

Construcción de software y toma de decisiones

TC2005B

Dr. Esteban Castillo Juarez

ITESM, Campus Santa Fe



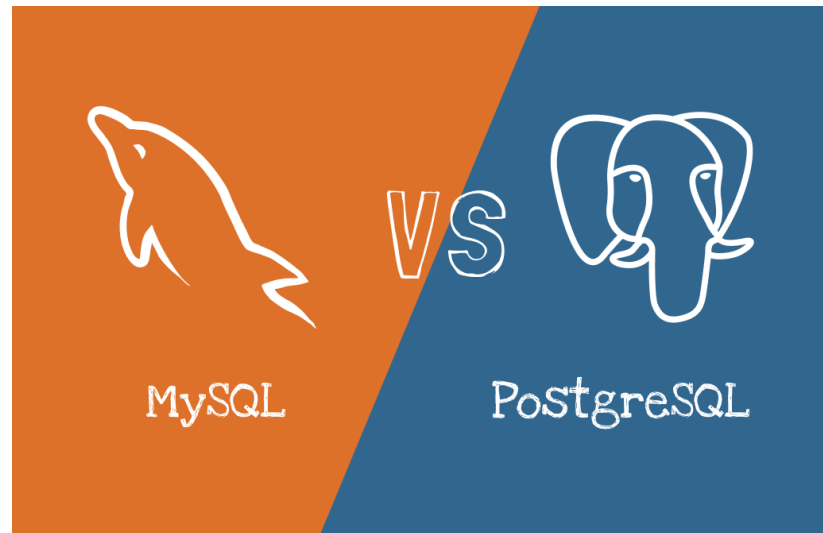
esteban.castillojz@tec.mx

Agenda

- MySQL vs PostgreSQL
- Creación de esquemas y tablas en PostgreSQL
- Inserción de datos en PostgreSQL
- Consulta de datos en PostgreSQL
- Creación de objetos en PostgreSQL
- Elementos extras

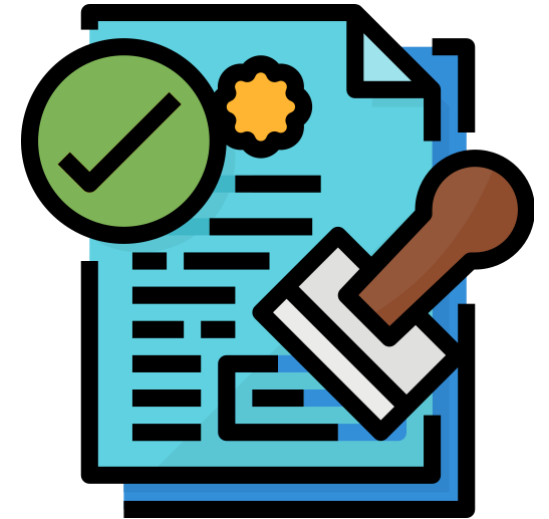
MySQL vs PostgreSQL

- MySQL y PostgreSQL son dos sistemas de gestión de bases de datos relacionales (RDBMS) muy populares y ampliamente utilizados en la industria.
- Aunque ambos cumplen con el estándar SQL y ofrecen una funcionalidad básica similar, tienen diferencias significativas en términos de características, rendimiento y filosofía de diseño.



MySQL vs PostgreSQL

- Una de las diferencias más destacadas entre MySQL y PostgreSQL es su **licencia y modelo de desarrollo**.
- **MySQL**, propiedad de Oracle Corporation, **está disponible bajo la licencia de código abierto** GPL (General Public License) para la comunidad, **pero también ofrece una versión comercial** con características adicionales.
- Por otro lado, **PostgreSQL**, desarrollado y mantenido por una comunidad de código abierto, **está bajo la licencia PostgreSQL, que es de código abierto** y permite un uso gratuito y la modificación del código fuente.



MySQL vs PostgreSQL

- **En cuanto a la arquitectura, PostgreSQL es conocido por su diseño extensible y orientado a objetos.** Ofrece soporte nativo para procedimientos almacenados, funciones definidas por el usuario, tipos de datos personalizados y otros elementos avanzados.
- **MySQL, por otro lado, se ha centrado históricamente en ser rápido y ligero,** lo que significa que su conjunto de características puede ser más limitado en comparación con PostgreSQL en algunos aspectos.



MySQL vs PostgreSQL

- **En términos de rendimiento**, las comparaciones pueden variar dependiendo de la carga de trabajo y la configuración específica de cada sistema.
- En general, **PostgreSQL tiende a ofrecer una mayor robustez y capacidad para manejar cargas de trabajo** complejas y de alta concurrencia.
- Lo anterior se debe en parte a su soporte avanzado para transacciones ACID y bloqueo de filas, que pueden ayudar a evitar problemas de integridad de datos y conflictos de concurrencia.



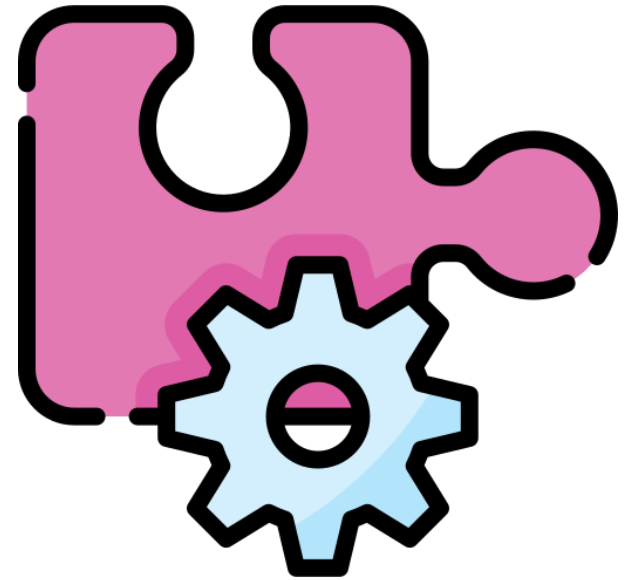
MySQL vs PostgreSQL

- **En cuanto a la escalabilidad y la gestión de clústeres**, ambas bases de datos ofrecen opciones, pero **PostgreSQL tiende a ser más flexible y potente** en este aspecto, especialmente para configuraciones de alta disponibilidad y distribución geográfica.
- Otro aspecto a considerar es **la compatibilidad con estándares y extensiones**. PostgreSQL tiende a ser más riguroso en adherirse a los estándares SQL y soportar extensiones que promueven la compatibilidad con otras bases de datos y facilitan la migración.



MySQL vs PostgreSQL

- **MySQL, aunque también compatible con SQL, ha tenido algunas desviaciones históricas y ha introducido sus propias extensiones**, lo que puede hacer que la migración entre plataformas sea más compleja en algunos casos.
- MySQL cuenta con una amplia gama de motores de almacenamiento alternativos, como InnoDB, MyISAM y Memory, que ofrecen diferentes características y compromisos de rendimiento.



MySQL vs PostgreSQL

- En general, tanto **MySQL** como **PostgreSQL** son opciones sólidas para la mayoría de los proyectos de bases de datos relacionales, pero es importante evaluar cuidadosamente las fortalezas y debilidades de cada uno en un ambiente de desarrollo comercial.
- La elección entre ellos dependerá de las necesidades específicas del proyecto, las preferencias del equipo de desarrollo y otros factores como el rendimiento, la escalabilidad y la compatibilidad con otras tecnologías y sistemas operativos.



MySQL vs PostgreSQL

PostgreSQL

Object-relational database management system that is the link between relational databases such as MySQL and object databases.

Multi-version concurrency control (MVCC) - it allows different readers and editors to use and manage the database at the same time, making the workflow more efficient.

MySQL

Supports a wide range of storage engines - this makes MySQL particularly flexible as many different table types can be chosen from

Cloud-ready database management system - Cloud platforms offer MySQL features that even take care of database installation and maintenance.

Multi-version concurrency control (MVCC) and ACID compliance are available using MySQL's InnoDB engine, which is currently MySQL's standard engine.

MySQL vs PostgreSQL

ACID compliance - data tampering and preserves the security of data at the transactional level.

Good for large databases - PostgreSQL can manage thousands of terabytes and happily process more than 100k queries per second.

Good for complex queries - for example, complicated writes with concurrent data usage that also needs to be validated, PostgreSQL is a very good choice.

NoSQL and a variety of other data types are supported - PostgreSQL is a particularly popular choice for using NoSQL features, but other data types such as XML, JSON or hstore are also supported by PostgreSQL.

Enables high scalability, combined with the high flexibility due to the many storage engines supported, the scalability is quite high.

Speed and reliability - MySQL is designed for speed and reliability, so certain SQL features have been kept out of the technology to keep MySQL lightweight.

Easy to use - unlike other database management systems, it is considered to be particularly easy to use and set up. As a result, many hosting providers offer MySQL as their default database system.

MySQL server optimisations - a variety of options for optimising the server are possible, as certain variables can be adjusted. NoSQL is also supported since MySQL 8.0

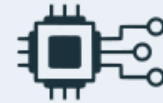
MySQL vs PostgreSQL

Factors	MySQL	Postgres
Architecture	It is a relational database management system (RDBMS)	It is an object-relational database management system (ORDBMS)
Speed	It is optimized for tables having more records but simpler queries.	It is optimized for complex queries
Performance	It is better suited for smaller applications with simple queries.	It is better suited for complex, high-traffic applications that require advanced features
Scalability	It does not support asynchronous replication, advanced locking, and row-level locking features	It has improved scalability due to asynchronous replication, advanced locking, and row-level locking features
Security	It includes authentication, encryption, and role-based access control when it comes to security.	It has more advanced authentication, access control, row-level security, and encryption features that make it a more secure choice for storing sensitive data.
Indexes	Most indexes are stored in B-trees and the memory table also supports hash indexes.	It supports B-tree, GiST, GIN, Hash, BRIN, and SP-GiST indexes
Data Types	All basic data types are supported	Along with basic data types custom data types can be created that are beneficial for complex data requirements.

MySQL vs PostgreSQL

When to use MySQL vs PostgreSQL vs SQLite?

Developing Embedded Applications



Launching Websites & Web Apps
that grow in the future



Creating Distributed Applications

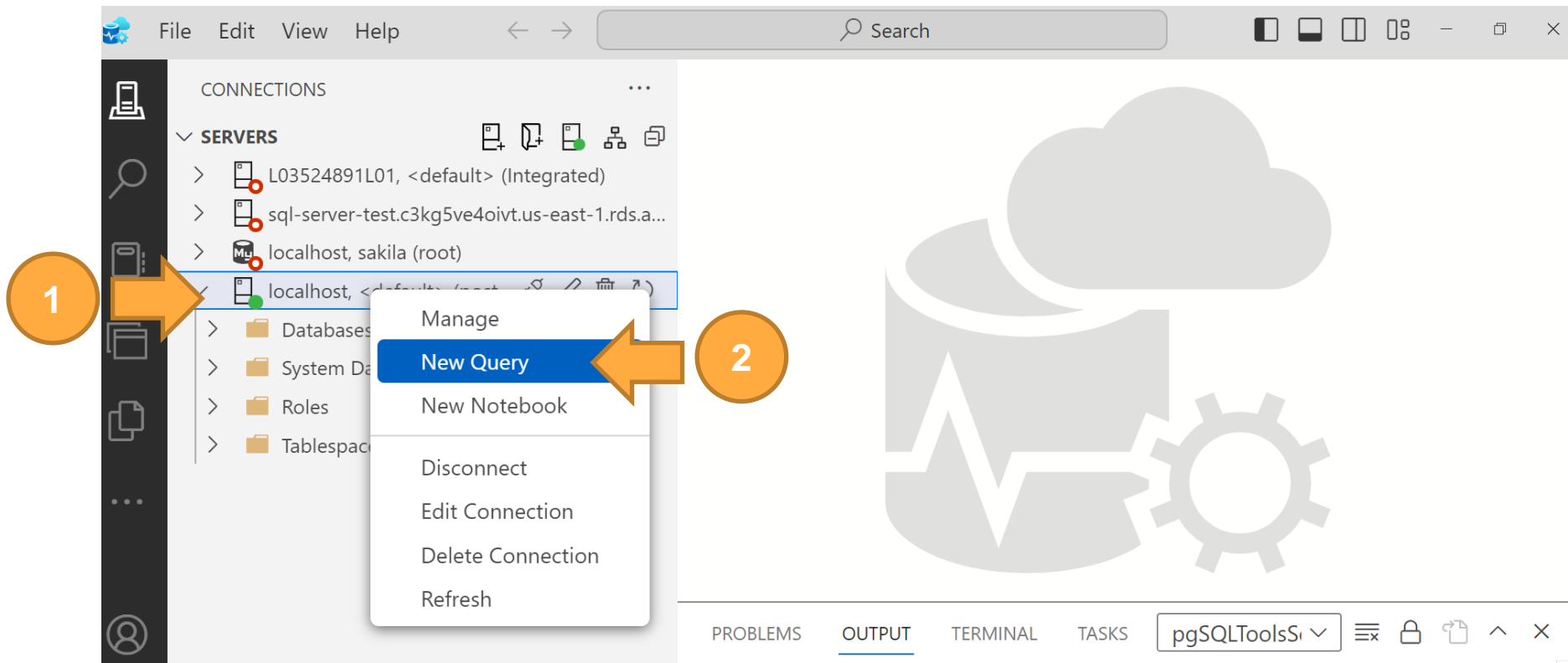


Developing Complex Applications



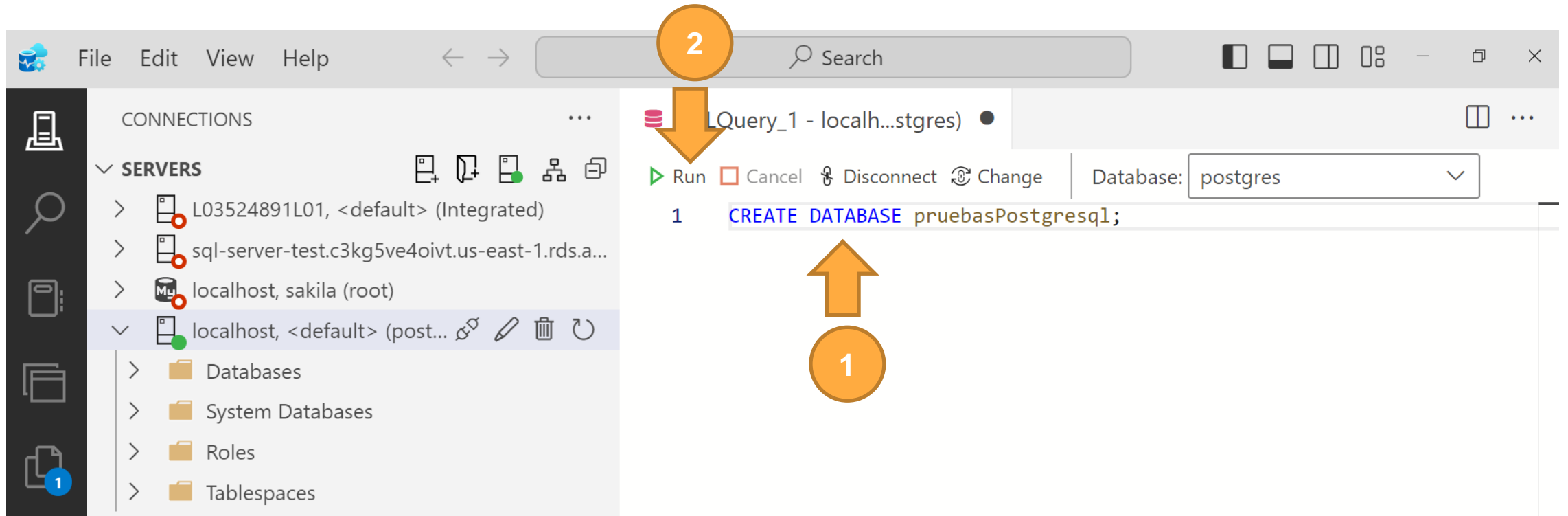
Creación de esquemas y tablas en PostgreSQL

1. Como primer paso, crearemos una base de datos en Azure Data Studio llamada pruebasPostgresql. Dentro de esta, se creara un esquema llamado prueba. Para ello, habrá la aplicación, conéctese a PostgreSQL y cree una nueva pagina de consulta.



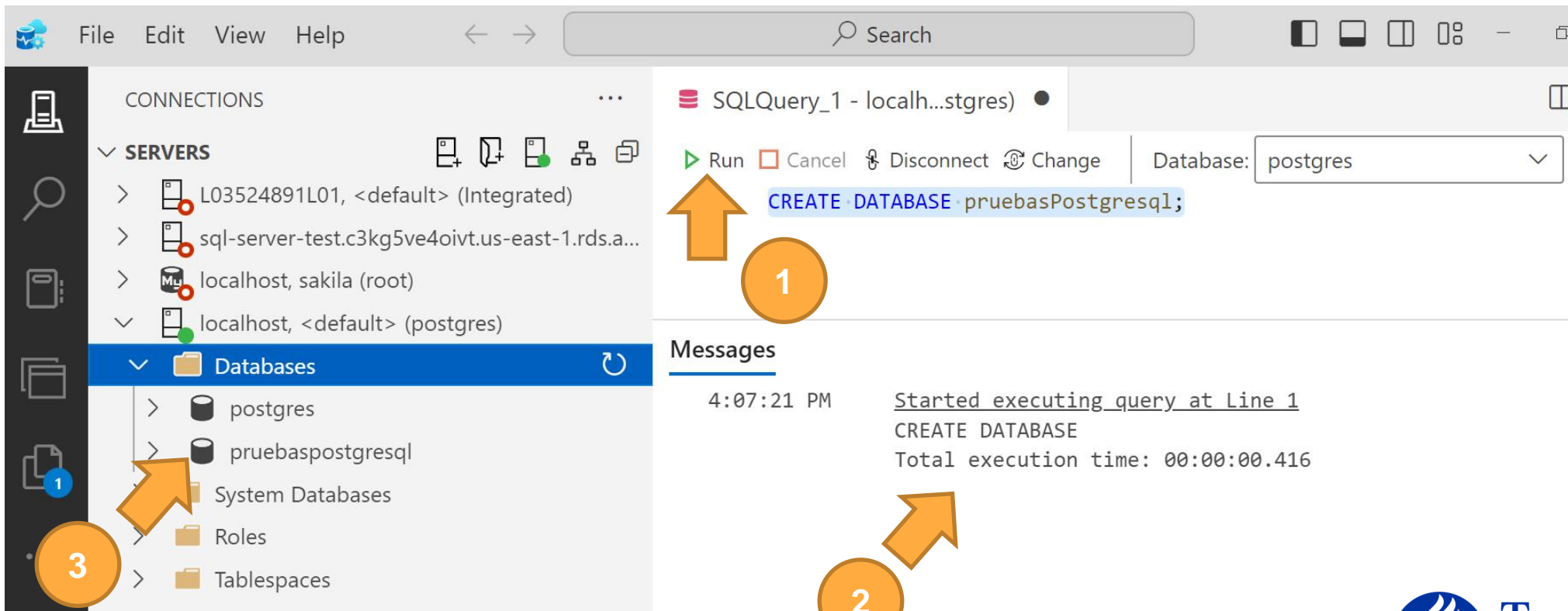
Creación de esquemas y tablas en PostgreSQL

2. Ejecute lo siguiente en la ventana de consulta: `CREATE DATABASE pruebasPostgresql;`



Creación de esquemas y tablas en PostgreSQL

3. Una vez creada la base de datos, podremos verla del lado izquierdo del navegador de elementos de Azure.



Creación de esquemas y tablas en PostgreSQL

4. El siguiente paso será el de crear un esquema. Para ello, cambie la ventana de consultas a la base de datos en la que quiere crear el esquema y ejecute lo siguiente: `CREATE SCHEMA prueba;`

The screenshot illustrates the steps to create a schema in PostgreSQL using the PostgreSQL Enterprise Studio interface. The interface is divided into three main sections: a left sidebar for connections, a central query editor, and a bottom messages pane.

- Step 1:** In the left sidebar, under the 'CONNECTIONS' tree, the 'Servers' folder is expanded. The 'localhost, <default> (postgres)' connection is selected.
- Step 2:** In the central query editor, the 'Database' dropdown menu is set to 'pruebaspostgresql'. The query `CREATE SCHEMA prueba;` is entered in the editor.
- Step 3:** The 'Run' button (a green play icon) is clicked to execute the query.
- Step 4:** In the left sidebar, the 'Schemas' folder is expanded, and the newly created 'prueba' schema is visible.

The 'Messages' pane at the bottom shows the execution results: 'Started executing query at Line 1', 'CREATE SCHEMA', and 'Total execution time: 00:00:00.004'.

Creación de esquemas y tablas en PostgreSQL

- **En MySQL**, el concepto de esquema se utiliza tanto para referirse a la base de datos como un “todo”, así como a un contenedor en particular.
- **En PostgreSQL**, el término "base de datos" se refiere a un conjunto separado de datos, con sus propias tablas, funciones, etc. Dentro de cada base de datos de PostgreSQL, se pueden organizar los objetos en "esquemas".
- Un esquema en PostgreSQL es similar a un directorio en un sistema de archivos: proporciona una forma de organizar los objetos de la base de datos en grupos lógicos.



Creación de esquemas y tablas en PostgreSQL

5. Una vez creada la base de datos y el esquema, es el momento de crear una tabla en PostgreSQL usando los siguientes comandos:

```
SET search_path TO prueba;
```

```
CREATE SEQUENCE actor_actor_id_seq INCREMENT BY 1 NO MAXVALUE NO MINVALUE CACHE 1;
```

```
CREATE TABLE actor (  
    actor_id integer DEFAULT nextval('actor_actor_id_seq'::regclass) NOT NULL,  
    first_name character varying(45) NOT NULL,  
    last_name character varying(45) NOT NULL,  
    last_update timestamp without time zone DEFAULT now() NOT NULL  
);
```

```
ALTER TABLE ONLY actor ADD CONSTRAINT actor_pkey PRIMARY KEY (actor_id);
```

Creación de esquemas y tablas en PostgreSQL

5. Una vez creada la base de datos y el esquema, es el momento de crear una tabla en PostgreSQL similar a sakila usando los siguientes comandos:

The screenshot shows a PostgreSQL client interface with three numbered callouts:

- 1**: Points to the 'Database' dropdown menu, which is set to 'pruebaspostgresql'.
- 2**: Points to the 'Run' button, which is highlighted with a green play icon.
- 3**: Points to the 'actor' table in the 'public' schema, which is selected in the left-hand pane.

The SQL query editor displays the following commands:

```
CREATE SEQUENCE actor_actor_id_seq
INCREMENT BY 1
NO MAXVALUE
NO MINVALUE
CACHE 1;

CREATE TABLE actor (
  actor_id integer DEFAULT nextval('actor_actor_id_seq'::regclass) NOT NULL,
  first_name character varying(45) NOT NULL,
  last_name character varying(45) NOT NULL,
  last_update timestamp without time zone DEFAULT now() NOT NULL
);

ALTER TABLE ONLY actor ADD CONSTRAINT actor_pkey PRIMARY KEY (actor_id);
```

The left-hand pane shows the 'CONNECTIONS' tree with the following structure:

- SERVERS
 - L03524891L01, <default...>
 - sql-server-test.c3kg5ve...
 - localhost, sakila (root)
 - localhost, <default> (po...)
 - Databases
 - postgres
 - pruebaspostgresql
 - Schemas
 - System
 - prueba
 - public
 - Tables
 - actor

Creación de esquemas y tablas en PostgreSQL

6. Creamos otra tabla similar a sakila para posteriormente hacer referencia entre tablas por medio de una llave foránea:

```
SET search_path TO prueba;
```

```
CREATE DOMAIN year AS integer  
CONSTRAINT year_check CHECK (((VALUE >= 1901) AND (VALUE <= 2155)));
```

```
CREATE TYPE mpaa_rating AS ENUM ('G','PG','PG-13','R', 'NC-17');
```

```
CREATE SEQUENCE film_film_id_seq INCREMENT BY 1 NO MAXVALUE NO MINVALUE  
CACHE 1;
```

Creación de esquemas y tablas en PostgreSQL

6. Creamos otra tabla similar a sakila para posteriormente hacer referencia entre tablas por medio de una llave foránea:

```
CREATE TABLE film (  
    film_id integer DEFAULT nextval('film_film_id_seq'::regclass) NOT NULL,  
    title character varying(255) NOT NULL,  
    description text,  
    release_year year,  
    language_id integer NOT NULL,  
    original_language_id integer,  
    rental_duration smallint DEFAULT 3 NOT NULL,  
    rental_rate numeric(4,2) DEFAULT 4.99 NOT NULL,  
    length smallint,  
    replacement_cost numeric(5,2) DEFAULT 19.99 NOT NULL,  
    rating mpaa_rating DEFAULT 'G'::mpaa_rating,  
    last_update timestamp without time zone DEFAULT now() NOT NULL,  
    special_features text[],  
    fulltext tsvector);
```

```
ALTER TABLE ONLY film ADD CONSTRAINT film_pkey PRIMARY KEY (film_id);
```

Creación de esquemas y tablas en PostgreSQL

6. Creamos otra tabla similar a sakila para posteriormente hacer referencia entre tablas por medio de una llave foránea:

The screenshot displays the PostgreSQL GUI interface. On the left, the 'CONNECTIONS' pane shows a tree view of servers and databases. The 'pruebaspostgresql' database is selected, and the 'prueba' schema is expanded, showing the 'film' table. An orange arrow labeled '3' points to the 'film' table. In the center, the 'SQLQuery_1 - localh...stgres)' window shows the following SQL code:

```
SET search_path TO prueba;  
  
CREATE DOMAIN year AS integer  
    CONSTRAINT year_check CHECK (((VALUE >= 1901) AND (VALUE <= 2155)));  
  
CREATE TYPE mpaa_rating AS ENUM ('G','PG','PG-13','R', 'NC-17');  
  
CREATE SEQUENCE film_film_id_seq INCREMENT BY 1 NO MAXVALUE NO MINVALUE  
    CACHE 1;  
  
CREATE TABLE film (  
    film_id integer DEFAULT nextval('film_film_id_seq'::regclass) NOT NULL,  
    title character varying(255) NOT NULL,  
    description text,  
    release_year year,
```

An orange arrow labeled '2' points to the 'Run' button. On the right, the 'Database:' dropdown menu is set to 'pruebaspostgresql', with an orange arrow labeled '1' pointing to it. At the bottom, the 'Messages' pane shows the execution status: 'Started executing query at Line 11' and 'CREATE TABLE'.

Tecnológico de Monterrey

Creación de esquemas y tablas en PostgreSQL

7. Creamos una tabla intermedia entre la tabla actor y film que contenga una llave foránea entre ambas:

```
SET search_path TO prueba;
```

```
CREATE TABLE film_actor (  
    actor_id integer NOT NULL,  
    film_id integer NOT NULL,  
    last_update timestamp without time zone DEFAULT now() NOT NULL  
);
```

```
ALTER TABLE ONLY film_actor  
    ADD CONSTRAINT film_actor_pkey PRIMARY KEY (actor_id, film_id);
```

```
ALTER TABLE ONLY film_actor  
    ADD CONSTRAINT film_actor_actor_id_fkey FOREIGN KEY (actor_id)  
    REFERENCES actor(actor_id) ON UPDATE CASCADE ON DELETE RESTRICT;
```


Creación de esquemas y tablas en PostgreSQL

7. Creamos una tabla intermedia entre la tabla actor y film que contenga una llave foránea entre ambas:

The screenshot displays the PostgreSQL GUI interface. On the left, the 'CONNECTIONS' pane shows a tree view of servers and databases. The 'pruebaspostgresql' database is selected, and the 'Tables' folder is expanded, showing the 'film_actor' table being created. An orange arrow labeled '3' points to this folder.

The main SQL editor window, titled 'SQLQuery_1 - localh...stgres)', shows the following SQL code:

```
SET search_path TO prueba;  
  
CREATE TABLE film_actor (  
    actor_id integer NOT NULL,  
    film_id integer NOT NULL,  
    last_update timestamp without time zone DEFAULT now() NOT NULL  
);  
  
ALTER TABLE ONLY film_actor  
    ADD CONSTRAINT film_actor_pkey PRIMARY KEY (actor_id, film_id);  
  
ALTER TABLE ONLY film_actor  
    ADD CONSTRAINT film_actor_actor_id_fkey FOREIGN KEY (actor_id)  
    REFERENCES actor(actor_id) ON UPDATE CASCADE ON DELETE RESTRICT;
```

An orange arrow labeled '2' points to the 'Run' button, and another orange arrow labeled '1' points to the 'Database' dropdown menu, which is set to 'pruebaspostgresql'.

The 'Messages' pane at the bottom shows the execution progress:

```
5:40:11 PM Started executing query at Line 9  
ALTER TABLE  
5:40:11 PM Started executing query at Line 12
```

Inserción de datos en PostgreSQL

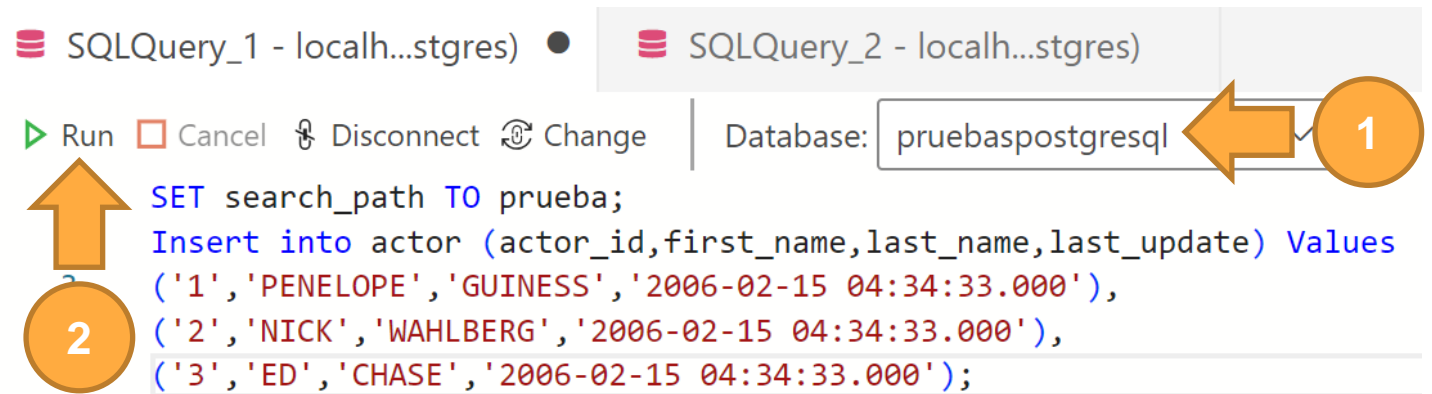
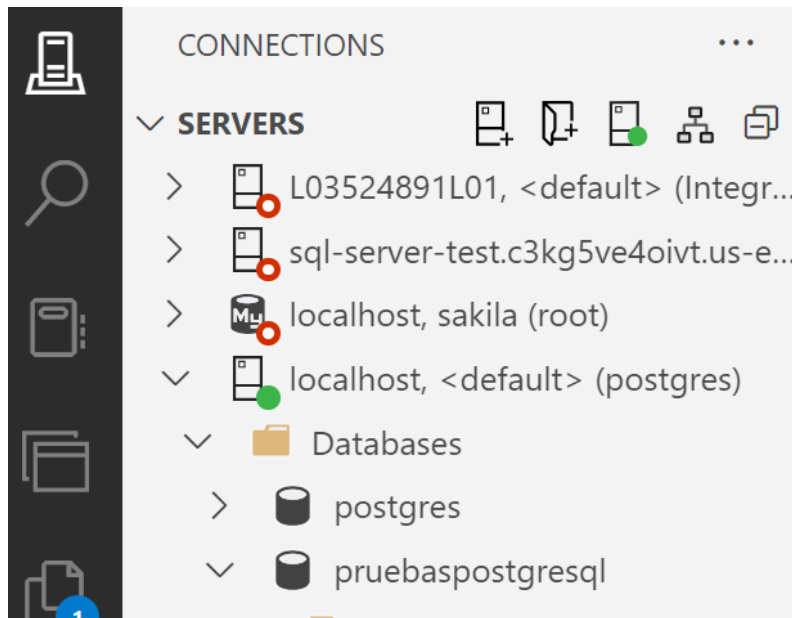
1. Con las tablas creadas en el esquema de prueba, lo siguiente es añadir información a cada una para su posterior consulta. Primera insertemos información para la tabla actor de la siguiente manera:

```
SET search_path TO prueba;
```

```
Insert into actor (actor_id,first_name,last_name,last_update) Values  
( '1', 'PENELOPE', 'GUINNESS', '2006-02-15 04:34:33.000' ),  
( '2', 'NICK', 'WAHLBERG', '2006-02-15 04:34:33.000' ),  
( '3', 'ED', 'CHASE', '2006-02-15 04:34:33.000' );
```

Inserción de datos en PostgreSQL

1. Con las tablas creadas en el esquema de prueba, lo siguiente es añadir información a cada una para su posterior consulta. Primera insertemos información para la tabla actor de la siguiente manera:



Inserción de datos en PostgreSQL

1. Con las tablas creadas en el esquema de prueba, lo siguiente es añadir información a cada una para su posterior consulta. Primera insertemos información para la tabla actor de la siguiente manera:

The screenshot shows the PostgreSQL DBeaver interface. On the left, the 'SERVERS' tree is expanded to 'localhost, <default> (postgres)' > 'Databases' > 'pruebaspostgresql' > 'Schemas' > 'prueba' > 'Tables' > 'actor'. A blue circle with the number '1' is next to the 'actor' table. A blue arrow labeled '3' points from the 'actor' table to the SQL editor. The SQL editor contains the following query:

```
1 select actor_id
2 ,first_name
3 ,last_name
4 ,last_update
5 from prueba.actor
6 LIMIT 1000
```

A blue arrow labeled '4' points from the 'actor' table to the 'Select Top 1000' button in the context menu. The context menu also shows 'Script as Create', 'Script as Drop', and 'Refresh'. The 'Messages' tab shows the results of the query:

actor_id	first_name	last_name	last_update
	PENELOPE	GUINNESS	2006-02-15 04:34:33
	NICK	WAHLBERG	2006-02-15 04:34:33
	ED	CHASE	2006-02-15 04:34:33

A blue arrow labeled '5' points from the 'Messages' tab to the results table.

Inserción de datos en PostgreSQL

2. De manera análoga, se añade información sobre la tabla film de la siguiente manera:

```
SET search_path TO prueba;
```

```
Insert into film (film_id,title,description,release_year,language_id,original_language_id,  
                rental_duration,rental_rate,length,replacement_cost,rating,  
                special_features,last_update) Values
```

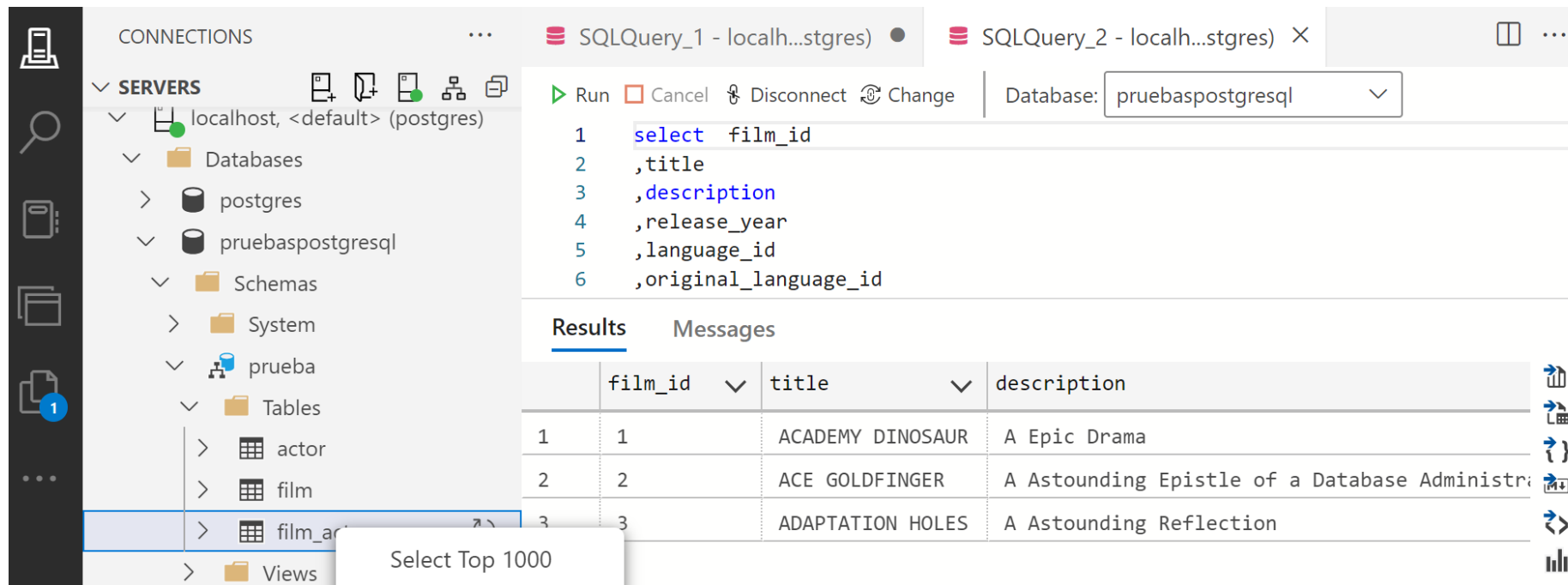
```
('1','ACADEMY DINOSAUR','A Epic Drama','2006','1',1,'6','0.99','86','20.99','PG',string_to_array('Deleted  
Scenes',' ',''),'2006-02-15 05:03:42.000'),
```

```
('2','ACE GOLDFINGER','A Astounding Epistle of a Database  
Administrator','2006','1',1,'3','4.99','48','12.99','G',string_to_array('Trailers,Deleted  
Scenes',' ',''),'2006-02-15 05:03:42.000'),
```

```
('3','ADAPTATION HOLES','A Astounding Reflection','2006','1',1,'7','2.99','50','18.99','NC-  
17',string_to_array('Trailers,Deleted Scenes',' ',''),'2006-02-15 05:03:42.000');
```

Inserción de datos en PostgreSQL

2. De manera análoga, se añade información sobre la tabla film de la siguiente manera:



The screenshot shows a database management interface with a sidebar on the left displaying a tree view of the database structure. The main area displays a SQL query and its results.

Database Structure:

- SERVERS
 - localhost, <default> (postgres)
 - Databases
 - postgres
 - pruebaspostgresql
 - Schemas
 - System
 - prueba
 - Tables
 - actor
 - film
 - film_ar
 - Views

SQL Query:

```
1 select film_id
2 ,title
3 ,description
4 ,release_year
5 ,language_id
6 ,original_language_id
```

Results:

	film_id	title	description
1	1	ACADEMY DINOSAUR	A Epic Drama
2	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administr
3	3	ADAPTATION HOLES	A Astounding Reflection

A tooltip "Select Top 1000" is visible over the "film_ar" table in the sidebar.

Inserción de datos en PostgreSQL

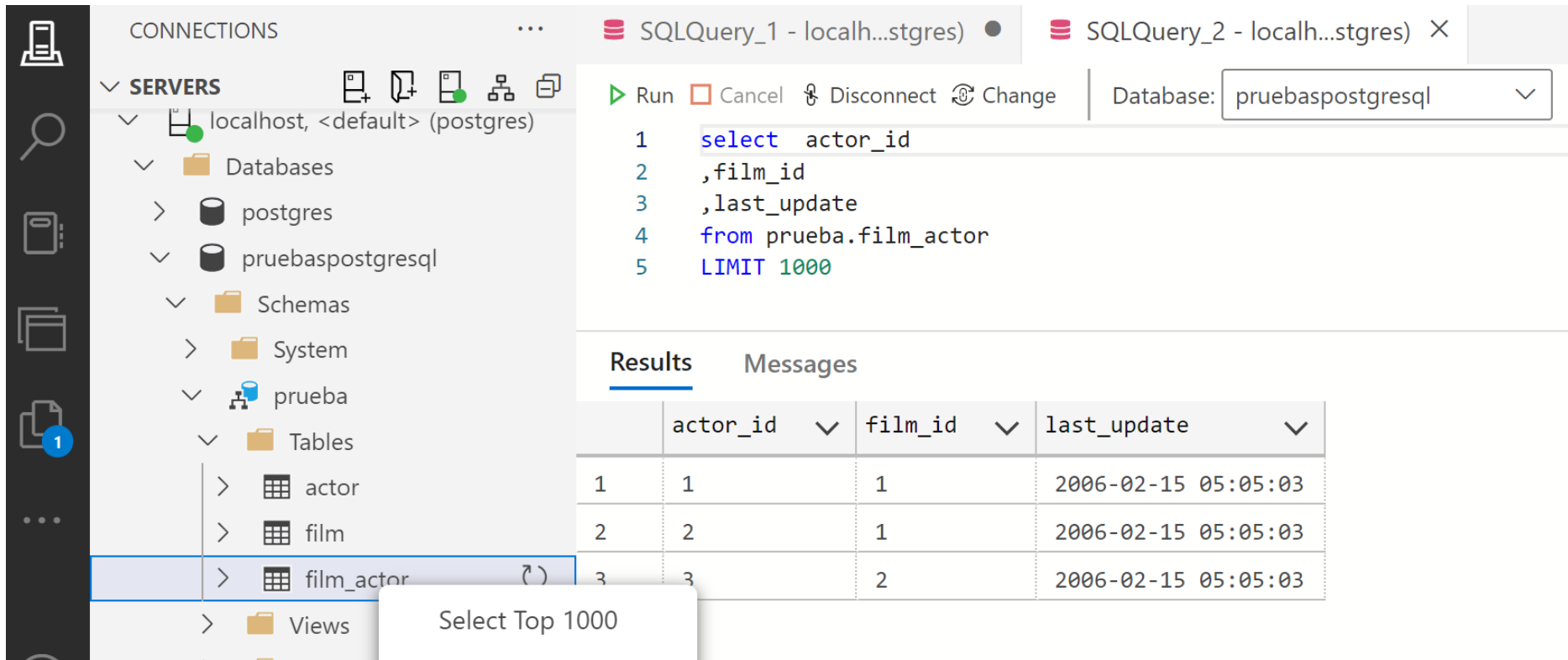
3. También se agrega información a la tabla `film_actor` de la siguiente manera:

```
SET search_path TO prueba;
```

```
Insert into film_actor (actor_id,film_id,last_update) Values  
( '1', '1', '2006-02-15 05:05:03.000' ),  
( '2', '1', '2006-02-15 05:05:03.000' ),  
( '3', '2', '2006-02-15 05:05:03.000' );
```

Inserción de datos en PostgreSQL

3. También se agrega información a la tabla `film_actor` de la siguiente manera:



The screenshot displays a PostgreSQL client interface. On the left, the 'CONNECTIONS' pane shows a tree view of the database structure: **SERVERS** > **localhost, <default> (postgres)** > **Databases** > **pruebaspostgresql** > **Schemas** > **prueba** > **Tables** > **film_actor**. The **film_actor** table is selected, and a context menu is open with the option 'Select Top 1000'. The main query editor shows the following SQL query:

```
1 select actor_id
2 ,film_id
3 ,last_update
4 from prueba.film_actor
5 LIMIT 1000
```

The query is executed, and the results are displayed in the 'Results' pane. The results table has four columns: **actor_id**, **film_id**, and **last_update**. The first three rows of data are shown:

	actor_id	film_id	last_update
1	1	1	2006-02-15 05:05:03
2	2	1	2006-02-15 05:05:03
3	3	2	2006-02-15 05:05:03

Consulta de datos en PostgreSQL

1. Con las tablas creadas y un poco de información, el siguiente paso es el de hacer algunas consultas básicas:

```
SET search_path TO prueba;  
SELECT actor_id, first_name, last_name  
FROM actor  
WHERE last_name LIKE 'G%';
```

Results Messages

	actor_id ▼	first_name ▼	last_name ▼
1	1	PENELOPE	GUINNESS

Consulta de datos en PostgreSQL

2. Con las tablas creadas y un poco de información, el siguiente paso es el de hacer algunas consultas básicas:

```
SET search_path TO prueba;  
SELECT film_id, title, description, release_year  
FROM film  
WHERE release_year >= 2005 AND release_year <= 2010;
```

	film_id	title	description
1	1	ACADEMY DINOSAUR	A Epic Drama
2	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administr
3	3	ADAPTATION HOLES	A Astounding Reflection

Consulta de datos en PostgreSQL

3. Con las tablas creadas y un poco de información, el siguiente paso es el de hacer algunas consultas básicas:

```
SET search_path TO prueba;
```

```
SELECT actor.first_name, actor.last_name, film.title
```

```
FROM actor
```

```
INNER JOIN film_actor ON actor.actor_id = film_actor.actor_id
```

```
INNER JOIN film ON film_actor.film_id = film.film_id;
```

	first_name ▼	last_name ▼	title ▼
1	PENELOPE	GUINNESS	ACADEMY DINOSAUR
2	NICK	WAHLBERG	ACADEMY DINOSAUR
3	ED	CHASE	ACE GOLDFINGER

Creación de objetos en PostgreSQL

1. De manera análoga a la creación de consultas, con las tablas creadas se pueden crear distintos objetos de la base de datos como **vistas (views)** de la siguiente manera:

```
SET search_path TO prueba;
```

```
CREATE VIEW vista_film_con_actores AS  
SELECT f.film_id, f.title, f.release_year, f.length,  
       COUNT(fa.actor_id) AS total_actores  
FROM film f  
LEFT JOIN film_actor fa ON f.film_id = fa.film_id  
GROUP BY f.film_id, f.title, f.release_year, f.length;
```

```
SELECT * FROM vista_film_con_actores;
```

Creación de objetos en PostgreSQL

1. De manera análoga a la creación de consultas, con las tablas creadas se pueden crear distintos objetos de la base de datos como **vistas (views)** de la siguiente manera:

The screenshot displays a PostgreSQL client interface with a sidebar on the left showing a tree view of the database structure. The tree includes 'Servers', 'Databases', 'Schemas', 'Tables', 'Views', 'Materialized Views', 'Functions', and 'Procedures'. The 'Tables' folder is expanded, showing 'actor', 'film', and 'film_actor'. The 'Views' folder is also visible. The main window shows a SQL query editor with the following code:

```
SET search_path TO prueba;  
  
CREATE VIEW vista_film_con_actores AS  
SELECT f.film_id, f.title, f.release_year, f.length,  
       COUNT(fa.actor_id) AS total_actores  
FROM film f  
LEFT JOIN film_actor fa ON f.film_id = fa.film_id  
GROUP BY f.film_id, f.title, f.release_year, f.length;  
  
SELECT * FROM vista_film_con_actores;
```

Three orange arrows with numbers 1, 2, and 3 point to specific elements: Arrow 1 points to the 'Database: pruebaspostgresql' dropdown menu. Arrow 2 points to the 'Run' button. Arrow 3 points to the 'Results' tab.

The 'Results' tab shows the following data:

film_id	title	release_year	length	total_actores
2	ACE GOLDFINGER	2006	48	1
3	ADAPTATION HOLES	2006	50	0
1	ACADEMY DINOSAUR	2006	86	2

Creación de objetos en PostgreSQL

2. De manera análoga a la creación de consultas, con las tablas creadas se pueden crear distintos objetos de la base de datos como **disparadores (triggers)** de la siguiente manera:

```
SET search_path TO prueba;
CREATE OR REPLACE FUNCTION imprimir_mensaje_insert()
RETURNS TRIGGER AS $$
BEGIN
    RAISE NOTICE 'Nuevo actor insertado con actor_id: %', NEW.actor_id;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_imprimir_mensaje_insert
AFTER INSERT ON actor
FOR EACH ROW
EXECUTE FUNCTION imprimir_mensaje_insert();
```

Creación de objetos en PostgreSQL

2. De manera análoga a la creación de consultas, con las tablas creadas se pueden crear distintos objetos de la base de datos como **disparadores (triggers)** de la siguiente manera:

The screenshot displays a PostgreSQL client interface with the following components:

- Left Sidebar (Tree View):** Shows the database structure. The 'pruebaspostgresql' database is selected, and the 'actor' table is highlighted under the 'prueba' schema.
- SQL Editor:** Contains the following SQL code:

```
CREATE OR REPLACE FUNCTION imprimir_mensaje_insert()  
RETURNS TRIGGER AS $$  
BEGIN  
    RAISE NOTICE 'Nuevo actor insertado con actor_id: %', NEW.actor_id;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
CREATE TRIGGER trigger_imprimir_mensaje_insert  
AFTER INSERT ON actor  
FOR EACH ROW  
EXECUTE FUNCTION imprimir_mensaje_insert();
```
- Database Dropdown:** Located at the top right of the SQL editor, showing 'pruebaspostgresql'.
- Run Button:** A green play button icon located below the SQL code.
- Messages Pane:** Located at the bottom, showing the execution results:

```
7:00:17 PM Started executing query at Line 1  
CREATE FUNCTION  
7:00:17 PM Started executing query at Line 9  
CREATE TRIGGER  
Total execution time: 00:00:00.004
```

Three orange callouts with numbers 1, 2, and 3 point to the database dropdown, the 'Run' button, and the messages pane, respectively.

Creación de objetos en PostgreSQL

3. De manera análoga a la creación de consultas, con las tablas creadas se pueden crear distintos objetos de la base de datos como un **procedimiento almacenado (stored procedure)** de la siguiente manera:

```
SET search_path TO prueba;
```

```
CREATE OR REPLACE FUNCTION contar_peliculas()
```

```
RETURNS INTEGER AS $$
```

```
DECLARE
```

```
    total INTEGER;
```

```
BEGIN
```

```
    SELECT COUNT(*) INTO total FROM film;
```

```
    RETURN total;
```

```
END;
```

```
$$ LANGUAGE plpgsql;
```

```
SELECT contar_peliculas();
```


Creación de objetos en PostgreSQL

3. De manera análoga a la creación de consultas, con las tablas creadas se pueden crear distintos objetos de la base de datos como un **procedimiento almacenado (stored procedure)** de la siguiente manera:

The screenshot displays the PostgreSQL GUI interface. On the left, the 'CONNECTIONS' pane shows a tree view of the database structure. The 'pruebaspostgresql' database is selected, and the 'actor' table is highlighted under the 'prueba' schema. An orange arrow labeled '1' points to the 'Database:' field in the top right, which is set to 'pruebaspostgresql'. Another orange arrow labeled '2' points to the 'Run' button in the top toolbar. The main SQL editor contains the following code:

```
SET search_path TO prueba;  
  
CREATE OR REPLACE FUNCTION contar_peliculas()  
RETURNS INTEGER AS $$  
DECLARE  
    total INTEGER;  
BEGIN  
    SELECT COUNT(*) INTO total FROM film;  
    RETURN total;  
END;  
$$ LANGUAGE plpgsql;  
  
SELECT contar_peliculas();
```

An orange arrow labeled '3' points to the 'SELECT' statement at the bottom of the code. Below the SQL editor, the 'Results' pane shows the output of the query:

	contar_peliculas
1	3

Elementos extras

- Para poder trabajar con PostgreSQL, se recomienda revisar varios ejemplos.
- En específico, se puede utilizar como referencia/inspiración, la versión de [sakila](#) provista en CANVAS, aunque no es recomendable ejecutarla ya que las inserciones no fueron creadas en un formato por lotes y pueden tomar mucho tiempo.
- En el caso de los eventos (events), estos no existen como tal en PostgreSQL por lo que solo deben limitarse a su uso en MySQL.



Elementos extras

- Todo lo realizado anteriormente puede ser implementado de igual manera en pgadmin de PostgreSQL, aunque su uso es mucho mas grafico y esta pensado para una sesión completa sobre este manejador de base de datos.
- En caso de error con Azure Data Studio, en el cual se mencione que la base de datos esta siendo ocupada por otro usuario, se recomienda reinicializar la aplicación para cerrar cualquier sesión de comandos abierta.
- En caso de cualquier duda extra, se recomienda el siguiente [tutorial](#) de pgadmin.



Referencias

- Sommerville, I., Software Engineering, 10th Edition, Pearson, 2016, IN, 1292096144, 9781292096148.
- Connolly Thomas M, Database systems : a practical approach to design, implementation and management, 5thed., London : Addison-Wesley, 2010, 9780321523068.
- Perez, C., MySQL para windows y Linux, España, Alfaomega, 2004.
- <https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html>

Gracias!
Preguntas...



Dr. Esteban Castillo Juarez

Google academics:

<https://scholar.google.com/citations?user=JfZpVO8AAAAJ&hl=en>

<https://dblp.uni-trier.de/pers/hd/c/Castillo:Esteban>