Unit 5: Creating databases and tables.

2022/2023

Contents

5.1. Introduction (I).

- **SEQUEL** (Structured English Query Language), **developed by IBM**, was designed to manipulate and retrieve data stored in IBM's original quasi-relational database management system.
- Renamed **Structured Query Language** (**SQL**), because "SEQUEL" was a registered trademark.
- By 1986, **ANSI and ISO standard groups** officially adopted the standard "**Database Language SQL**" language definition. New versions of the standard were published in 1989, 1992, 1996, 1999, 2003, 2006, 2008, 2011, and **2016**.

5.1. Introduction (II).

- **Commercial systems** offer most, but not all, **SQL** features, plus varying proprietary feature sets: Not all examples of the slides may work on a particular system.

And many more!!

5.1. Introduction (III).

- PostgreSQL is an **object-relational database management system (ORDBMS)** based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department.

- PostgreSQL is an **open-source** descendant of this original Berkeley code. It supports a large part of the SQL standard and offers many modern features:

complex queries	foreign keys	triggers	updatable views
transactional integrity	procedural language	concurrency control	domains

5.1. Introduction (IV).

- **"MariaDB** is an open source relational database management system (DBMS) that is a compatible drop-in replacement for the widely used MySQL database technology. It was created as a software **fork of MySQL** by developers who played key roles in building the original database; they devised MariaDB in **2009** in response to Oracle Corp.'s acquisition of MySQL". (TechTarget).

- MariaDB supports a large part of the SQL standard and offers many modern features:

  - complex queries
  - foreign keys
  - triggers
  - updatable views
  - transactional integrity
  - procedural language
  - concurrency control

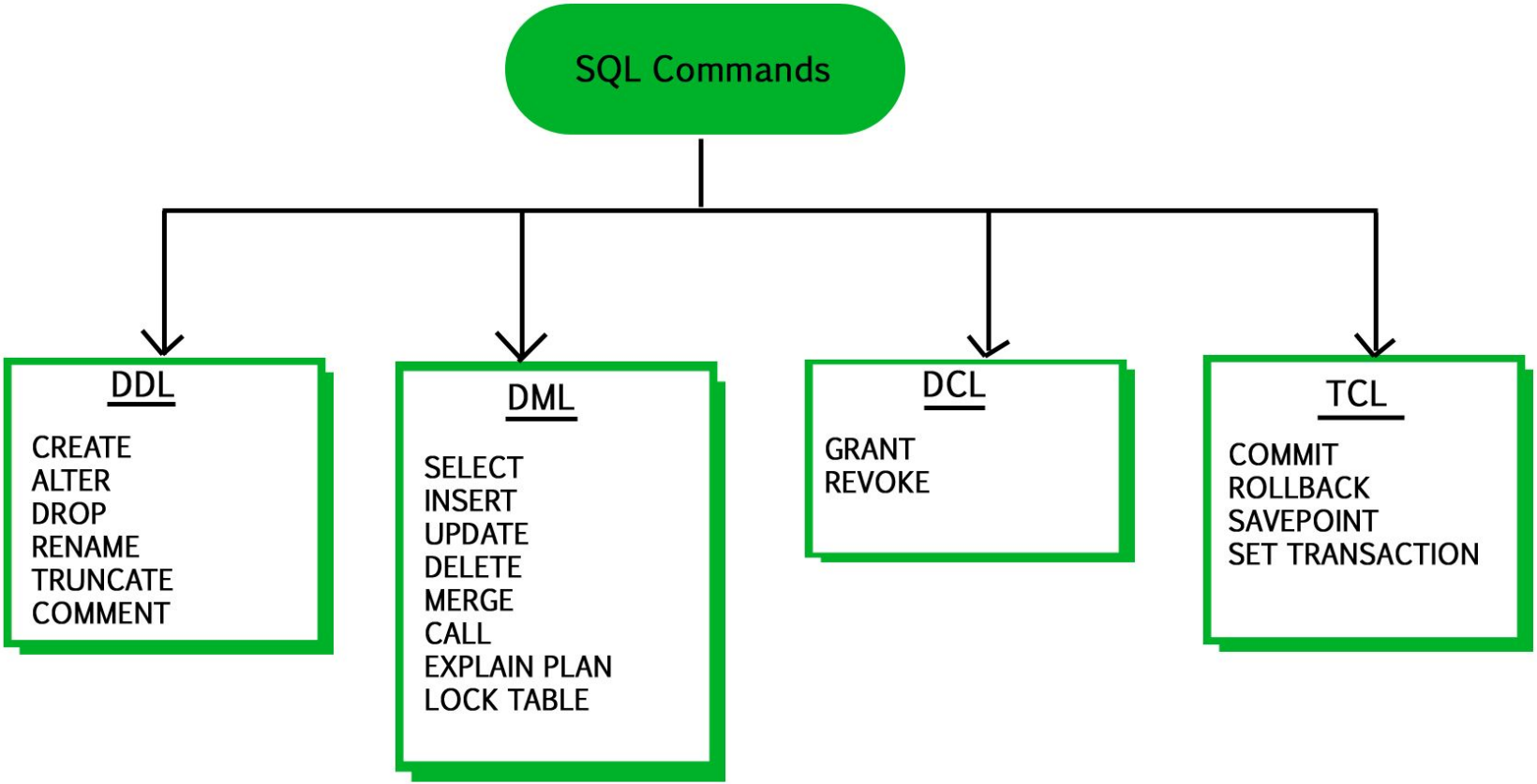
5.1. Introduction (IV).

- MariaDB and PostgreSQL: Because of the **liberal license**, they can be used, modified, and distributed by anyone free of charge for any purpose, be it private, commercial, or academic.

- PostgreSQL is released under the PostgreSQL License.

- The MariaDB server is available under the terms of the GNU General Public License, version 2.
- The GNU project manteins an official page with information about the GNU GPL 2 license, including a FAQ and various translations.

5.1. Introduction (III).



5.2. Data Definition Language (DDL).

DDL (Data Definition Language): Part of SQL to define the **database schema**.

Statement	Function
CREATE DATABASE	Creates a new database and the file used to store the database.
DROP DATABASE	Removes an existing database.
CREATE TABLE	Creates a new table.
DROP TABLE	Removes a table definition and all data, indexes, and constraints for that table.
ALTER TABLE	Modifies a table definition by altering, adding, or dropping columns and constraints.
CREATE INDEX	Creates an index on a given table.
DROP INDEX	Removes one or more indexes from the current database.
Also…	ALTER DATABASE, CREATE DOMAIN, CREATE TYPE, and others!

5.3. Creating databases.

CREATE DATABASE db_name;

Creating a database **does not select it** for use:

MariaDB

```
USE db_name;
SELECT * FROM db_name.table_name;
```

PostgreSQL

```
\c db_name
```

MariaDB

```
CREATE [OR REPLACE] {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_specification] ...

create_specification:
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'comment'
```

PostgreSQL

```
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 ]
```

PostgreSQL

createdb

createdb — create a new PostgreSQL database

Synopsis

createdb [*connection-option*...] [*option*...] [*dbname* [*description*]]

Description

createdb creates a new PostgreSQL database.

Normally, the database user who executes this command becomes the owner of the new database. However, a different owner can be specified via the `-O` option, if the executing user has appropriate privileges.

createdb is a wrapper around the SQL command `CREATE DATABASE`. There is no effective difference between creating databases via this utility and via other methods for accessing the server.

5.3. Creating databases.

MariaDB

```
MariaDB [(none)]> CREATE DATABASE company;
Query OK, 1 row affected (0,000 sec)

MariaDB [(none)]> USE company;
Database changed
MariaDB [company]> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| billing            |
| company            |
| information_schema |
| mysql              |
| performance_schema |
| phpmyadmin         |
| shop               |
| sys                |
+--------------------+
8 rows in set (0,001 sec)
```

PostgreSQL

```
postgres=# CREATE DATABASE company;
CREATE DATABASE
postgres=# \c company
You are now connected to database "company" as user "postgres".
company=# \l
                                  List of databases
   Name    |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
-----------+----------+----------+-------------+-------------+-----------------------
 company   | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 postgres  | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 shop      | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 template0 | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
 template1 | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 | =c/postgres          +
           |          |          |             |             | postgres=CTc/postgres
(5 rows)
```

5.3. Creating databases.

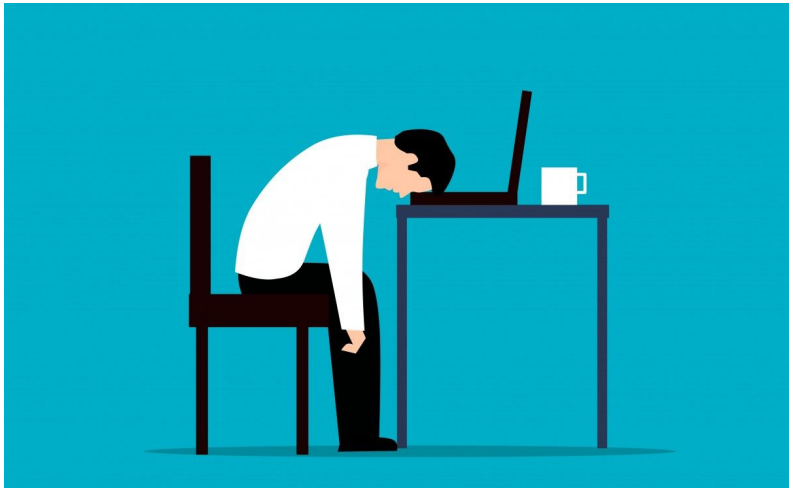
```
sergi@dragonfruit:~$ sudo su postgres
[sudo] contraseña para sergi:
postgres@dragonfruit:/home/sergi$ createdb hola
postgres@dragonfruit:/home/sergi$ psql
psql (14.5 (Ubuntu 14.5-1ubuntu1))
Type "help" for help.

postgres=# \l
                                  List of databases
      Name      |  Owner   | Encoding |   Collate   |    Ctype    |   Access privileges
----------------+----------+----------+-------------+-------------+-----------------------
 company        | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 freaky_league  | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 gonzalezsergio | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 hola           | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 kk             | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 miguelangel    | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 postgres       | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 practice05     | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 shop           | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
 template0      | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 | =c/postgres          +
                |          |          |             |             | postgres=CTc/postgres
 template1      | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 | =c/postgres          +
                |          |          |             |             | postgres=CTc/postgres
 webshop        | postgres | UTF8     | es_ES.UTF-8 | es_ES.UTF-8 |
(12 rows)
```

5.4. Dropping databases.

```
DROP DATABASE db_name;
```

https://mariadb.com/kb/en/drop-database/
https://www.postgresql.org/docs/current/sql-dropdatabase.html
https://www.postgresql.org/docs/current/app-createdb.html



5.5. Users and roles (basic authorization).

Roles in MariaDB: "A role bundles a number of privileges together. It assists larger organizations where, typically, a number of users would have the same privileges, and, previously, the only way to change the privileges for a group of users was by changing each user's privileges individually". Source:https://mariadb.com/kb/en/roles_overview/

PostgreSQL

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]

where option can be:

      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT connlimit
    | [ ENCRYPTED ] PASSWORD 'password' | PASSWORD NULL
    | VALID UNTIL 'timestamp'
    | IN ROLE role_name [, ...]
    | IN GROUP role_name [, ...]
    | ROLE role_name [, ...]
    | ADMIN role_name [, ...]
    | USER role_name [, ...]
    | SYSID uid
```

MariaDB

```
CREATE [OR REPLACE] USER [IF NOT EXISTS]
 user_specification [,user_specification ...]
  [REQUIRE (NONE | tls_option [[AND] tls_option ...] )]
  [WITH resource_option [resource_option ...] ]
  [lock_option] [password_option]

user_specification:
  username [authentication_option]

authentication_option:
  IDENTIFIED BY 'password'
  | IDENTIFIED BY PASSWORD 'password_hash'
  | IDENTIFIED {VIA|WITH} authentication_rule [OR authentication_rule  ...]

authentication_rule:
    authentication_plugin
  | authentication_plugin {USING|AS} 'authentication_string'
  | authentication_plugin {USING|AS} PASSWORD('password')

tls_option:
  SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'

resource_option:
  MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
  | MAX_STATEMENT_TIME time

password_option:
  PASSWORD EXPIRE
  | PASSWORD EXPIRE DEFAULT
  | PASSWORD EXPIRE NEVER
  | PASSWORD EXPIRE INTERVAL N DAY

lock_option:
    ACCOUNT LOCK
  | ACCOUNT UNLOCK
}
```

5.5. Users and roles (basic authorization).

createuser

Prev Up PostgreSQL Client Applications

createuser

createuser — define a new PostgreSQL user account

Synopsis

```
createuser [connection-option...] [option...] [username]
```

5.5. Users and roles (basic authorization).

PostgreSQL

```
postgres=# CREATE USER alumne WITH PASSWORD 'alualualu';
CREATE ROLE
postgres=# CREATE ROLE alumne WITH PASSWORD 'alualualu';
ERROR:  role "alumne" already exists
```

MariaDB

```
MariaDB [(none)]> CREATE USER alumne IDENTIFIED BY 'alualualu';
Query OK, 0 rows affected (0,005 sec)
```

PostgreSQL

To create the user joe as a superuser, and assign a password immediately:

```
postgres@dragonfruit:/home/sergi$ createuser -P -s -e joe
Enter password for new role:
Enter it again:
SELECT pg_catalog.set_config('search_path', '', false);
CREATE ROLE joe PASSWORD 'SCRAM-SHA-256$4096:P+U6VNoghIDLVbx+ok5WUw==$3lWu6JSSTMBy3X7NRoUQI/uI16yQjn7Eqj2VSXI4e2U=:ndi/oU5aCjuZj7zmutgOaqVRnlensQFiFrpsSAlMUHU=' SUPER
USER CREATEDB CREATEROLE INHERIT LOGIN;
```

5.5. Users and roles (basic authorization).

PostgreSQL

```
postgres=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 alumne    |                                                            | {}
 joe       | Superuser, Create role, Create DB                          | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}
```

MariaDB

```
MariaDB [(none)]> SELECT User FROM mysql.user;
+-------------+
| User        |
+-------------+
| alumne      |
| mariadb.sys |
| mysql       |
| root        |
+-------------+
4 rows in set (0,001 sec)
```

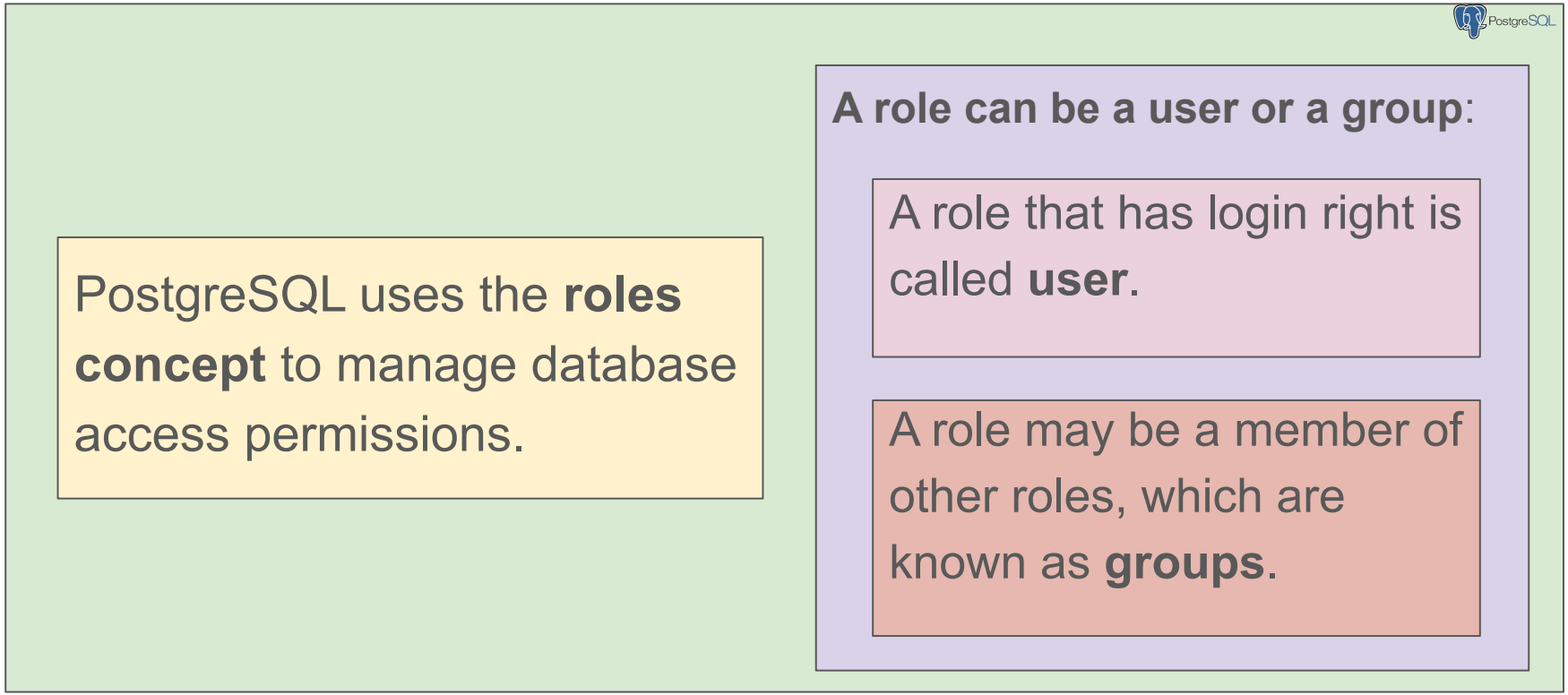
5.5. Users and roles (basic authorization).

```
postgres=# SELECT usename AS role_name,
 CASE
  WHEN usesuper AND usecreatedb THEN
    CAST('superuser, create database' AS pg_catalog.text)
  WHEN usesuper THEN
    CAST('superuser' AS pg_catalog.text)
  WHEN usecreatedb THEN
    CAST('create database' AS pg_catalog.text)
  ELSE
    CAST('' AS pg_catalog.text)
 END role_attributes
FROM pg_catalog.pg_user
ORDER BY role_name desc;
 role_name |      role_attributes
-----------+---------------------------
 postgres  | superuser, create database
 joe       | superuser, create database
 alumne    |
(3 rows)
```

5.5. Users and roles (basic authorization).

- **Example 1:**
  - To create a new user “user1” to connect to MySQL/MariaDB from the **localhost** with the password “alu01”:
  - CREATE USER 'user1'@'localhost' IDENTIFIED BY 'alu01';
- **Example 2:**
  - To create a new user “user1” to connect to MySQL/MariaDB from from **any host** with the password “alu02”:
  - CREATE USER 'user1'@'%' IDENTIFIED BY 'alu01';
- **Example 3:**
  - To create two new users “user2” and “user3” **with a single statement** to connect to MySQL/MariaDB from the localhost with the password “alu02” and “alu03”:
  - CREATE USER
    'user2'@'localhost' IDENTIFIED BY 'alu02',
    'user3'@'localhost' IDENTIFIED BY 'alu03';

5.5. Users and roles (basic authorization).



5.5. Users and roles (basic authorization).

- **Example 1:**
  - Create a role with a password:
  - CREATE USER davide WITH PASSWORD 'jw8s0F4';
- **Example 2:**
  - Create a role with a password that is valid until the end of 2004. After one second has ticked in 2005, the password is no longer valid:
  - CREATE ROLE miriam WITH LOGIN PASSWORD 'jw8s0F4' VALID UNTIL '2025-01-01';
- **Example 3:**
  - Create a role that can create databases and manage roles:
    CREATE ROLE admin WITH CREATEDB CREATEROLE;

5.5. Users and roles (basic authorization).

We have already seen how to create user using the create user statement, but that statement **only creates a new user but does not grant any privileges to the user account**.

To grant privileges to a user account, the GRANT statement is used.

Syntax:

- GRANT privileges_names ON object TO user_list;

5.5. Users and roles (basic authorization).

Privilege Name	Description
SELECT	Execute select statement on tables.
INSERT	Execute insert statement on tables.
UPDATE	Execute update statement on tables.
DELETE	Execute delete statement on tables.
INDEX	Create an INDEX on an existing table.
CREATE	Execute CREATE TABLE statements.
ALTER	Perform ALTER TABLE on table.
DROP	Execute DROP TABLE statements.
GRANT OPTION	Grant privileges to other users.
ALL	grants all permissions except GRANT OPTION.

5.5. Users and roles (basic authorization).

Revoking privileges from a table:

- The Revoke statement is used to revoke some or all of the privileges which have been granted to a user.

- Syntax:
  - REVOKE privileges ON object FROM user_list;

5.5. Users and roles (basic authorization).

- **Example 1:**
  - Revoking SELECT privilege to “user3” in the table “R1” inside the database “UNIT04” only from localhost:
  - REVOKE SELECT ON UNIT04.R1 FROM 'user3'@'localhost';
- **Example 2:**
  - Revoking SELECT, INSERT, UPDATE, and DELETE privileges to “user3” in the table “R1” inside the database “UNIT04” only from localhost:
  - REVOKE SELECT, INSERT, UPDATE, DELETE ON UNIT04.R1 FROM 'user3'@'localhost';

5.5. Users and roles (basic authorization).

- **Example 3:**
  - Revoking **all** the privileges to user3 in table UNIT04.R1:
  - REVOKE ALL ON UNIT04.R1 FROM 'user3'@'localhost';
- **Example 4:**
  - Revoking only the **GRANT OPTION** to user3 in table UNIT04.R1 with grant option:
  - REVOKE GRANT OPTION ON UNIT04.R1 FROM 'user3'@'localhost';

5.5. Users and roles (basic authorization).

- **Examples:**
  - GRANT group_role TO user_role;
  - REVOKE group_role FROM user_role;
  - GRANT SELECT ON TABLE table_name TO user_role;
  - REVOKE SELECT ON TABLE table_name FROM user_role;
  - GRANT USAGE ON SCHEMA schema_name TO role_name;
  - GRANT ALL PRIVILEGES ON TABLES IN SCHEMA schema_name TO role_name;

5.5. Users and roles (basic authorization).

MariaDB

The “SHOW GRANTS” statement is used to view the permissions of a user account.

The syntax for the SHOW GRANTS is:

- SHOW GRANTS FOR user-account;

Example:

- SHOW GRANTS FOR alumne@'%';

PostgreSQL

\du

```
MariaDB [(none)]> SHOW GRANTS FOR alumne@'%';
+-----------------------------------------------------------------------------------------------------------------+
| Grants for alumne@%                                                                                             |
+-----------------------------------------------------------------------------------------------------------------+
| GRANT ALL PRIVILEGES ON *.* TO 'alumne'@'%' IDENTIFIED BY PASSWORD '*FEBAFAABF3035BF32237F30A0B3D5CF3B6BA7E4C' |
+-----------------------------------------------------------------------------------------------------------------+
1 row in set (0.000 sec)
```

5.5. Users and roles (basic authorization).

You will see all this in a subject in the second course.

MariaDB

- https://mariadb.com/kb/en/grant/
- https://mariadb.com/kb/en/revoke/

PostgreSQL

- https://www.postgresql.org/docs/current/sql-grant.html
- https://www.postgresql.org/docs/current/ddl-priv.html
- https://www.postgresql.org/docs/current/sql-revoke.html

5.6. Data types in SQL.

MariaDB

Name	Description
`char(n)`	Fixed length character string (n: length).
`varchar(n)`	Variable length character strings (n: length).
`text`	Maxim length 2,147,483,647 characters (variable length non-unicode data).
`int`	Others: smallint, longint, ...
`numeric(p,d)`	Fixed point number (p: precision of p digits, d: digits to the decimal point).
`real, double precision`	Floating point and double-precision floating point numbers.
`float(n)`	Floating point number (with precision of at least n digits).

5.6. Data types in SQL.

MariaDB

Name	Description
`blob`	A blob column with a maximum length of about 65535 bytes. While in storage, each of them can use a 2 byte length prefix that indicates the byte quantity in the value. Also: tinyblob, mediumblob, or longblob.
`date`	Dates from 0001-01-01 to 9999-12-31.
`time`	00:00:00.0000000 to 23:59:59.9999999 with 100 nanosecond accuracy.
`datetime`	Data types date and time combined in one.
And many more that we will see in the following units...	
https://mariadb.com/kb/en/data-types/	

5.6. Data types in SQL.

Name	Aliases	Description
`bigint`	`int8`	signed eight-byte integer
`bigserial`	`serial8`	autoincrementing eight-byte integer
`bit[(n)]`		fixed-length bit string
`bit varying[(n)]`	`varbit [ (n) ]`	variable-length bit string
`boolean`	`bool`	logical Boolean (true/false)
`box`		rectangular box on a plane
`bytea`		binary data (“byte array”)
`character[(n)]`	`char [ (n) ]`	fixed-length character string
`character varying[(n)]`	`varchar [ (n) ]`	variable-length character string

5.6. Data types in SQL.

Name	Aliases	Description
`cidr`		IPv4 or IPv6 network address
`circle`		circle on a plane
`date`		calendar date (year, month, day)
`double precision`	`float8`	double precision floating-point number (8 bytes)
`inet`		IPv4 or IPv6 host address
`integer`	`int`, `int4`	signed four-byte integer
`interval[`***`fields`***`][(`***`p`***`)]`		time span
`json`		textual JSON data
`jsonb`		binary JSON data, decomposed

5.6. Data types in SQL.

Name	Aliases	Description
`line`		infinite line on a plane
`lseg`		line segment on a plane
`macaddr`		MAC (Media Access Control) address
`macaddr8`		MAC (Media Access Control) address (EUI-64 format)
`money`		currency amount
`numeric[(p,s)]`	`decimal[(p,s)]`	exact numeric of selectable precision
`path`		geometric path on a plane
`pg_lsn`		PostgreSQL Log Sequence Number

5.6. Data types in SQL.

https://stackoverflow.com/questions/4848964/difference-between-text-and-varchar-character-varying

Name	Aliases	Description
point		geometric point on a plane
polygon		closed geometric path on a plane
real	float4	single precision floating-point number (4 bytes)
smallint	int2	signed two-byte integer
smallserial	serial2	autoincrementing two-byte integer
serial	serial4	autoincrementing four-byte integer
text		variable-length character string

5.6. Data types in SQL.

Name	Aliases	Description
`time[(p)][without time zone]`		time of day (no time zone)
`time[(p)]with time zone`	`timetz`	time of day, including time zone
`timestamp [(p)] [without time zone]`		date and time (no time zone)
`timestamp [(p)] with time zone`	`timestamptz`	date and time, including time zone
`tsquery`		text search query
`tsvector`		text search document
`txid_snapshot`		user-level transaction ID snapshot

5.6. Data types in SQL.

Name	Aliases	Description
`uuid`		universally unique identifier
`xml`		XML data

Compatibility:

The following types (or spellings thereof) are specified by SQL: bigint, bit, bit varying, boolean, char, character varying, character, varchar, date, double precision, integer, interval, numeric, decimal, real, smallint, time (with or without time zone), timestamp (with or without time zone), xml.

More information about data types in PostgreSQL:

https://www.postgresql.org/docs/current/datatype.html

5.6. Data types in SQL.

serial VS auto_increment:

PostgreSQL

```
CREATE TABLE animals (
     id SERIAL NOT NULL,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
 );

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('fox'),('whale'),('ostrich');
```

MariaDB

```
CREATE TABLE animals (
     id AUTO_INCREMENT NOT NULL,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
 );

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('fox'),('whale'),('ostrich');
```

5.6. Data types in SQL.

serial VS auto_increment:

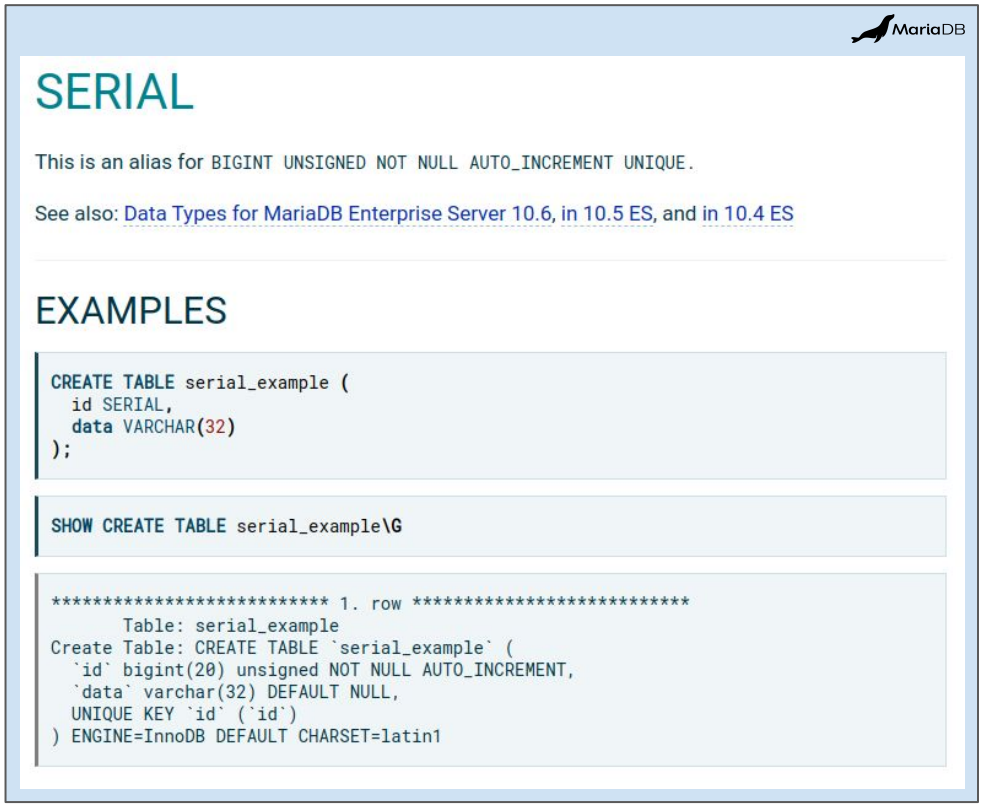
PostgreSQL

```
postgres=# select * from animals;
 id |               name
----+---------------------------------
  1 | dog
  2 | cat
  3 | penguin
  4 | fox
  5 | whale
  6 | ostrich
(6 rows)
```

MariaDB

```
MariaDB [company]> select * from animals;
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | fox     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
6 rows in set (0,000 sec)
```

5.6. Data types in SQL.



MariaDB

SERIAL

This is an alias for `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`.

See also: Data Types for MariaDB Enterprise Server 10.6, in 10.5 ES, and in 10.4 ES

EXAMPLES

```
CREATE TABLE serial_example (
  id SERIAL,
  data VARCHAR(32)
);
```

```
SHOW CREATE TABLE serial_example\G
```

```
*************************** 1. row ***************************
       Table: serial_example
Create Table: CREATE TABLE `serial_example` (
  `id` bigint(20) unsigned NOT NULL AUTO_INCREMENT,
  `data` varchar(32) DEFAULT NULL,
  UNIQUE KEY `id` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Source: https://mariadb.com/docs/skysql/ref/mdb/data-types/SERIAL/

5.6. Data types in SQL.

- In MySQL/MariaDB **cast between data types** is automatic. In Postgresql is not automatic…

- **CAST function**:
  - CAST (expression AS target_type);

Data type: money

```
bookings=# SELECT * FROM facilities WHERE cust_cost = 0;
ERROR:  operator does not exist: money = integer
LINE 1: SELECT * FROM facilities WHERE cust_cost = 0;
                                                 ^
HINT:  No operator matches the given name and argument types. You might need to add explicit type casts.
bookings=# SELECT * FROM facilities WHERE CAST(cust_cost AS numeric) = 0;
 id |      name       | cust_cost | guest_cost | purchase_cost | maintenance_cost
----+-----------------+-----------+------------+---------------+------------------
  2 | Badminton Court |     $0.00 |    $155.00 |     $4,000.00 |           $50.00
  3 | Table Tennis    |     $0.00 |      $5.00 |       $320.00 |           $10.00
  7 | Snooker Table   |     $0.00 |      $5.00 |       $450.00 |           $15.00
  8 | Pool Table      |     $0.00 |      $5.00 |       $400.00 |           $15.00
(4 rows)
```

5.6. Data types in SQL.

- Examples:

1. Cast a string to an integer:
   a. SELECT CAST ('100' AS INTEGER);
2. Cast a string to a date:
   a. SELECT CAST ('2015-01-01' AS DATE), CAST ('01-OCT-2015' AS DATE);
3. Cast a string to a double:
   a. SELECT CAST ('10.2' AS DOUBLE PRECISION);
4. Cast a string to a boolean:
   a. SELECT CAST('true' AS BOOLEAN), CAST('false' as BOOLEAN),
      CAST('T' as BOOLEAN), CAST('F' as BOOLEAN);
5. Convert a string to a timestamp:
   a. SELECT '2019-06-15 14:30:20'::timestamp;

5.6. Data types in SQL.

Domains in PostgreSQL:

Video
Creació de dominis en PostgreSQL (Miquel Boada Artigas).
Doc:
https://www.postgresql.org/docs/current/sql-createdomain.html

Domains are used to specify valid values for a column tables. The difference between a domain and an enum is that with domains we can check using a range of values, a regular expression…

```
CREATE DOMAIN name [ AS ] data_type
    [ COLLATE collation ]
    [ DEFAULT expression ]
    [ constraint [ ... ] ]

where constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK ( expression) }
```

```
CREATE DOMAIN postal_code_IB AS TEXT
CHECK(
   VALUE ~ '^07\d{3}$'
);
```

5.6. Data types in SQL.

Domains in PostgreSQL:

```
CREATE DOMAIN postal_code_IB AS TEXT
CHECK(
   VALUE ~ '^07\d{3}$'
);
```

```
create table customers (
  ID serial primary key,
  fullname varchar(150),
  address varchar(250),
  postal_code postal_code_IB
);
```

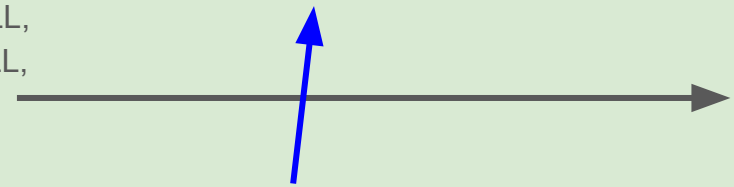
```
postgres=# insert into customers (fullname, address, postal_code) values
('Sergi González', 'Caracas Street, 6', '07006');
INSERT 0 1
postgres=# insert into customers (fullname, address, postal_code) values
('Sergi González', 'Caracas Street, 6', '08006');
ERROR:  value for domain postal_code_ib violates check constraint "postal_code_ib_chec
```

5.6. Data types in SQL.

```
CREATE TABLE myemployees (
    ID SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    salary int,
    CHECK (
        salary > 1000
    )
);
```

```
CREATE DOMAIN contact_name AS
    VARCHAR NOT NULL CHECK (value !~ '\s');
```

```
CREATE TABLE mail_list (
    id serial PRIMARY KEY,
    first_name contact_name,
    last_name contact_name,
    email VARCHAR NOT NULL
);
```



CHECK constraint to ensure values are not null and also do not contain space.

```
insert into myemployees (first_name, last_name, salary)
values ('Sergi', 'González', 500);
```

```
ERROR:  new row for relation "myemployees"
violates check constraint
"myemployees_salary_check"
DETAIL:  Failing row contains (1, Sergi, González,
500).
```

5.6. Data types in SQL.

To change or remove a domain, you use the **ALTER DOMAIN** or **DROP DOMAIN** respectively.

To view all domains in the current database, you use the \dD command as follows:

```
test=#\dD
                                    List of domains
Schema |    Name   |      Type         | Modifier |              Check
--------+-------------+-------------------+----------+-----------------------------------
public | contact_name | character varying | not null | CHECK (VALUE::text !~ '\s'::text)
(1 row)
```

5.6. Data types in SQL.

CREATE TYPE in PostgreSQL:

```
CREATE TYPE film_summary AS (
    film_id INT,
    title VARCHAR,
    release_year YEAR
);
```

```
CREATE OR REPLACE FUNCTION
get_film_summary (f_id INT)
    RETURNS film_summary AS
$$
SELECT film_id,  title, release_year
FROM film
WHERE film_id = f_id ;
$$
LANGUAGE SQL;
```

```
SELECT *
FROM  get_film_summary (1234);
```

film_id	title	release_year
1234	Cinema Paradiso	1988

5.6. Data types in SQL.

To change a type or remove a type, you use the **ALTER TYPE** or **DROP TYPE** statement respectively.

The command for listing all user-defined types in the current database is \dT or \dT+:

```
dvdrental=# \dT
          List of data types
 Schema |     Name      | Description
--------+---------------+-------------
 public | film_summary  |
 public | mpaa_rating   |
 public | year          |
(3 rows)
```

5.6. Data types in SQL.

ENUM types in PostgreSQL:

Video
Creació de dominis en PostgreSQL (Miquel Boada Artigas).
Doc:
https://www.postgresql.org/docs/current/sql-createdomain.html

```
CREATE TYPE name AS ENUM
    ( [ 'label' [, ... ] ] )
```

Using ENUM types we can limit the valid values for a data column.

```
CREATE TYPE tp_userType AS ENUM (
        'Administrator', 'Writer', 'Reader'
);
CREATE TABLE tp_user (
        login       varchar(20),
        name        varchar(100),
        email       varchar(255),
        userType    tp_userType,
        password    char(60),
        primary key (login)
);
```

5.6. Data types in SQL.

ENUM types in MariaDB:

Doc: https://mariadb.com/kb/en/enum/

```
CREATE TABLE fruits (
  id INT NOT NULL auto_increment PRIMARY KEY,
  fruit ENUM('apple','orange','pear'),
  bushels INT);
```

```
ENUM('value1','value2',...) [CHARACTER SET charset_name] [COLLATE collation_name]
```

5.7. Creating tables.

An SQL relation is defined using the create table command:

create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
(integrity-constraint$_1$),
...,
(integrity-constraint$_k$))

r is the name of the relation

each A_i is an attribute name in the schema of relation r

D_i is the data type of values in the domain of attribute A_i

5.7. Creating tables.

```
create table departments (
  num int not null,
  name text not null,
  constraint pk_departments primary key (num)
);
```

```
create table departments (
  num int not null,
  name varchar(50) not null,
  constraint pk_departments primary key (num)
);
```

5.7. Creating tables.

You can find a good article about this here.

```
create table employees (
  num int not null,
  name text not null,
  salary int,
  dnum int not null,
  constraint pk_employees primary key (num),
  constraint fk_departments FOREIGN KEY (dnum) REFERENCES departments (num)
);
CREATE INDEX idx_fk_departments ON employees (dnum);
```

PostgreSQL

```
create table employees (
  num int not null,
  name varchar(150) not null,
  salary int,
  dnum int not null,
  constraint pk_employees primary key (num),
  constraint fk_departments FOREIGN KEY (dnum) REFERENCES departments (num)
);
```

MariaDB

5.7. Creating tables.

Foreign keys and indexes in PostgreSQL

PostgreSQL **automatically creates indexes** on **primary keys** and **unique constraints, but not on** the referencing side of **foreign key relationships**.

```
create table employees (
  num int not null,
  name text not null,
  salary int,
  dnum int not null,
  constraint pk_employees primary key (num),
  constraint fk_departments FOREIGN KEY (dnum)
REFERENCES departments(num)
);
```

```
CREATE INDEX idx_fk_departments ON employees (dnum);
```

PostgreSQL

5.7. Creating tables.

Ways to define a foreign key in PostgreSQL

CASE 1:

```
create table employees (
  num int not null,
  name text not null,
  salary int,
  dnum int not null,
  constraint pk_employees primary key (num),
  constraint fk_departments FOREIGN KEY (dnum) REFERENCES departments (num)
);
```

PostgreSQL

5.7. Creating tables.

Ways to define a foreign key in PostgreSQL

CASE 2:

```
create table employees (
  num int not null,
  name text not null,
  salary int,
  dnum int not null,
  constraint pk_employees primary key (num),
  FOREIGN KEY (dnum) REFERENCES departments (num)
);
```

PostgreSQL

5.7. Creating tables.

Ways to define a foreign key in PostgreSQL

CASE 4:

```
create table employees (
  num int not null,
  name text not null,
  salary int,
  dnum int not null REFERENCES departments (num),
  constraint pk_employees primary key (num)
);
```

PostgreSQL

5.7. Creating tables.

Ways to define a foreign key in PostgreSQL

CASE 5:

```
create table employees (
  num int not null,
  name text not null,
  salary int,
  dnum int not null,
  constraint pk_employees primary key (num)
);
ALTER TABLE employees
   ADD CONSTRAINT fk_departments FOREIGN KEY (dnum) REFERENCES departments(num);
```

PostgreSQL

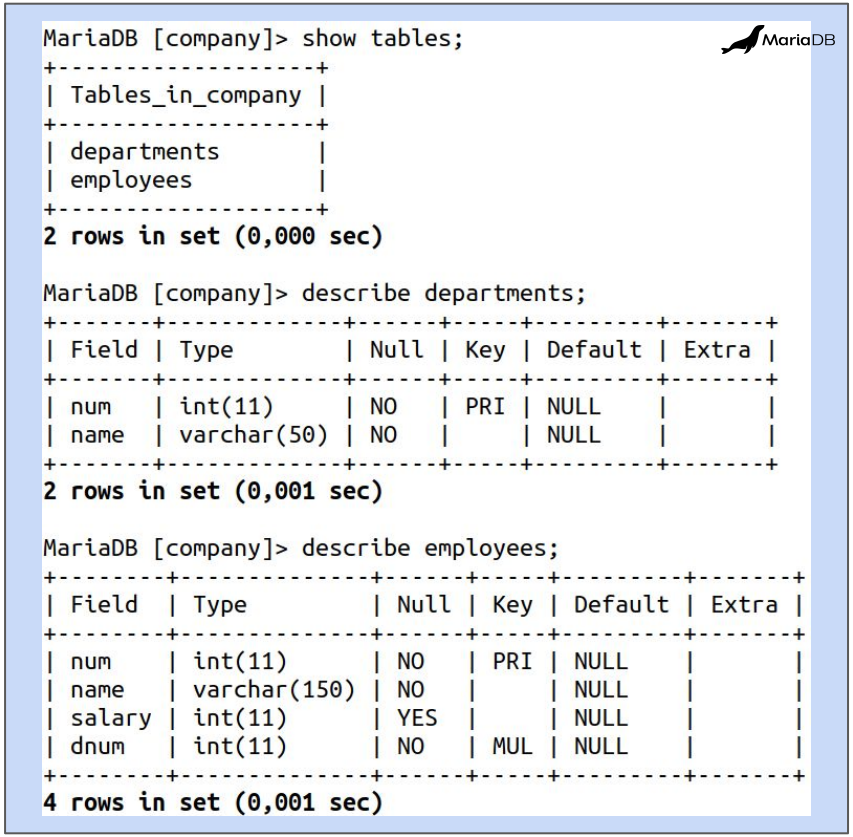
5.7. Creating tables.

Ways to define a foreign key in MariaDB

Just the same!!

But you do not need to create an index for the foreign keys.

5.7. Creating tables.



```
MariaDB [company]> show tables;
+-------------------+
| Tables_in_company |
+-------------------+
| departments       |
| employees         |
+-------------------+
2 rows in set (0,000 sec)

MariaDB [company]> describe departments;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| num   | int(11)     | NO   | PRI | NULL    |       |
| name  | varchar(50) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0,001 sec)

MariaDB [company]> describe employees;
+--------+-------------+------+-----+---------+-------+
| Field  | Type        | Null | Key | Default | Extra |
+--------+-------------+------+-----+---------+-------+
| num    | int(11)     | NO   | PRI | NULL    |       |
| name   | varchar(150) | NO   |     | NULL    |       |
| salary | int(11)     | YES  |     | NULL    |       |
| dnum   | int(11)     | NO   | MUL | NULL    |       |
+--------+-------------+------+-----+---------+-------+
4 rows in set (0,001 sec)
```

5.7. Creating tables.

```
company=# \dt
            List of relations
 Schema |    Name     | Type  |  Owner
--------+-------------+-------+----------
 public | departments | table | postgres
 public | employees   | table | postgres
(2 rows)

company=# \d departments
            Table "public.departments"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 num    | integer |           | not null |
 name   | text    |           | not null |
Indexes:
    "pk_departments" PRIMARY KEY, btree (num)
Referenced by:
    TABLE "employees" CONSTRAINT "fk_departments" FOREIGN KEY (dnum) REFERENCES departments(num)

company=# \d employees
                    Table "public.employees"
 Column |          Type          | Collation | Nullable | Default
--------+------------------------+-----------+----------+---------
 num    | integer                |           | not null |
 name   | character varying(150) |           | not null |
 salary | integer                |           |          |
 dnum   | integer                |           | not null |
Indexes:
    "pk_employees" PRIMARY KEY, btree (num)
Foreign-key constraints:
    "fk_departments" FOREIGN KEY (dnum) REFERENCES departments(num)
```

5.7. Creating tables.

Table names with blank spaces, SQL keyworks, strange characters, etc.

```
CREATE TABLE `final test'` (
    `field one` INT AUTO_INCREMENT NOT NULL,
    `order` CHAR(30) NOT NULL,
    `table` CHAR(30) NOT NULL,
    PRIMARY KEY (`field one`)
 );
```

```
MariaDB [company]> CREATE TABLE `final test'` (
    ->      `field one` INT AUTO_INCREMENT NOT NULL,
    ->      `order` CHAR(30) NOT NULL,
    ->      `table` CHAR(30) NOT NULL,
    ->      PRIMARY KEY (`field one`)
    ->  );
Query OK, 0 rows affected (0,021 sec)

MariaDB [company]> describe `final test'`;
+-----------+----------+------+-----+---------+----------------+
| Field     | Type     | Null | Key | Default | Extra          |
+-----------+----------+------+-----+---------+----------------+
| field one | int(11)  | NO   | PRI | NULL    | auto_increment |
| order     | char(30) | NO   |     | NULL    |                |
| table     | char(30) | NO   |     | NULL    |                |
+-----------+----------+------+-----+---------+----------------+
3 rows in set (0,002 sec)
```

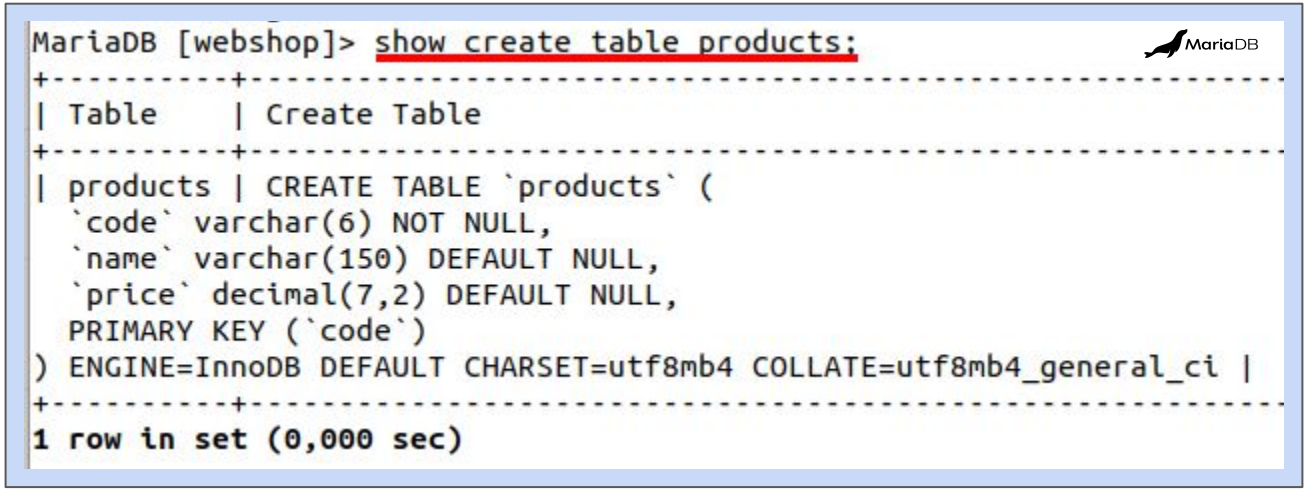
5.7. Creating tables.

Table names with blank spaces, SQL keyworks, strange characters, etc.

```
CREATE TABLE "final test'" (
    "field one" SERIAL NOT NULL,
    "order" CHAR(30) NOT NULL,
    "table" CHAR(30) NOT NULL,
    PRIMARY KEY ("field one")
 );
```

```
company=# CREATE TABLE "final test'" (
    "field one" SERIAL NOT NULL,
    "order" CHAR(30) NOT NULL,
    "table" CHAR(30) NOT NULL,
    PRIMARY KEY ("field one")
 );
CREATE TABLE
company=# \d "final test'"
                                    Table "public.final test'"
  Column   |     Type      | Collation | Nullable |                      Default
-----------+---------------+-----------+----------+-----------------------------------------------------
 field one | integer       |           | not null | nextval('"final test''_field one_seq"'::regclass)
 order     | character(30) |           | not null |
 table     | character(30) |           | not null |
Indexes:
    "final test'_pkey" PRIMARY KEY, btree ("field one")
```

5.7. Creating tables.



5.7. Creating tables.

```
webshop=# SELECT
  'CREATE TABLE ' || relname || E'\n(\n' ||
  array_to_string(
    array_agg(
      '    ' || column_name || ' ' ||  type || ' '|| not_null
    )
    , E',\n'
  ) || E'\n);\n'
from
(
  SELECT
    c.relname, a.attname AS column_name,
    pg_catalog.format_type(a.atttypid, a.atttypmod) as type,
    case
      when a.attnotnull
    then 'NOT NULL'
    else 'NULL'
    END as not_null
  FROM pg_class c,
   pg_attribute a,
   pg_type t
   WHERE c.relname = 'products'
   AND a.attnum > 0
   AND a.attrelid = c.oid
   AND a.atttypid = t.oid
 ORDER BY a.attnum
) as tabledefinition
group by relname;
                ?column?
----------------------------------------
 CREATE TABLE products                  +
 (                                      +
     code character varying(5) NOT NULL,  +
     name character varying(150) NOT NULL,+
     price money NOT NULL               +
 );                                     +

(1 row)
```

```
SELECT
  'CREATE TABLE ' || relname || E'\n(\n' ||
  array_to_string(
    array_agg(
      '    ' || column_name || ' ' ||  type || ' '|| not_null
    )
    , E',\n'
  ) || E'\n);\n'
from
(
  SELECT
    c.relname, a.attname AS column_name,
    pg_catalog.format_type(a.atttypid, a.atttypmod) as type,
    case
      when a.attnotnull
    then 'NOT NULL'
    else 'NULL'
    END as not_null
  FROM pg_class c,
   pg_attribute a,
   pg_type t
   WHERE c.relname = 'tablename'
   AND a.attnum > 0
   AND a.attrelid = c.oid
   AND a.atttypid = t.oid
 ORDER BY a.attnum
) as tabledefinition
group by relname;
```

Source:
https://stackoverflow.com/questions/2593803/how-to-generate-the-create-table-sql-statement-for-an-existing-table-in-postgr

5.7. Integrity constraints.

3.7. Relational integrity.

THE ENTITY INTEGRITY RULE

No component of the primary key of a base relation is allowed to accept nulls.

Remember that:

- **NULL** may mean "**property does not apply**". For example, the supplier may be a country, in which case the attribute CITY has a null value because such property does not apply.
- **NULL** may mean "**value is unknown**". For example, if the supplier is a person, then a null value for CITY attribute means we do not know the location of this supplier.
- **NULLs cannot be in primary keys**, **but can be in foreign keys**.
- EXAMPLE: DNI_partner can be NULL if you are not married.

5.7. Integrity constraints.

3.7. Relational integrity.

THE REFERENTIAL INTEGRITY RULE

The database can not contain inconsistent foreign key values.

In other words, the **valid values of a foreign key** are:

- Existing values in the primary key of reference.
- NULL values.

Another way of saying it:

- The database must not contain any unmatched foreign key values.

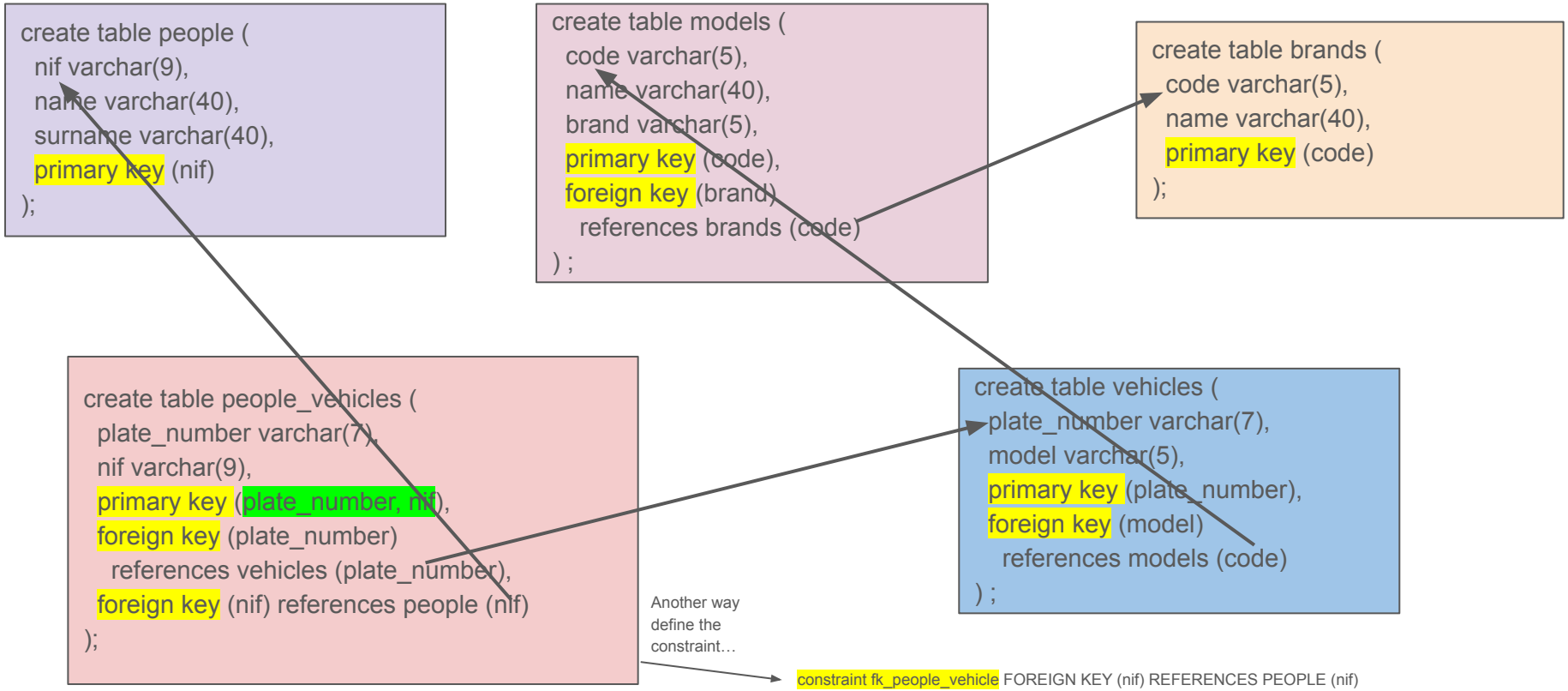
5.7. Integrity constraints.

primary key declaration on an attribute automatically ensures **not null**

- **not null**
- **primary key** (A_1, ..., A_n)
- **foreign key** (A_m, ..., A_n) **references** r
- **unique**
- **check (expression)**
- **default**

```
CREATE TABLE EMPLOYEES (
  num INTEGER,
  surname VARCHAR(50) NOT NULL,
  name VARCHAR(50) NOT NULL,
  manager INTEGER UNIQUE,
  start_date DATE,
  salary INTEGER,
  commission INTEGER,
  dept_num INTEGER DEFAULT 10,
  CHECK salary > 1000;
  PRIMARY KEY (num),
  FOREIGN KEY (dept_num)
    REFERENCES DEPARTMENTS (num),
  FOREIGN KEY (manager)
    REFERENCES EMPLOYEES (num)
);
```

5.7. Integrity constraints.



5.7. Integrity constraints.

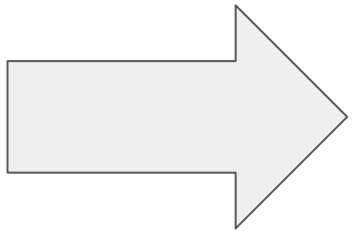
```
create table ITEMS (
        ID int primary key,
        category text NOT NULL,
        color text NOT NULL,
        primary key (ID)) ENGINE=InnoDB;
```

```
create table SALES (
        num int,
        itemID int,
        custID int,
        salesDate date,
        price float,
        primary key (num),
        foreign key (itemID) references ITEMS(ID),
        foreign key (custID) references CUSTOMERS(ID)
        ) ENGINE=InnoDB;
```

```
create table CUSTOMERS (
        ID int primary key,
        name VARCHAR(30),
        gender char NOT NULL,
        age int,
        primary key (num))
ENGINE=InnoDB;
```

5.7. Integrity constraints.

Cascading actions in referential integrity



If you delete a certain customer, what does it happen with his/her sales??

5.7. Integrity constraints.

```
create table SALES (
        [...]
        primary key (num),
        foreign key (itemID) references ITEMS(ID),
        foreign key (custID) references CUSTOMERS(ID)
                on delete cascade
                on update cascade
        );
```

If you delete a certain article, what should we do?

on delete?
on update?

- alternative actions to cascade:
  - **set null**
  - **set default**
  - **restrict**

Examples:
https://www.sqlteam.com/articles/using-set-null-and-set-default-with-foreign-key-constraints

5.7. Integrity constraints.

```
create table departments (
 id serial primary key,
 name varchar(20)
);
create table employees (
 id serial primary key,
 name varchar(150),
 deptid int references departments (id)
   on delete cascade
   on update cascade
);
```

```
insert into departments (name) values
 ('Accounting'),
 ('Sales'),
 ('Production');

insert into employees (name, deptid) values
 ('Sergi G.', 1),
 ('Paco Pol', 2),
 ('Andrew Bauman', 1);

update departments set id=10 where id=1;

delete from departments where id=10;
```

5.7. Integrity constraints.

```
employees=# select * from departments;
 id |    name
----+------------
  1 | Accounting
  2 | Sales
  3 | Production
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |      1
  2 | Paco Pol       |      2
  3 | Andrew Bauman  |      1
(3 rows)
```

update departments set id=10
where id=1;


```
employees=# select * from departments;
 id |    name
----+------------
  2 | Sales
  3 | Production
 10 | Accounting
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  2 | Paco Pol       |      2
  1 | Sergi G.       |     10
  3 | Andrew Bauman  |     10
(3 rows)
```

5.7. Integrity constraints.

```
employees=# select * from departments;
 id |    name
----+------------
  2 | Sales
  3 | Production
 10 | Accounting
(3 rows)

employees=# select * from employees;
 id |     name      | deptid
----+---------------+--------
  2 | Paco Pol      |      2
  1 | Sergi G.      |     10
  3 | Andrew Bauman |     10
(3 rows)
```


```
employees=# select * from departments;
 id |    name
----+------------
  2 | Sales
  3 | Production
(2 rows)

employees=# select * from employees;
 id |   name   | deptid
----+----------+--------
  2 | Paco Pol |      2
(1 row)
```

5.7. Integrity constraints.

ON UPDATE/DELETE SET NULL

```
create table departments (
 id serial primary key,
 name varchar(20)
);
create table employees (
 id serial primary key,
 name varchar(150),
 deptid int references departments (id)
   on delete set null
   on update set null
);
```

```
insert into departments (name) values
 ('Accounting'),
 ('Sales'),
 ('Production');

insert into employees (name, deptid) values
 ('Sergi G.', 1),
 ('Paco Pol', 2),
 ('Andrew Bauman', 1);

update departments set id=10 where id=1;

delete from department where id=2;
```

5.7. Integrity constraints.

ON UPDATE SET NULL

```
employees=# select * from departments;
 id |    name
----+------------
  1 | Accounting
  2 | Sales
  3 | Production
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |      1
  2 | Paco Pol       |      2
  3 | Andrew Bauman  |      1
(3 rows)
```

update departments set id=10
where id=1;

PostgreSQL

on update set null

```
employees=# select * from departments;
 id |    name
----+------------
  2 | Sales
  3 | Production
 10 | Accounting
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  2 | Paco Pol       |      2
  1 | Sergi G.       |
  3 | Andrew Bauman  |
(3 rows)
```

5.7. Integrity constraints.

ON DELETE SET NULL

```
employees=# select * from departments;
 id |    name
----+------------
  2 | Sales
  3 | Production
 10 | Accounting
(3 rows)
```

```
employees=# select * from employees;
 id |     name      | deptid
----+---------------+--------
  2 | Paco Pol      |      2
  1 | Sergi G.      |
  3 | Andrew Bauman |
(3 rows)
```

delete from departments where id=2;

on delete set null

```
employees=# select * from departments;
 id |    name
----+------------
  3 | Production
 10 | Accounting
(2 rows)
```

```
employees=# select * from employees;
 id |     name      | deptid
----+---------------+--------
  1 | Sergi G.      |
  3 | Andrew Bauman |
  2 | Paco Pol      |
(3 rows)
```

5.7. Integrity constraints.

ON UPDATE/DELETE CASCADE

```
create table departments (
 id serial primary key,
 name varchar(20)
);
create table employees (
 id serial primary key,
 name varchar(150),
 deptid int references departments (id)
   on delete cascade
   on update cascade
);
```

```
insert into departments (name) values
 ('Accounting'),
 ('Sales'),
 ('Production');

insert into employees (name, deptid) values
 ('Sergi G.', 1),
 ('Paco Pol', 2),
 ('Andrew Bauman', 1);

update departments set id=10 where id=1;

delete from department where id=2;
```

5.7. Integrity constraints.

ON UPDATE CASCADE

```
employees=# select * from departments;
 id |    name
----+------------
  1 | Accounting
  2 | Sales
  3 | Production
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |      1
  2 | Paco Pol       |      2
  3 | Andrew Bauman  |      1
(3 rows)
```

update departments set id=10
where id=1;

PostgreSQL

on update set null

```
employees=# select * from departments;
 id |    name
----+------------
  2 | Sales
  3 | Production
 10 | Accounting
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  2 | Paco Pol       |      2
  1 | Sergi G.       |     10
  3 | Andrew Bauman  |     10
(3 rows)
```

5.7. Integrity constraints.

ON DELETE CASCADE

```
employees=# select * from departments;
 id |    name
----+------------
  2 | Sales
  3 | Production
 10 | Accounting
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  2 | Paco Pol       |      2
  1 | Sergi G.       |     10
  3 | Andrew Bauman  |     10
(3 rows)
```

delete from departments where id=2;

PostgreSQL

on delete cascade

```
employees=# select * from departments;
 id |    name
----+------------
 3 | Production
 10 | Accounting
(2 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |     10
  3 | Andrew Bauman  |     10
(2 rows)
```

X

5.7. Integrity constraints.

ON UPDATE/DELETE SET DEFAULT

```
create table departments (
 id serial primary key,
 name varchar(20)
);
create table employees (
 id serial primary key,
 name varchar(150),
 deptid int default 1 references
departments (id)
   on delete set default
   on update set default
);
```

```
insert into departments (name) values
 ('Accounting'),
 ('Sales'),
 ('Production');

insert into employees (name, deptid) values
 ('Sergi G.', 1),
 ('Paco Pol', 2),
 ('Andrew Bauman', 3);

update departments set id=10 where id=2;

delete from department where id=3;
```

5.7. Integrity constraints.

ON UPDATE SET DEFAULT

```
employees=# select * from departments;
 id |    name
----+------------
  1 | Accounting
  2 | Sales
  3 | Production
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |      1
  2 | Paco Pol       |      2
  3 | Andrew Bauman  |      3
(3 rows)
```

update departments set id=10
where id=2;

PostgreSQL

on update set default

```
employees=# select * from departments;
 id |    name
----+------------
  1 | Accounting
  3 | Production
 10 | Sales
(3 rows)

employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |      1
  3 | Andrew Bauman  |      3
  2 | Paco Pol       |      1
(3 rows)
```

5.7. Integrity constraints.

ON DELETE SET DEFAULT

```
employees=# select * from departments;
 id |    name
----+------------
  1 | Accounting
  3 | Production
 10 | Sales
(3 rows)
```

```
employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |      1
  3 | Andrew Bauman  |      3
  2 | Paco Pol       |      1
(3 rows)
```

delete from departments where id=3;

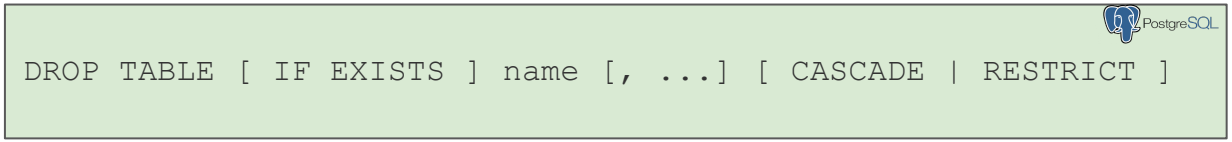
PostgreSQL

```
employees=# select * from departments;
 id |    name
----+------------
  1 | Accounting
 10 | Sales
(2 rows)
```

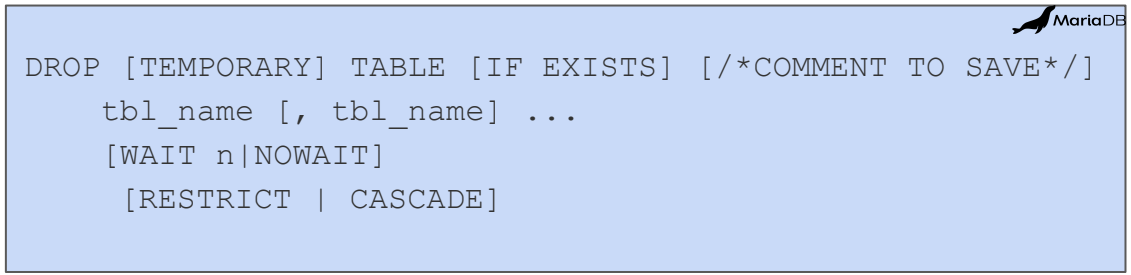
```
employees=# select * from employees;
 id |     name       | deptid
----+----------------+--------
  1 | Sergi G.       |      1
  2 | Paco Pol       |      1
  3 | Andrew Bauman  |      1
(3 rows)
```

on delete set default

5.8. Dropping tables.



https://www.postgresql.org/docs/current/sql-droptable.html



https://mariadb.com/kb/en/drop-table/

If a foreign key references this table, the table cannot be dropped. In this case, it is necessary to drop the foreign key first.

`RESTRICT` and `CASCADE` are allowed to make porting from other database systems easier. In MariaDB, they do nothing.

5.8. Dropping tables.

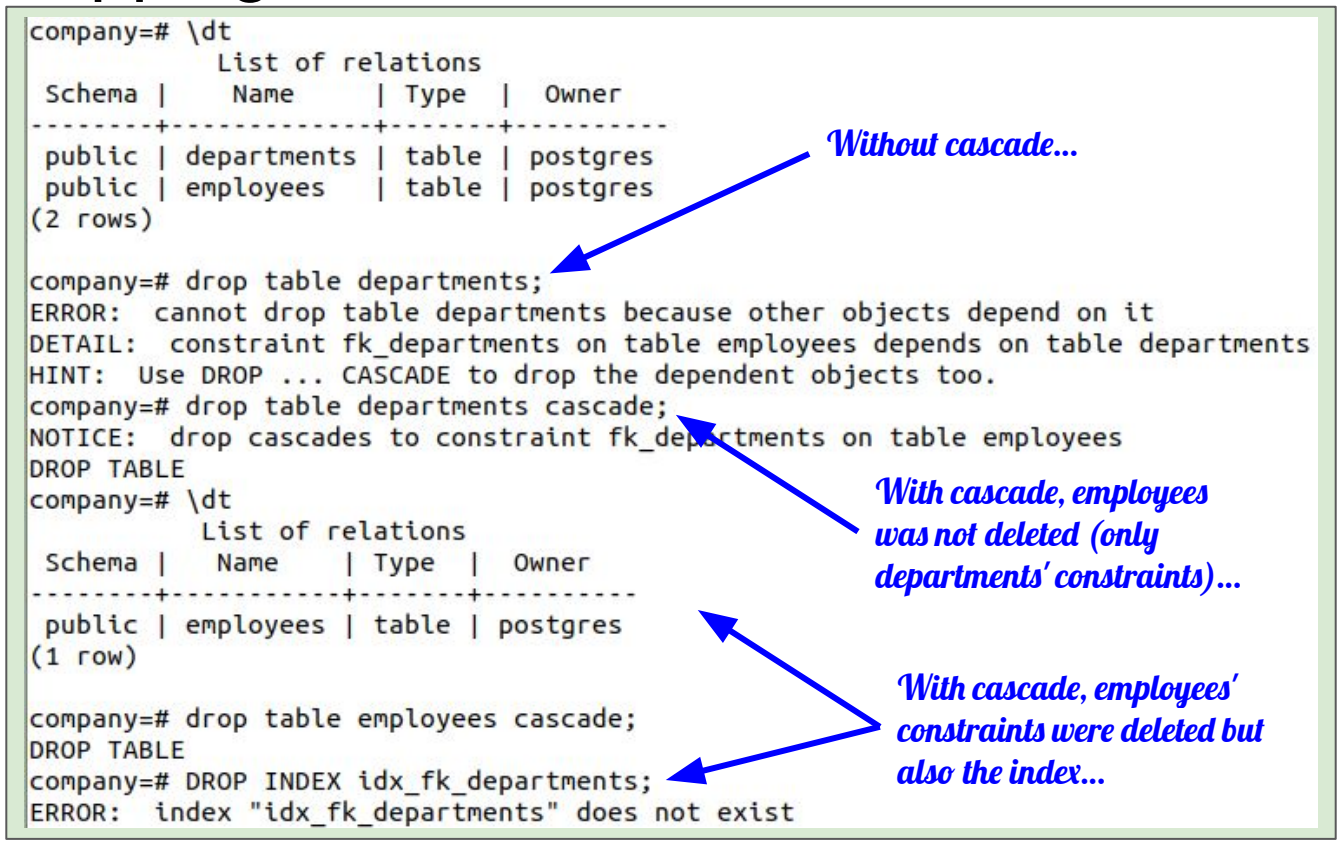
```
MariaDB [company]> show tables;
+-------------------+
| Tables_in_company |
+-------------------+
| departments       |
| employees         |
+-------------------+
2 rows in set (0,000 sec)

MariaDB [company]> drop table departments cascade;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails
MariaDB [company]>
MariaDB [company]> drop table employees;
Query OK, 0 rows affected (0,025 sec)

MariaDB [company]> drop table departments;
Query OK, 0 rows affected (0,022 sec)

MariaDB [company]> show tables;
Empty set (0,000 sec)
```

5.8. Dropping tables.



5.9. Creating indexes.

Indexes are a **data structure** used in the background of a database to **speed up querying**. We will see this concept in more detail later…

MariaDB

```
CREATE [OR REPLACE] [UNIQUE|FULLTEXT|SPATIAL] INDEX
  [IF NOT EXISTS] index_name
    [index_type]
    ON tbl_name (index_col_name,...)
    [WAIT n | NOWAIT]
    [index_option]
    [algorithm_option | lock_option] ...

index_col_name:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH | RTREE}

index_option:
    [ KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
  | CLUSTERING={YES| NO} ]
  [ IGNORED | NOT IGNORED ]

algorithm_option:
    ALGORITHM [=] {DEFAULT|INPLACE|COPY|NOCOPY|INSTANT}

lock_option:
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

PostgreSQL

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ [ IF NOT EXISTS ] name ] ON [ ONLY ] table_name [ USING method ]
    ( { column_name | ( expression ) } [ COLLATE collation ] [ opclass [ ( opclass_parameter = value [, ... ] ) ] ] [ ASC | DESC ] [ NULLS { FIRST | LAST } ] [, ...] )
    [ INCLUDE ( column_name [, ...] ) ]
    [ NULLS [ NOT ] DISTINCT ]
    [ WITH ( storage_parameter [= value] [, ... ] ) ]
    [ TABLESPACE tablespace_name ]
    [ WHERE predicate ]
```

5.9. Creating indexes.

PostgreSQL **automatically creates indexes** on **primary keys** and **unique constraints**, **but not on** the referencing side of **foreign key relationships**.

```
CREATE TABLE a (
   a_id int PRIMARY KEY
);

CREATE TABLE b (
   b_id int PRIMARY KEY,
   FOREIGN KEY a_id int REFERENCES a (a_id)
                                  ON UPDATE CASCADE
                                   ON DELETE CASCADE
);
```

create the index

```
CREATE INDEX idx_b ON b (a_id);
```

You can find a good article about this here.

5.10. Dropping indexes.

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

https://www.postgresql.org/docs/15/sql-dropindex.html

```
DROP INDEX [IF EXISTS] index_name ON tbl_name [WAIT n |NOWAIT]
```

https://mariadb.com/kb/en/drop-index/

5.11. Altering databases, tables, and indexes.

- You can alter all the objects of a database with the keyword **ALTER**.
- For example:
  - You may alter a table to add or drop a column. Or maybe to change the data type of that column.
- We are not going to see it in the presentation because **I want you to research and acquire the skills by researching autonomously**.
- You must use the official documentation:
  - https://mariadb.com/kb/en/
  - https://www.postgresql.org/docs/current/

5.11. Altering databases, tables, and indexes.

Some examples…

- **add a column**:

```
ALTER TABLE table_name
  ADD new_column_name column_definition
    [ FIRST | AFTER column_name ];
```

```
ALTER TABLE websites
  ADD host_name varchar(40)
    AFTER server_name;
```

- **modify a column**:

```
ALTER TABLE table_name
  MODIFY column_name column_definition
    [ FIRST | AFTER column_name ];
```

```
ALTER TABLE websites
  MODIFY host_name
varchar(50);
```

5.11. Altering databases, tables, and indexes.

Some examples…

- **rename a column**:

  ALTER TABLE table_name
  CHANGE COLUMN old_name new_name
  column_definition
  [FIRST | AFTER column_name]

MariaDB

```
ALTER TABLE websites
  CHANGE COLUMN host_name hname
    varchar(25);
```

- **rename a table**:

  ALTER TABLE table_name
  RENAME TO new_table_name;

MariaDB

```
ALTER TABLE websites
  RENAME TO sites;
```

5.11. Altering databases, tables, and indexes.

Some examples…

- **add pk**:

```
ALTER TABLE table_name
  ADD PRIMARY KEY old_name new_name
    column_definition;
```

```
ALTER TABLE goods
  ADD PRIMARY KEY (id);
```

- **add fk**:

```
ALTER TABLE table_name
  ADD FOREIGN KEY (column_name)
REFERENCES table_name(coulmn_name)
```

```
ALTER TABLE users
  ADD CONSTRAINT fk_grade_id
FOREIGN KEY (grade_id)
REFERENCES grades(id);
```

5.11. Altering databases, tables, and indexes.

Some examples…

ALTER TABLE table_name ADD COLUMN new_column_name TYPE;	Add a new column to a table.
ALTER TABLE table_name DROP COLUMN column_name	Drop a column in a table.
ALTER TABLE table_name RENAME column_name TO new_column_name;	Rename a column.
ALTER TABLE table_name ALTER COLUMN [SET DEFAULT value | DROP DEFAULT].	Set or remove a default value for a column.

5.11. Altering databases, tables, and indexes.

Some examples…

ALTER TABLE table_name ADD PRIMARY KEY (column,...);	Add a primary key to a table.
ALTER TABLE table_name DROP CONSTRAINT primary_key_constraint_name;	Remove the primary key from a table.
ALTER TABLE table_name RENAME TO new_table_name;	Rename a table.

5.12. A bit of DML.

In a nutshell, DML (Data Manipulation Language) is:

Statement	Description
INSERT	Adds a new row to a table.
UPDATE	Changes existing data in a table.
DELETE	Removes rows from a table.
SELECT	Selects rows from tables.

5.12. A bit of DML.

- **Without field names (respect column order):**
  - INSERT INTO departments VALUES (10, 'Accounting and finance');
- **If you want to change column order:**
  - INSERT INTO departments (name, num)
    VALUES ('Marketing and sales',20);
- **Inserting multiple rows**:
  - INSERT INTO departments (num, name) VALUES
    (30, 'Human resources'),
    (40, 'Customer service'),
    (50, 'Research and development');

5.12. A bit of DML.

Inserts and serial/auto_increment:

```
CREATE TABLE departments (
  num serial NOT NULL primary key,
  name varchar(30) NOT NULL
);
```

Not using the serial/auto_increment:

```
INSERT INTO departments (num, name)
VALUES (1, 'ACCOUNTING'),
  (2, 'RESEARCH'),
  (3, 'SALES'),
  (4, 'PRODUCTION');
```

```
CREATE TABLE departments (
  num smallint AUTO_INCREMENT NOT NULL primary key,
  name varchar(30) NOT NULL
);
```

Using it <u>with</u> the keyword 'default':

```
INSERT INTO departments (num, name)
VALUES (default, 'ACCOUNTING'),
  (default, 'RESEARCH'),
  (default, 'SALES'),
  (default, 'PRODUCTION');
```

Using it <u>without</u> the keyword 'default':

```
INSERT INTO departments (name)
VALUES ('ACCOUNTING'),
  ('RESEARCH'),
  ('SALES'),
  ('PRODUCTION');
```

5.12. A bit of DML.

- **Updating a field of a row:**
  - update employees
    set name = 'Sergi González'
    where num = 1000;
- **Updating some fields of a row:**
  - update employees
    set name = 'Sergi González', dnum = 20
    where num = 1000;
- **Updating doing operations**:
  - update employees
    set salary = salary*2
    where num = 1000;

Update without where would change all the columns in the table.

5.12. A bit of DML.

- **Deleting a particular row (use PK in where):**
  - delete from employees
    where num = 1000;
- **Deleting (possibly) many rows:**
  - delete from employees
    where dnum = 10;
- **Deleting (possibly) many rows:**
  - delete from employees
    where dnum = 10 and salary > 3000;

Delete without where would delete all the columns in the table.

5.12. A bit of DML.

DML DDL

Delete VS Truncate:

This is only for PostSQL.

TRUNCATE table_a; **or DELETE** FROM table_a; is the same from a pint of view: **table_a will be emptied**.

With DELETEs, dead rows remain in database pages and their dead pointers are still present in indices (you can solve it running: VACUUM (FULL, ANALYZE) table_a;).

TRUNCATE, on the other hand, keeps the table “clean” (the resulting table looks almost identical internally to a newly created table). TRUNCATE is all or nothing: it doesn’t have WHERE!

But, Delete is DML, Truncate is DDL. As you know, it has implications: See this link.

So, maybe it’s a good idea if you change Delete for Truncate in Postgresql.

More information: here

Practical examples: here

PostgreSQL

But…

There are more concepts that we haven't seen in this unit. For example, the concept of schema in PostgreSQL. Why?

4. Módulo profesional: Gestión de bases de datos

Código: 0372

Contenidos:

a) Sistemas de almacenamiento de la información:

Ficheros (planos, indexados y acceso directo, entre otros).
Bases de datos. Conceptos, usos y tipos según el modelo de datos, la ubicación de la información.
Otros sistemas de almacenamiento (XML y servicios de directorios, entre otros).
Sistemas de información. Sistemas de información empresarial.
Sistemas gestores de base de datos: funciones, componentes y tipos.

b) Diseño lógico de bases de datos:

Modelo de datos.
Modelo lógico de la base de datos. Metodología.
La representación del problema: los diagramas E/R: Entidades y relaciones. Cardinalidad. Debilidad.
El modelo E/R ampliado.
El modelo relacional: Terminología del modelo relacional. Características de una relación. Claves primarias y claves enas. Álgebra relacional. Cálculo relacional.
Paso del diagrama E/R al modelo relacional.
Normalización: Dependencias funcionales. Formas normales. Justificación de la desnormalización.
El modelo orientado a objetos: Conceptos básicos del modelo orientado a objeto. Diagramas de clases y de objetos.

c) Diseño físico de bases de datos:

Herramientas gráficas proporcionadas por el sistema gestor para la implementación de la base de datos.
El lenguaje de definición de datos.
Creación, modificación y eliminación de bases de datos.
Creación, modificación y eliminación de tablas. Tipos de datos.
Implementación de restricciones.
Verificación de las restricciones.
Documentación del diseño.

d) Realización de consultas:

Herramientas gráficas proporcionadas por el sistema gestor para la realización de consultas.
Herramientas externas al gestor.
Sentencias para la consulta.
La sentencia SELECT.
Selección y ordenación de registros. Tratamiento de valores nulos.
Consultas de resumen. Agrupamiento de registros.
Unión de consultas.
Composiciones internas y externas.
Subconsultas.
Consultas complejas.

e) Edición de los datos:

Herramientas gráficas proporcionadas por el sistema gestor para la edición de la información.
Herramientas externas al gestor.
Las sentencias INSERT, DELETE y UPDATE.
Modelos de transacciones y de consultas.
Subconsultas y combinaciones en órdenes de edición.
Transacciones; sentencias de procesamiento de transacciones.
Formas de acceso a datos.
Acceso simultáneo a los datos: políticas de bloqueo.

f) Construcción de guiones:

Introducción. Lenguaje de programación.
Tipos de datos, identificadores, variables.
Operadores. Estructuras de control.
Estructuras funcionales: Módulos, procedimientos, funciones.
Funciones de librerías básicas disponibles.

g) Gestión de la seguridad de los datos:

Recuperación de fallos.
Principales fallos en una base de datos.
Herramientas del SGBD para la recuperación ante fallos.
Copias de seguridad. Tipos.
Planificación de copias de seguridad.
Herramientas gráficas y utilidades proporcionadas por el sistema gestor para la realización y recuperación de copias seguridad.
Sentencias para la realización y recuperación de copias de seguridad.
Herramientas gráficas y utilidades para importación y exportación de datos.
Herramientas de verificación de integridad de la base de datos.
Migración de datos entre sistemas gestores.
Documentación de las medidas y políticas de seguridad.

9. Módulo profesional: Administración de sistemas gestores de bases de datos

Código: 0377

Contenidos:

a) Instalación y configuración de un sistema gestor de base de datos:

Funciones del Sistema gestor de base de datos. Componentes. Tipos.
Arquitectura del Sistema gestor de base de datos, arquitectura ANSI/SPARC.
Sistemas gestores de base de datos comerciales y libres.
Instalación y configuración de un SGBD monocapa; parámetros relevantes.
Elementos de un SGBD de dos capas.
Instalación de un SGBD de dos capas.
Interfaces estándar (ODBC, JDBC); conectores.
Configuración de los parámetros relevantes.
Estructura del diccionario de datos.
Ficheros LOG.
SGBD de tres capas.
Otros sistemas de almacenamiento (XML, servicios de directorios,..).
Documentación.

b) Acceso a la información:

Creación, modificación y eliminación de vistas.
Administración de usuarios.
Creación y eliminación de usuarios.
Asignación y desasignación de derechos a usuarios; puntos de acceso al sistema.
Verificación de los derechos de usuario.
Seguridad en el acceso al sistema.
Definición de roles; asignación y desasignación de roles a usuarios.
Normativa legal vigente sobre protección de datos.

c) Automatización de tareas. Construcción de guiones de administración:

Herramientas para creación de guiones; procedimientos de ejecución.
Planificación de tareas de administración mediante guiones:

Técnicas de planificación de tareas.
Herramientas de planificación del SGBD.

Eventos.
Disparadores.
Excepciones.
Documentación.

d) Optimización del rendimiento. Monitorización y optimización:

Herramientas de monitorización disponibles en el sistema gestor.
Trazas, «log» y alertas.
Elementos y parámetros susceptibles de ser monitorizados.
Optimización: Almacenamiento en memoria, espacio en disco.
Transferencia y comunicaciones.
Ejecución de consultas.
Herramientas y sentencias para la gestión de índices.
Herramientas para la creación de alertas de rendimiento.

e) Aplicación de criterios de disponibilidad a bases de datos distribuidas y replicadas:

Bases de datos distribuidas; reglas de Date.
Tipos de SGBD distribuidos.
Tipo de SGBD: homogéneos y heterogéneos.
Distribución de los datos: centralizados y no centralizados.
Autonomía de los nodos: compuestos, federados y multibase.
Componentes de un SGBD distribuido.
Técnicas de fragmentación.
Técnicas de asignación.
Consulta distribuida.
Transacciones distribuidas.
Optimización de consultas sobre bases de datos distribuidas.
Replicación.
Configuración del nodo maestro y los nodos esclavos.

But…

7.26. Schemas (I).

- A database can contain one or more named **schemas**, which in turn contain tables.
- Schemas also contain other kinds of named objects, including tables, data types, functions, and operators.
- The same object name can be used in different schemas without conflict; for example, both “schema1” and “myschema” can contain tables named “mytable”.
- Unlike databases, schemas are not rigidly separated: a user can access objects in any of the schemas in the database they are connected to, if they have privileges to do so.
- More info: https://www.postgresql.org/docs/current/ddl-schemas.html

7.26. Schemas (II).

- There are several reasons why one might want to use schemas:
  - Allow many users to use one database without interfering with each other.
  - Organize database objects into logical groups to make them more manageable.
  - Third-party applications can be put into separate schemas so they do not collide with the names of other objects.
  - Schemas are analogous to directories at the operating system level, except that schemas cannot be nested.
- To create or access objects in a schema, write a qualified name consisting of the schema name and table name separated by a dot:
  - `schema.table`
- Or the even more general syntax:
  - `database.schema.table`

7.26. Schemas (III).

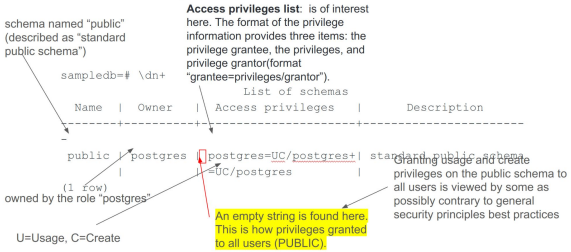
```
customer01@192:bookings> show search_path;
+-----------------+
| search_path     |
|-----------------|
| "$user", public |
+-----------------+
SHOW
Time: 0.083s
```

- To create an schema:
  - CREATE SCHEMA schemaname [Authorization username];
    - Authorization username: If omitted, the owner of the schema is the user executing the statement.
  - The variable SEARCH_PATH, which can be set at the cluster level or at user level, indicates a list of schemas were objects are searched in case the name of the owner is not given.
    - SET search_path TO my_schema, public;
    - ALTER USER user_name SET search_path = my_schema, public;
  - You can see your search path with:
    - SHOW search_path;
  - To drop a schema including all contained objects, use:
    - DROP SCHEMA myschema CASCADE;
  - Move a table from a schema to another:
    - ALTER TABLE table_name SET SCHEMA new_schema_name;

7.26. Schemas (IV).

- The Public Schema:
  - In the previous sections we created tables without specifying any schema names. By default such tables (and other objects) are automatically put into a schema named “public”. Every new database contains such a schema. Thus, the following are equivalent:
    - CREATE TABLE products (...);
    - and:
    - CREATE TABLE public.products (...);

7.26. Schemas (V).



7.26. Schemas (VI).

- sampledb=# REVOKE USAGE ON SCHEMA public FROM PUBLIC;
- sampledb=# REVOKE CREATE ON SCHEMA public FROM PUBLIC;

```
sampledb=# \dn+
                          List of schemas
  Name  | Owner  | Access privileges |      Description
--------+--------+-------------------+-----------------------
 public | postgres | postgres=UC/postgres | standard public schema
(1 row)
```

- sampledb=# REVOKE ALL PRIVILEGES ON SCHEMA public FROM PUBLIC;

```
sampledb=# \dn+
                          List of schemas
  Name  | Owner  | Access privileges |      Description
--------+--------+-------------------+-----------------------
 public | postgres |                   | standard public schema
(1 row)
```

But…

7.26. Schemas (VII).

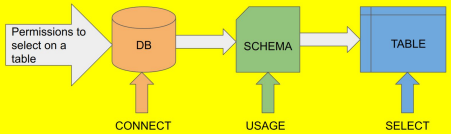
```
employeesdb_auth> SELECT current_schema();
+--------------------+
| current_schema     |
|--------------------|
| public             |
+--------------------+
SELECT 1
Time: 0.020s
employeesdb_auth> SET search_path TO test;
SET
Time: 0.001s
employeesdb_auth> SELECT current_schema();
+--------------------+
| current_schema     |
|--------------------|
| test               |
+--------------------+
SELECT 1
Time: 0.019s
```

7.26. Schemas (VIII).

- To drop a schema if it is empty (all objects in it have been dropped), use the command:
  - `DROP SCHEMA myschema;`
- To drop a schema including all contained objects, use the command −
  - `DROP SCHEMA myschema CASCADE;`

7.26. Schemas (IX).

- GRANTing on a database doesn't GRANT rights to the schema within.
- Similarly, GRANTing on a schema does not grant rights on the tables within.



7.26. Schemas (X).

- VERY IMPORTANT -> read this article (answer 91): https://dba.stackexchange.com/questions/117109/how-to-manage-default-privileges-for-users-on-a-database-vs-schema/117661#117661

7.26. Schemas (XI).

- Show permissions of a table (customers) inside a schema (app):

```
SELECT grantee, privilege_type
FROM
information_schema.role_table_grants
WHERE table_name='employees' and
table_schema='professors';
```

```
postgres@5192:p87> SELECT grantee, privilege_type
 FROM information_schema.role_table_grants
 WHERE table_name='customers' and table_schema='app';
+----------+----------------+
| grantee  | privilege_type |
|----------+----------------|
| user1    | INSERT         |
| user1    | SELECT         |
| user1    | UPDATE         |
| user1    | DELETE         |
| user1    | TRUNCATE       |
| user1    | REFERENCES     |
| user1    | TRIGGER        |
+----------+----------------+
SELECT 7
Time: 0.028s
```

- Show permissions granted to a role (adurango):

```
SELECT table_catalog, table_schema, table_name, privilege_type
FROM   information_schema.table_privileges
WHERE  grantee = 'adurango';
```

- Show roles granted to a role (adurango):

```
WITH RECURSIVE cte AS (
   SELECT oid FROM pg_roles WHERE rolname = 'adurango'

   UNION ALL
   SELECT m.roleid
   FROM   cte
   JOIN   pg_auth_members m ON m.member = cte.oid
)
SELECT oid, rolname FROM pg_roles WHERE oid IN (SELECT * FROM cte);
```

Sources.

- https://mariadb.com/kb/en/
- https://www.postgresql.org/docs/