALTER DATABASE POSTGRESQL

ALTER DATABASE changes the name of the database

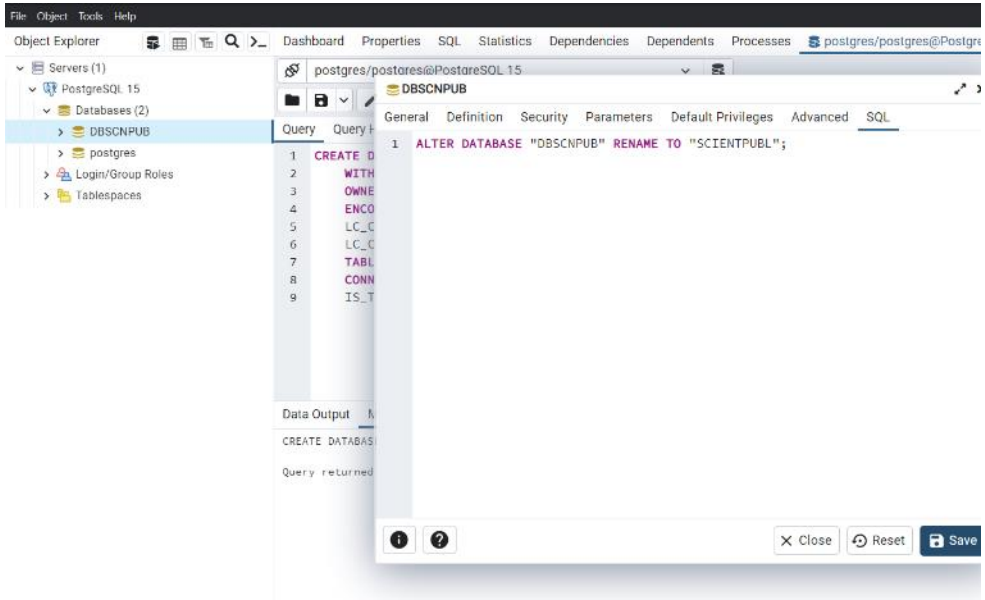
```
ALTER DATABASE name [ [ WITH ] option [ ... ] ]

where option can be:

    ALLOW_CONNECTIONS allowconn
    CONNECTION LIMIT connlimit
    IS_TEMPLATE istemplate
```

ALTER DATABASE name RENAME TO new_name
ALTER DATABASE name OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
ALTER DATABASE name SET TABLESPACE new_tablespace
ALTER DATABASE name REFRESH COLLATION VERSION
ALTER DATABASE name SET configuration_parameter { TO | = } { value | DEFAULT }
ALTER DATABASE name SET configuration_parameter FROM CURRENT
ALTER DATABASE name RESET configuration_parameter
ALTER DATABASE name RESET ALL

DDL

ALTER DATABASE POSTGRESQL

ALTER DATABASE " DBSCNPUB" RENAME TO " SCIENTPUBL "



Only the database owner or a superuser can create a database; non-superuser owners must also have the CREATEDB privilege

DDL

BACKUP DATABASE

The **BACKUP DATABASE** statement is used in SQL Server to create a full back up of an existing SQL database

SINTAX

```
BACKUP DATABASE databasename
TO DISK='filepath';
```

DDL

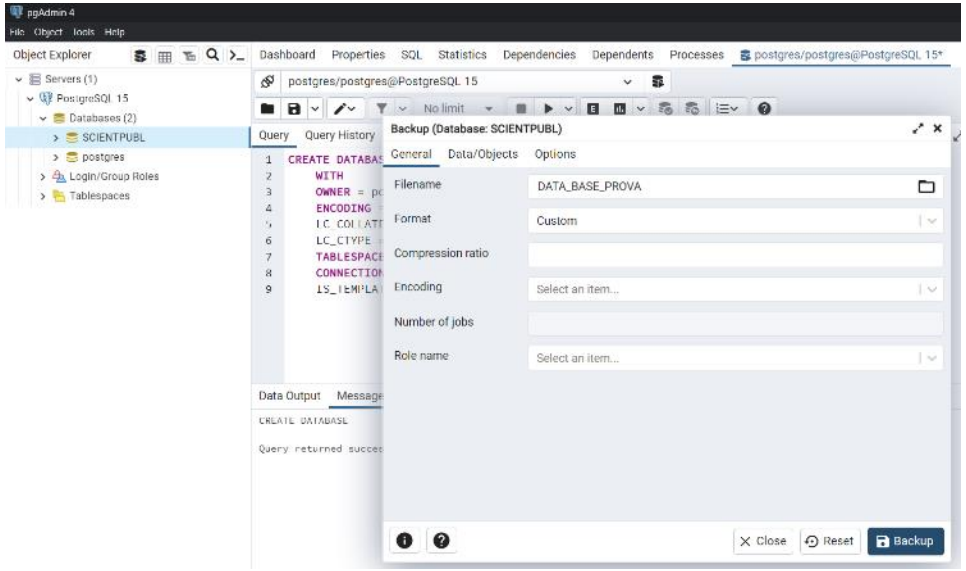
BACKUP DATABASE POSTGRESQL

BACKUP

pg_dump dbname > dumpfile

RESTORE

pg_dump dbname < dumpfile



Only the database owner or a superuser can create a database; non-superuser owners must also have the CREATEDB privilege

DDL

CREATE TABLE

The **CREATE TABLE** statement is used to create a new table in a database

SINTAX

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

DDL

CREATE TABLE

Define a simple table:

```
CREATE TABLE films (
    code            char(5) CONSTRAINT firstkey PRIMARY KEY,
    title           varchar(40) NOT NULL,
    did             integer NOT NULL,
    date_prod       date,
    kind            varchar(10),
    len             interval hour to minute
);
```

Define a check column constraint:

```
CREATE TABLE distributors (
    did     integer CHECK (did > 100),
    name    varchar(40)
);
```

DDL

CREATE TABLE POSTGRESQL

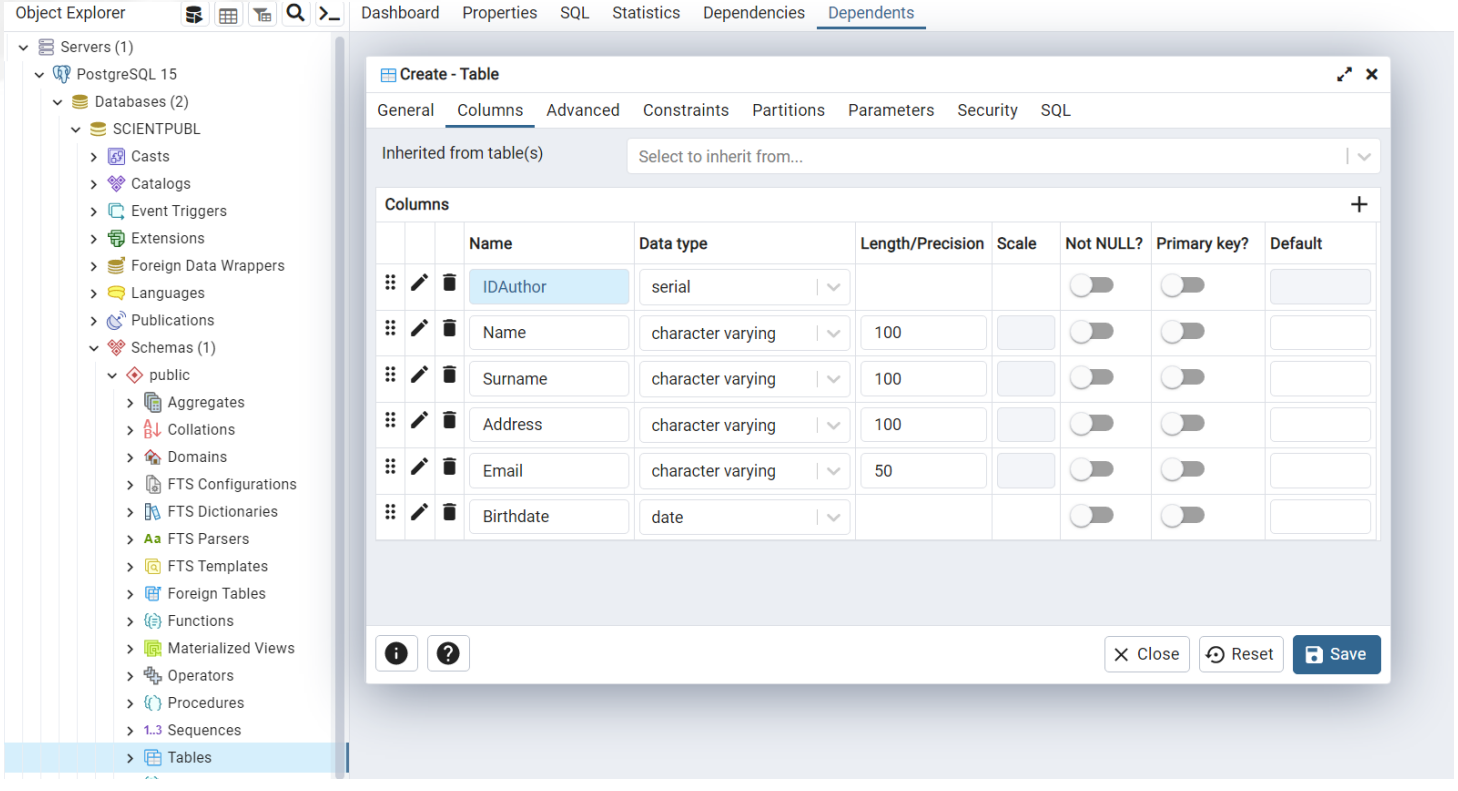
CREATE TABLE defines a new table

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT EXISTS ]
table_name
( [ { column_name data_type
                              [ COMPRESSION compression_method ]
                              [ COLLATE collation ]
                              [ column_constraint [ ... ] ]
                              | table_constraint
                              | LIKE source_table [ like_option ... ]
      }
    [, ... ]
] )
[ INHERITS ( parent_table [, ... ] ) ]
[ PARTITION BY { RANGE | LIST | HASH } ( { column_name | ( expression ) }
[ COLLATE collation ] [ opclass ] [, ... ] ) ]
[ USING method ]
[ WITH ( storage_parameter [= value] [, ... ] ) | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE tablespace_name ]
```

DDL

CREATE TABLE POSTGRESQL

PgAdmin interface



DDL

DATA TYPE

Each column in a database table is required to have a **name** and a **data type**

An SQL developer **must decide what type of data that will be stored inside each column when creating a table**
The data type is a guideline for SQL to understand what type of data is expected inside of each column, and it also identifies how SQL will interact with the stored data

Note: Data types might have different names in different DBMS. And even if the name is the same, the size and other details may be different!
Always check the documentation!

DDL

DATA TYPE POSTGRES

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
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

DDL

DATA TYPE POSTGRES

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

DDL

DATA TYPE POSTGRES

Name	Storage Size	Description
bytea	1 or 4 bytes plus the actual binary string	variable-length binary string

Name	Storage Size	Description
boolean	1 byte	state of true or false

DDL

DATA TYPE POSTGRES

Name	Storage Size	Description	Low Value	High Value	Resolution
timestamp [(*p*)] [without time zone]	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond
timestamp [(*p*)] with time zone	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond
date	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
time [(*p*)] [without time zone]	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond
time [(*p*)] with time zone	12 bytes	time of day (no date), with time zone	00:00:00+1559	24:00:00-1559	1 microsecond
interval [*fields*] [(*p*)]	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond

DDL

DATA TYPE: DATE (format)

The most difficult part when working with **dates** is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.

In Postgres, **Data formats are**:

DATE - format YYYY-MM-DD
TIME - format: HH:MI:SS
TIMESTAMP - format: YYYY-MM-DD HH:MI:SS

DDL

DATA TYPE: ENUM

Enum types are created using the **CREATE TYPE command** and **ENUM**

For example:

```
CREATE TYPE name_enum AS ENUM (‘elem1', ‘elem2', ‘elem3’);
```

DDL

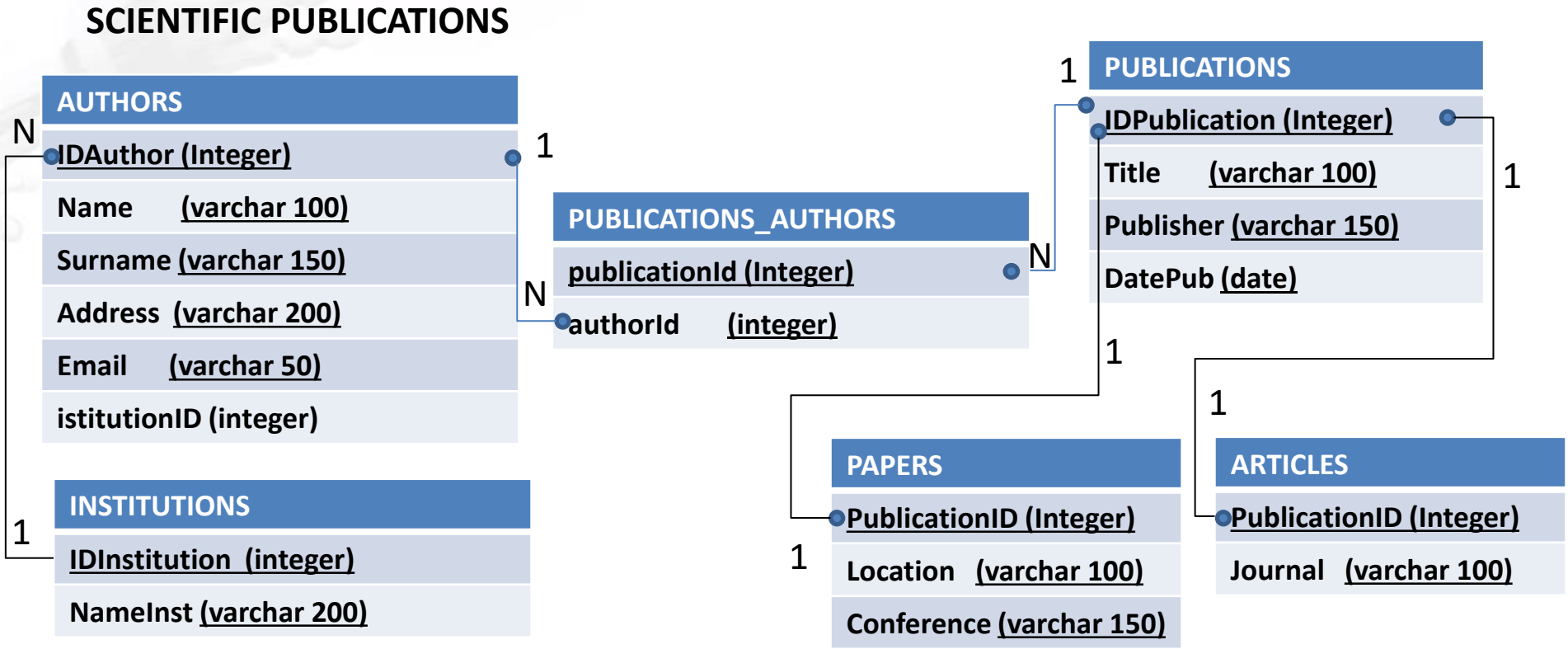
DATA TYPE: ENUM

```
CREATE TYPE mood AS ENUM ('sad', 'ok', 'happy');
CREATE TABLE person (
    name text,
    current_mood mood
);
INSERT INTO person VALUES ('Moe', 'happy');
```

```
SELECT * FROM person WHERE current_mood = 'happy';

 name | current_mood
------+--------------
 Moe  | happy
```

SQL

Sample Database: Relational Model



DDL

CREATE TABLE

```
CREATE TABLE "AUTHORS"
(
    "IDAuthor" integer,
    "Name" character varying(100),
    "Surname" character varying(150),
    "Address" character varying(200),
    "Email" character varying(50),
    "institionID" integer
);
```

```
CREATE TABLE "INSTITUTIONS"
(
    "IDInstitutions" integer,
    "NameInst" character varying(200)
);
```

```
CREATE TABLE "PUBL_AUT"
(
    "publicationID" integer,
    "authorID" integer
);
```

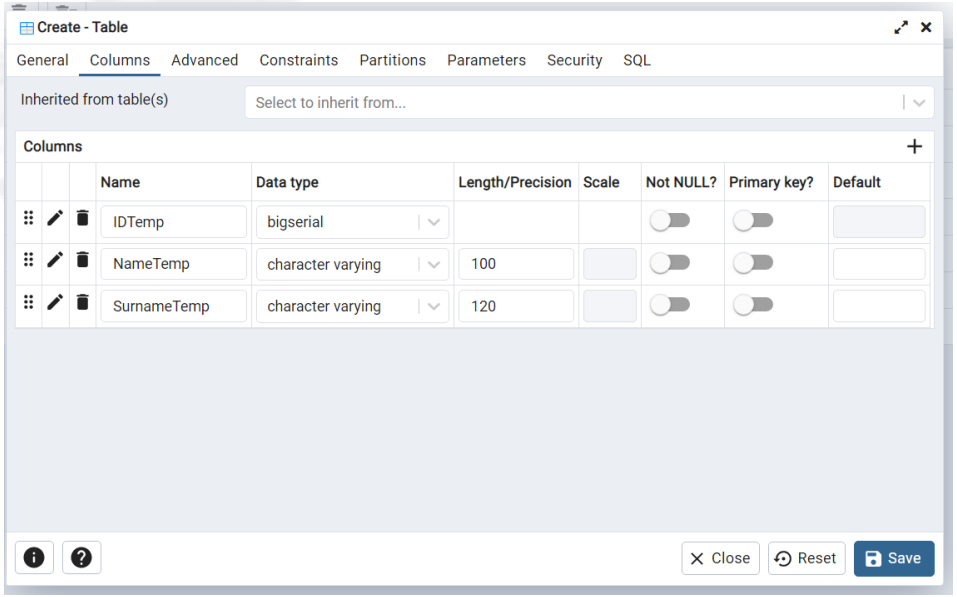
```
CREATE TABLE "PUBLICATIONS"
(
    "IDPublication" integer,
    "Title" character varying(100),
    "Publisher" character varying(150),
    "DatePub" date
);
```

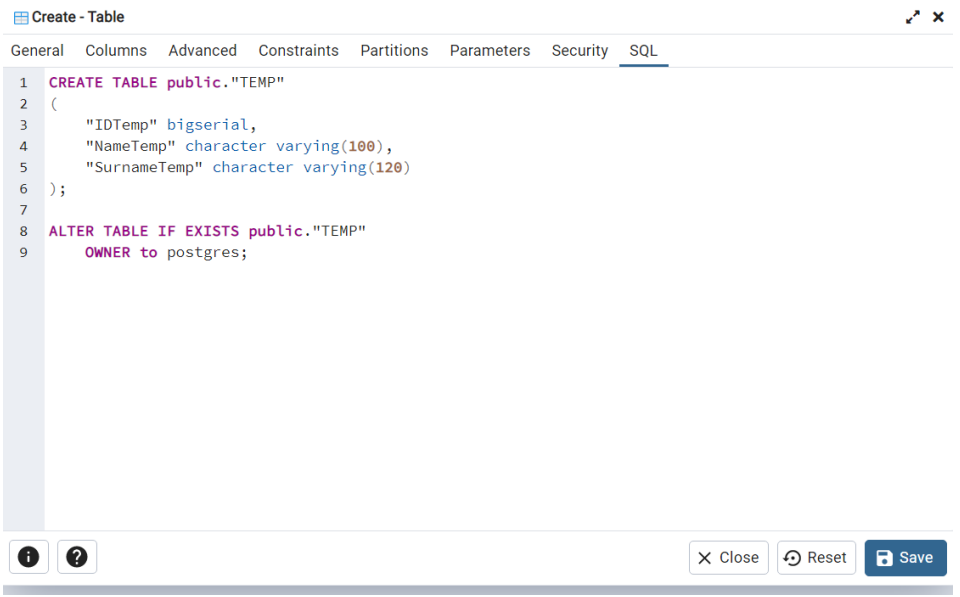
```
CREATE TABLE "PAPERS"
(
    "PublicationID" integer,
    "Location" character varying(100),
     "Conference" character varying(150)
);
```

```
CREATE TABLE "ARTICLES"
(
    "PublicationID" integer,
    “Journal" character varying(100)
);
```

DDL

CREATE TABLE (PgAdmin)





DDL

ALTER TABLE

The **ALTER TABLE** statement is used to add, delete, or modify columns in an existing table

The ALTER TABLE statement is also used to add and drop various constraints on an existing table

SINTAX

```
ALTER TABLE table_name;
ALTER TABLE table_name ADD column_name;
ALTER TABLE table_name ADD column_name
datatype;
```

DDL

ALTER TABLE POSTGRESQL

ALTER TABLE changes the definition of a table

```
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    action [, ... ]
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME [ COLUMN ] column_name TO new_column_name
ALTER TABLE [ IF EXISTS ] [ ONLY ] name [ * ]
    RENAME CONSTRAINT constraint_name TO new_constraint_name
ALTER TABLE [ IF EXISTS ] name
    RENAME TO new_name
ALTER TABLE [ IF EXISTS ] name
    SET SCHEMA new_schema
ALTER TABLE ALL IN TABLESPACE name [ OWNED BY role_name [, ... ] ]
    SET TABLESPACE new_tablespace [ NOWAIT ]
ALTER TABLE [ IF EXISTS ] name
    ATTACH PARTITION partition_name { FOR VALUES partition_bound_spec | DEFAULT }
ALTER TABLE [ IF EXISTS ] name
    DETACH PARTITION partition_name [ CONCURRENTLY | FINALIZE ]
```

DDL

ALTER TABLE POSTGRESQL (action part I)

```
ALTER [ COLUMN ] column_name [ SET DATA ] TYPE data_type [ COLLATE collation ] [ USING
expression ]
ALTER [ COLUMN ] column_name SET DEFAULT expression
ALTER [ COLUMN ] column_name DROP DEFAULT
ALTER [ COLUMN ] column_name { SET | DROP } NOT NULL
ALTER [ COLUMN ] column_name DROP EXPRESSION [ IF EXISTS ]
ALTER [ COLUMN ] column_name ADD GENERATED { ALWAYS | BY DEFAULT } AS IDENTITY [ (
sequence_options ) ]
ALTER [ COLUMN ] column_name { SET GENERATED { ALWAYS | BY DEFAULT } | SET sequence_option
| RESTART [ [ WITH ] restart ] } [...]
ALTER [ COLUMN ] column_name DROP IDENTITY [ IF EXISTS ]
ALTER [ COLUMN ] column_name SET STATISTICS integer
ALTER [ COLUMN ] column_name SET ( attribute_option = value [, ... ] )
ALTER [ COLUMN ] column_name RESET ( attribute_option [, ... ] )
ALTER [ COLUMN ] column_name SET STORAGE { PLAIN | EXTERNAL | EXTENDED | MAIN }
ALTER [ COLUMN ] column_name SET COMPRESSION compression_method
```

DDL

ALTER TABLE POSTGRESQL (action part III)

```
ADD [ COLUMN ] [ IF NOT EXISTS ] column_name data_type [ COLLATE collation ] [ column_constraint [ ... ] ]
DROP [ COLUMN ] [ IF EXISTS ] column_name [ RESTRICT | CASCADE ]
 ADD table_constraint [ NOT VALID ]
 ADD table_constraint_using_index
 ALTER CONSTRAINT constraint_name [ DEFERRABLE | NOT DEFERRABLE ] [ INITIALLY DEFERRED
 | INITIALLY IMMEDIATE ]
 VALIDATE CONSTRAINT constraint_name
 DROP CONSTRAINT [ IF EXISTS ]  constraint_name [ RESTRICT | CASCADE ]
 DISABLE TRIGGER [ trigger_name | ALL | USER ]
 ENABLE TRIGGER [ trigger_name | ALL | USER ]
 ENABLE REPLICA TRIGGER trigger_name
 ENABLE ALWAYS TRIGGER trigger_name
 DISABLE RULE rewrite_rule_name
 ENABLE RULE rewrite_rule_name
 ENABLE REPLICA RULE rewrite_rule_name
 ENABLE ALWAYS RULE rewrite_rule_name
 DISABLE ROW LEVEL SECURITY
 ENABLE ROW LEVEL SECURITY
 FORCE ROW LEVEL SECURITY
 NO FORCE ROW LEVEL SECURITY
```

DDL

ALTER TABLE POSTGRESQL (action part IV)

```
CLUSTER ON index_name
SET WITHOUT CLUSTER
SET WITHOUT OIDS
SET ACCESS METHOD new_access_method
SET TABLESPACE new_tablespace
SET { LOGGED | UNLOGGED }
SET ( storage_parameter [= value] [, ... ] )
RESET ( storage_parameter [, ... ] )
INHERIT parent_table
NO INHERIT parent_table
OF type_name
NOT OF
OWNER TO { new_owner | CURRENT_ROLE | CURRENT_USER | SESSION_USER }
REPLICA IDENTITY { DEFAULT | USING INDEX index_name | FULL | NOTHING }
```

DDL

CONSTRAIN TABLE

SQL constraints are used to specify rules for the data in a table

Constraints are used to limit the type of data that can go into a table.

This ensures the **accuracy and reliability of the data in the table**

If there is any violation between the constraint and the data action, the **action fails**

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table

DDL

CONSTRAIN TABLE

The following constraints are commonly used in SQL:

- **NOT NULL**: Ensures that a column cannot have a NULL value
- **UNIQUE**: Ensures that all values in a column are different
- **PRIMARY KEY**: A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- **FOREIGN KEY**: Prevents actions that would destroy links between tables
- **CHECK**: Ensures that the values in a column satisfies a specific condition
- **DEFAULT**: Sets a default value for a column if no value is specified
- **CREATE INDEX**: Used to create and retrieve data from the database very quickly

DDL

CONSTRAIN TABLE (NOT NULL)

By default, a column can hold NULL values.

The **NOT NULL constraint** enforces a column to NOT accept NULL values.

This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

EXAMPLE (CREATING DATABASE)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

EXAMPLE (ALTERING TABLE)

```
ALTER TABLE Persons
ALTER COLUMN Age int NOT NULL;
```

DDL

CONSTRAINT NOT NULL

```
ALTER TABLE "AUTHORS"
ALTER COLUMN "Name" SET NOT
NULL;

ALTER TABLE "AUTHORS"
ALTER COLUMN "Surname" SET NOT
NULL;
```

```
ALTER TABLE "PUBLICATIONS"
ALTER COLUMN "Publisher" SET NOT
NULL;
```

```
ALTER TABLE "PAPERS"
ALTER COLUMN " Conference " SET
NOT NULL;
```

```
ALTER TABLE "ARTICLES"
ALTER COLUMN "Journal" SET NOT
NULL;
```

```
ALTER TABLE "INSTITUTIONS"
ALTER COLUMN "NameInst" SET NOT
NULL;
```

DDL

CONSTRAIN TABLE (UNIQUE)

The **UNIQUE constraint** ensures that all values in a column are different

Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns

A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, you can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table

EXAMPLE (CREATING DATABASE)

```
CREATE TABLE Persons (
    ID int NOT NULL UNIQUE,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

EXAMPLE (ALTERING TABLE)

```
ALTER TABLE Persons ADD UNIQUE (ID);

ALTER TABLE Persons
ADD CONSTRAINT UC_Person UNIQUE (ID,LastName);
```

DDL

CONSTRAIN TABLE (PRIMARY KEY)

The **PRIMARY KEY** constraint uniquely identifies each record in a table

Primary keys must contain UNIQUE values, and cannot contain NULL values

A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields)

EXAMPLE (CREATING DATABASE)

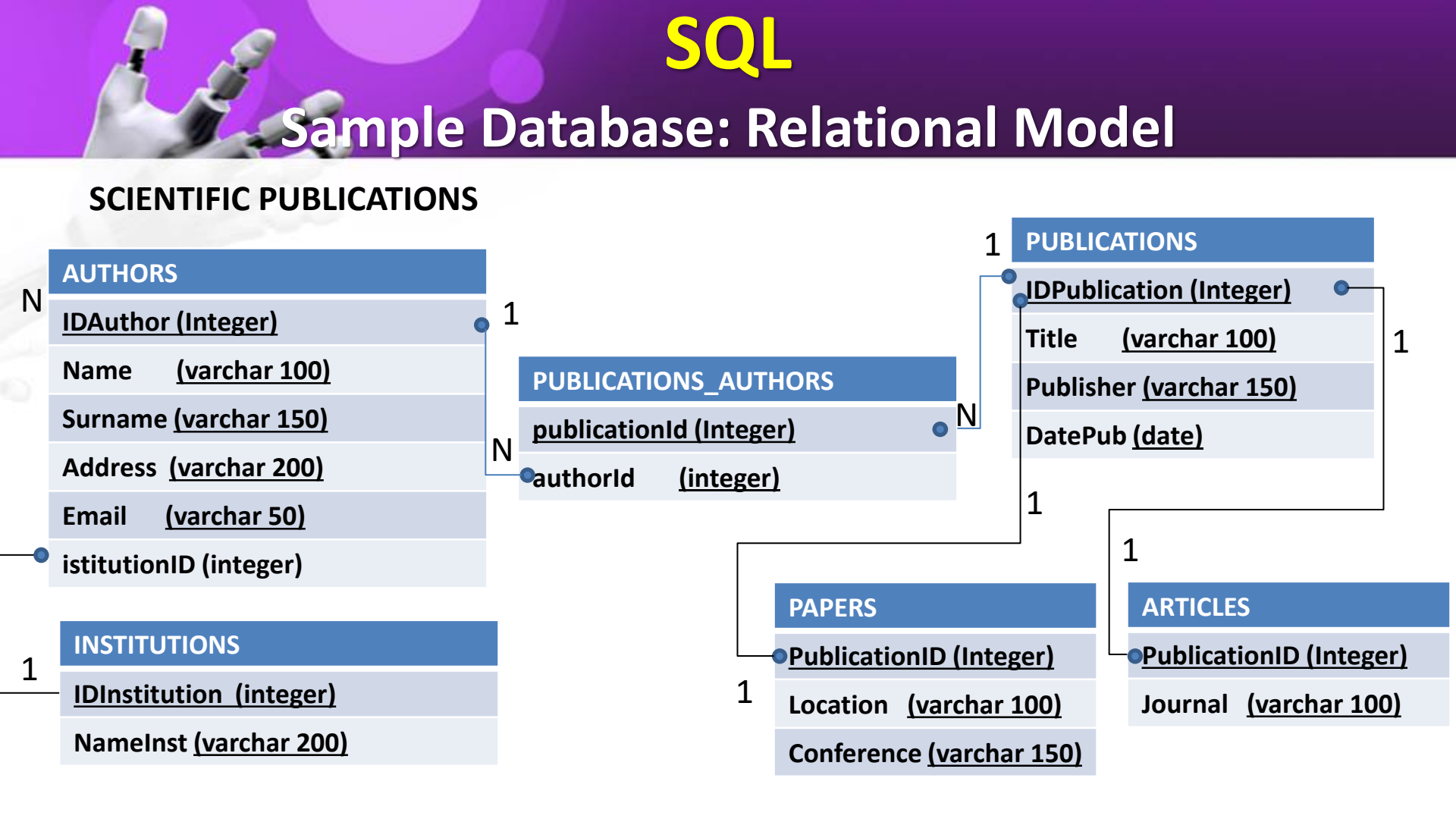
```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

EXAMPLE (ALTERING TABLE)

```
ALTER TABLE Persons ADD PRIMARY KEY (ID);
```

SQL

Sample Database: Relational Model



DDL

CONSTRAIN TABLE: PRIMARY KEY

```
ALTER TABLE "AUTHORS" ADD PRIMARY KEY ("IDAuthor");
```

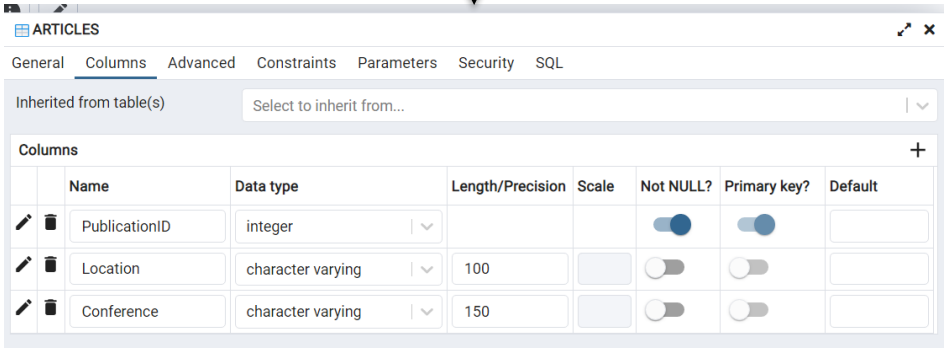
```
ALTER TABLE "PUBL_AUT" ADD PRIMARY KEY ("publicationID");
```

```
ALTER TABLE "PUBLICATIONS" ADD PRIMARY KEY ("IDPublication");
```

```
ALTER TABLE "INSTITUTIONS"  ADD PRIMARY KEY ("IDInstitutions");
```

```
ALTER TABLE "PAPERS" ADD PRIMARY KEY ("PublicationID");
```

```
ALTER TABLE "ARTICLES" ADD PRIMARY KEY ("PublicationID");
```



DDL

CONSTRAIN TABLE (FOREING KEY)

The **FOREIGN KEY constraint** is used to prevent actions that would destroy links between tables

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table

The table with the foreign key is called the **child table**, and the table with the primary key is called the referenced or **parent table**.

DDL

CONSTRAIN TABLE (FOREING KEY-example)

PERSONS

PersonID	Name	Surname	Age
1	Carl	Biden	34
2	Emma	Regan	22
3	Pria	Stallone	16
4	Mason	De Niro	56

EXAMPLE (CREATING DATABASE)

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

ORDERS

OrderID	OrderNumber	PersonID
1	778876	2
2	554466	2
3	456435	1
4	356433	3

EXAMPLE (ALTERING TABLE)

```
ALTER TABLE Orders
ADD FOREIGN KEY (PersonID) REFERENCES
Persons(PersonID);
```

DDL

CONSTRAIN TABLE (FOREING KEY)

In PostgreSQL, you define a foreign key using the **FOREIGN KEY** constraint

The FOREIGN KEY constraint helps maintain the referential integrity of data between the child and parent tables

A FOREIGN KEY constraint indicates that values in a column or a group of columns in the child table equal the values in a column or a group of columns of the parent table

```
[CONSTRAINT fk_name]
        FOREIGN KEY(fk_columns)
                REFERENCES parent_table(parent_key_columns)
                        [ON DELETE delete_action]
                        [ON UPDATE update_action]
```

DDL

CONSTRAIN TABLE (FOREING KEY)

```
ALTER TABLE "PUBL_AUT"
    ADD CONSTRAINT "AuthorID" FOREIGN KEY ("authorID")
    REFERENCES "AUTHORS" ("IDAuthor");
```

```
ALTER TABLE "PUBL_AUT"
    ADD CONSTRAINT "PublicationID" FOREIGN KEY ("publicationID")
    REFERENCES "PUBLICATIONS" ("IDPublication");
```

DDL

CONSTRAIN TABLE (CHECK)

The **CHECK constraint** is used to limit the value range that can be placed in a column

If you define a CHECK constraint on a column it will allow only certain values for this column

If you define a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row

EXAMPLE (CREATING DATABASE)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

EXAMPLE (ALTERING TABLE)

```
ALTER TABLE Persons
ADD CONSTRAINT
CHECK (Age>=18 );
```

DDL

CONSTRAIN TABLE (CHECK)

```
ALTER TABLE "PUBLICATIONS"
   ADD CONSTRAINT "DatePub" CHECK ("DatePub">'1800-01-01');
```

Query | Query History

```
1  INSERT INTO "PUBLICATIONS" VALUES(1,'Fondamneti di Linguaggio R', 'StreetLib','1789-01-01');
```

Data Output | Notifications | Messages

```
ERROR:  La riga in errore contiene (1, Fondamneti di Linguaggio R, StreetLib, 1789-01-01).la nuova riga per la
relazione "PUBLICATIONS" viola il vincolo di controllo "DatePub"

ERRORE:  la nuova riga per la relazione "PUBLICATIONS" viola il vincolo di controllo "DatePub"
SQL state: 23514
Detail: La riga in errore contiene (1, Fondamneti di Linguaggio R, StreetLib, 1789-01-01).
```

DDL

CONSTRAIN TABLE (DEFAULT)

The **DEFAULT constraint** is used to set a default value for a column.

The default value will be added to all new records, if no other value is specified

EXAMPLE (CREATING DATABASE)

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);
```

EXAMPLE (ALTERING TABLE)

```
ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';
```

DDL

CONSTRAIN TABLE (DEFAULT)

```
ALTER TABLE "PAPERS"
    ALTER COLUMN "Location" SET DEFAULT Rome;
```

```
INSERT INTO "PAPERS" VALUES(1, 'Catania', 'Giornata delle Scienze Applicate');
```

```
INSERT INTO "PAPERS" VALUES(2,DEFAULT, 'Giornata delle Scienze Matematiche');
```

Data Output | Notifications | Messages

	PublicationID [PK] integer	Location character varying	Conference character varying
1	1	Catania	Giornata delel Scienze Applicate
2	2	Rome	Giornata delle Scienze Matematiche

DDL

CONSTRAIN TABLE (INDEX)

The **CREATE INDEX statement** is used to create indexes in tables

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

Note: Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

EXAMPLE (CREATING INDEX)

```
CREATE INDEX index_name ON table_name (column1, column2, ...);
```

DDL

CONSTRAIN TABLE (AUTO INCREMENTAL)

Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table

Often this is the primary key field that we would like to be created automatically every time a new record is inserted

EXAMPLE (CREATING INDEX)

```
CREATE TABLE Persons (
    Personid int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (Personid)
);
```

DDL

CONSTRAIN TABLE (SERIAL)

```
ALTER TABLE "PUBL_AUT“ ADD COLUMN "IDPA" serial;
```

DDL

DROP TABLE

The **DROP TABLE** statement is used to drop an existing table in a database

SINTAX

```
DROP TABLE table_name;
```

DDL

DROP TABLE POSTGRESQL

DROP TABLE removes a table

```
DROP TABLE [ IF EXISTS ] name
[, ...]
[ CASCADE | RESTRICT ]
```

End