

Construcción de software y toma de decisiones

TC2005B

Dr. Esteban Castillo Juarez

ITESM, Campus Santa Fe



esteban.castillojz@tec.mx

Agenda

- Variables de usuario
- Eventos
- Estructuras de decisión
- Estructuras de repetición
- Transacciones

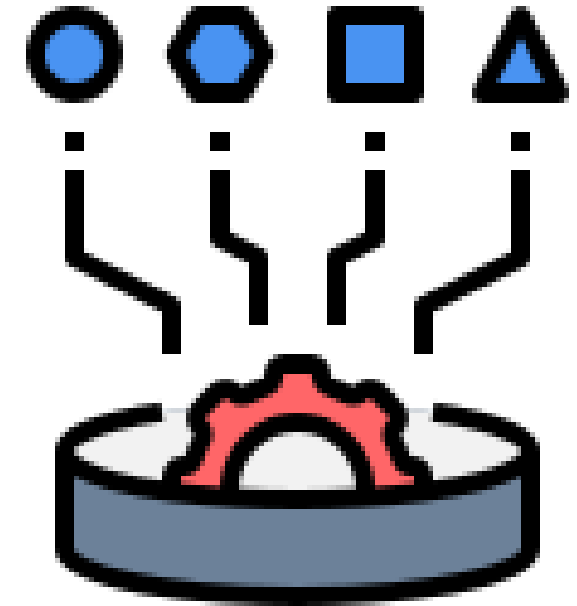
Variables de usuario

- En MySQL, una variable de usuario es una forma de almacenar temporalmente un valor durante la ejecución de una sesión de administración de datos.
- Estas variables pueden ser utilizadas para almacenar valores que se desean utilizar en múltiples consultas sin tener que repetir información.
- Las variables de usuario en MySQL se pueden declarar y utilizar en varias partes de una operación, como en sentencias `SELECT`, `INSERT`, `UPDATE`, `DELETE`, etc.



Variables de usuario

- Pueden ser útiles para almacenar resultados intermedios, configuraciones personalizadas o valores que se reutilizarán en varias consultas dentro de una misma sesión.
- Las variables pueden contener valores de diferentes tipos de datos admitidos por MySQL, como números, cadenas de texto, fechas, entre otros.
- Las variables en MySQL se declaran utilizando el símbolo @ seguido del nombre de la variable y el operador de asignación :=.



Variables de usuario

- Es importante tener en cuenta que las variables en MySQL son específicas de cada sesión.
- Lo anterior significa que una variable definida en una sesión no estará disponible en otras sesiones. Además, las variables se pierden al finalizar la aplicación de MySQL, es decir, no persisten más allá.
- MySQL proporciona variables de sistema y de sesión predefinidas que se pueden utilizar para varios propósitos, como configurar el comportamiento del servidor, controlar el entorno de ejecución de las consultas, entre otros.



Variables de usuario

```
USE sakila;  
-- Consultamos la tabla clientes  
SELECT * FROM sakila.customer;  
-- Declaramos algunas variables  
SET @nombre_cliente = 'MARY';  
SET @apellido_cliente = 'SMITH';
```

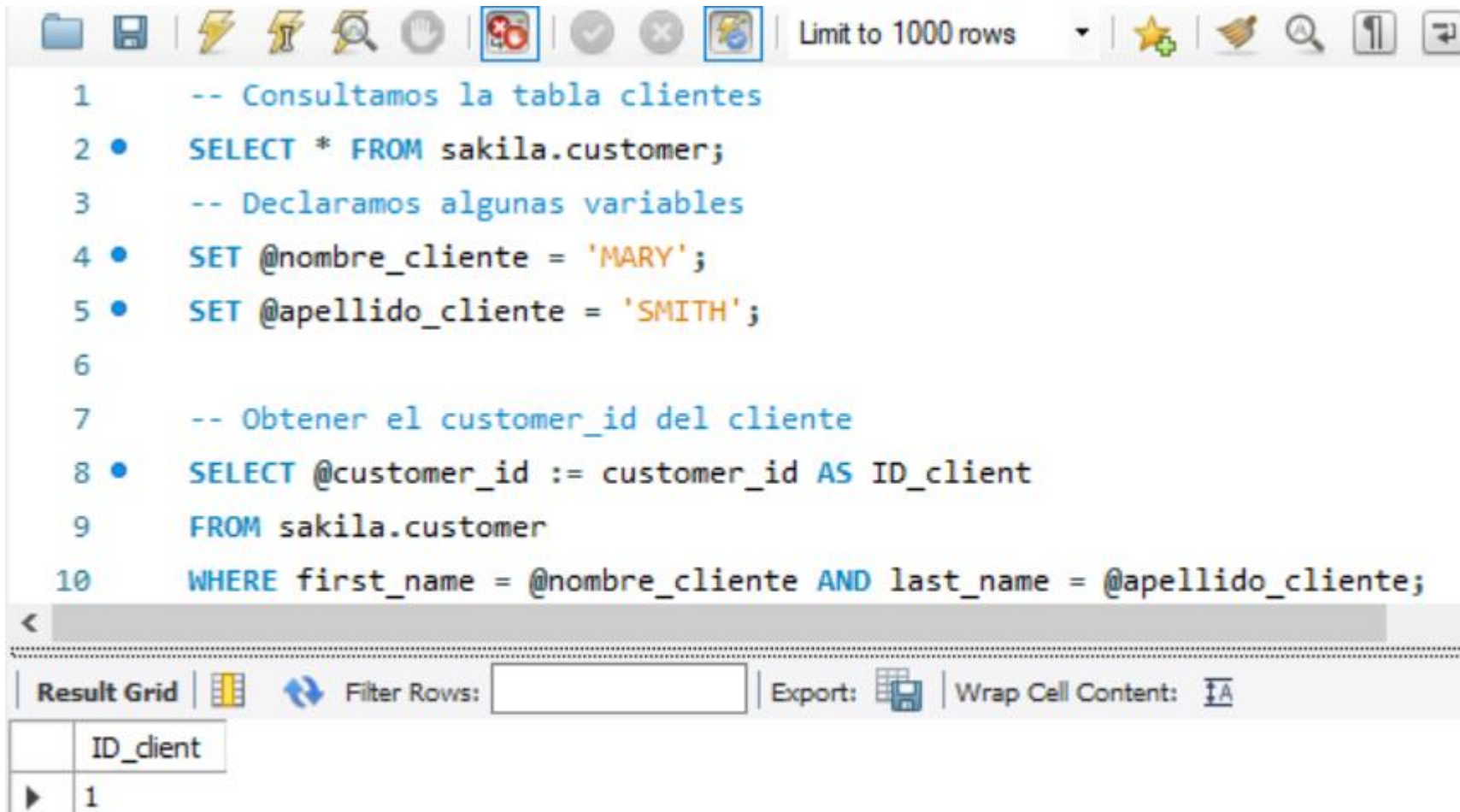
```
-- Obtener el customer_id del cliente  
SELECT @customer_id := customer_id AS ID_client  
FROM sakila.customer  
WHERE first_name = @nombre_cliente AND last_name = @apellido_cliente;
```

```
-- Obtener el nombre del cliente y email usando la variable customer_id  
SELECT CONCAT(first_name, ' ', last_name) AS complete_name, email  
FROM sakila.customer  
WHERE customer_id = @customer_id;
```

Supongamos que queremos obtener el nombre completo (nombre y apellido) de un cliente específico en la base de datos Sakila utilizando variables:

Variables de usuario

- Supongamos que queremos obtener el nombre completo (nombre y apellido) de un cliente específico en la base de datos Sakila utilizando variables:

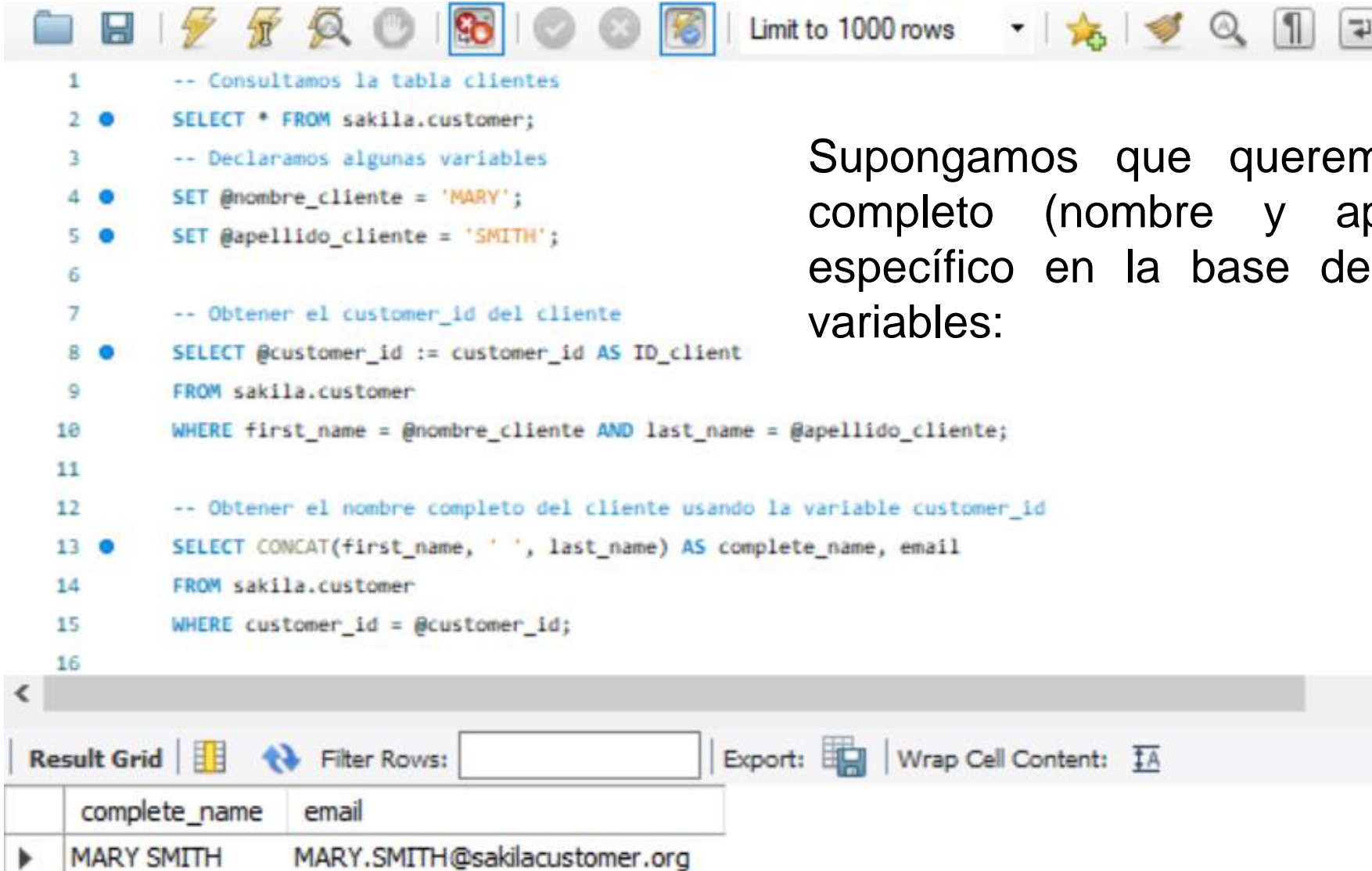


```
1  -- Consultamos la tabla clientes
2  • SELECT * FROM sakila.customer;
3  -- Declaramos algunas variables
4  • SET @nombre_cliente = 'MARY';
5  • SET @apellido_cliente = 'SMITH';
6
7  -- Obtener el customer_id del cliente
8  • SELECT @customer_id := customer_id AS ID_client
9  FROM sakila.customer
10 WHERE first_name = @nombre_cliente AND last_name = @apellido_cliente;
```

Result Grid

ID_client
1

Variables de usuario



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The main area contains a SQL script with 16 lines of code. The script first selects all data from the 'sakila.customer' table, then declares two user variables: '@nombre_cliente' with the value 'MARY' and '@apellido_cliente' with the value 'SMITH'. It then uses a subquery to find the 'customer_id' for the customer with that name and last name, storing it in '@customer_id'. Finally, it selects the concatenated first and last names (aliased as 'complete_name') and the email for that specific customer.

```
1  -- Consultamos la tabla clientes
2  ● SELECT * FROM sakila.customer;
3  -- Declaramos algunas variables
4  ● SET @nombre_cliente = 'MARY';
5  ● SET @apellido_cliente = 'SMITH';
6
7  -- Obtener el customer_id del cliente
8  ● SELECT @customer_id := customer_id AS ID_client
9  FROM sakila.customer
10 WHERE first_name = @nombre_cliente AND last_name = @apellido_cliente;
11
12 -- Obtener el nombre completo del cliente usando la variable customer_id
13 ● SELECT CONCAT(first_name, ' ', last_name) AS complete_name, email
14 FROM sakila.customer
15 WHERE customer_id = @customer_id;
16
```

Below the script, the 'Result Grid' shows the output of the final query. It has two columns: 'complete_name' and 'email'. The first row shows 'MARY SMITH' and 'MARY.SMITH@sakilacustomer.org'.

	complete_name	email
▶	MARY SMITH	MARY.SMITH@sakilacustomer.org

Supongamos que queremos obtener el nombre completo (nombre y apellido) de un cliente específico en la base de datos Sakila utilizando variables:

Variables de usuario

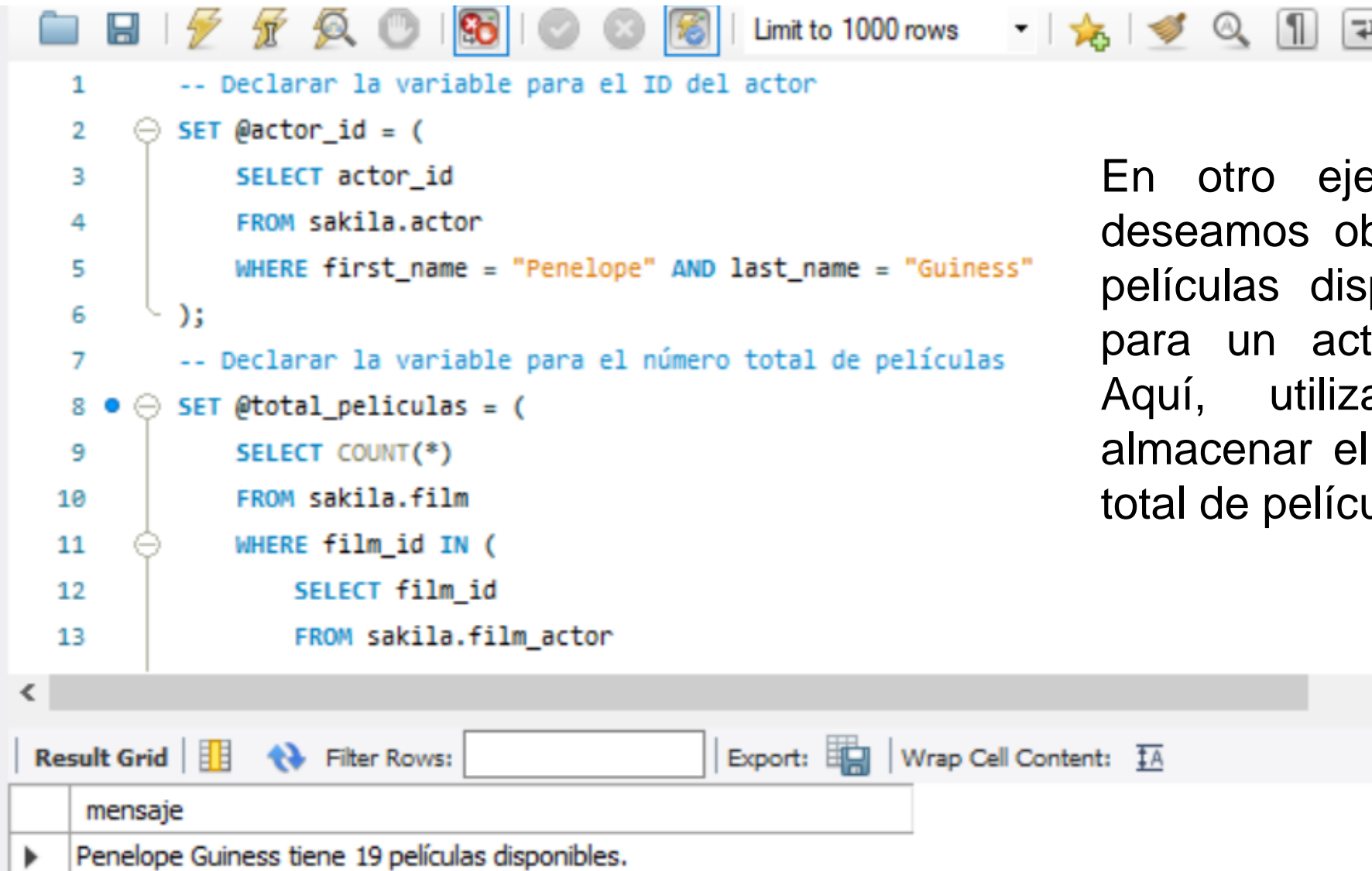
```
USE sakila;
-- Declarar la variable para el ID del actor
SET @actor_id = (
    SELECT actor_id
    FROM sakila.actor
    WHERE first_name = "Penelope" AND
           last_name = "Guinness");
```

```
-- Declarar la variable para el número total de películas
SET @total_peliculas = ( SELECT COUNT(*) FROM sakila.film
    WHERE film_id IN (
        SELECT film_id
        FROM sakila.film_actor
        WHERE actor_id = @actor_id
    ));
```

```
-- Mostrar el número total de películas disponibles para el actor
SELECT CONCAT("Penelope Guinness tiene ", @total_peliculas, " películas disponibles.") AS mensaje;
```

En otro ejemplo, supongamos que deseamos obtener el número total de películas disponibles en el inventario para un actor específico en Sakila. Aquí, utilizaremos variables para almacenar el ID del actor y el número total de películas:

Variables de usuario



```
1  -- Declarar la variable para el ID del actor
2  SET @actor_id = (
3      SELECT actor_id
4      FROM sakila.actor
5      WHERE first_name = "Penelope" AND last_name = "Guinness"
6  );
7  -- Declarar la variable para el número total de películas
8  SET @total_peliculas = (
9      SELECT COUNT(*)
10     FROM sakila.film
11     WHERE film_id IN (
12         SELECT film_id
13         FROM sakila.film_actor
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: IA

mensaje
Penelope Guinness tiene 19 películas disponibles.

En otro ejemplo, supongamos que deseamos obtener el número total de películas disponibles en el inventario para un actor específico en Sakila. Aquí, utilizaremos variables para almacenar el ID del actor y el número total de películas:

Variables de usuario

- Otro ejemplo, esta asociado al uso de la instrucción INSERT con variables, para introducir una nueva película a la base de datos Sakila:

```
USE sakila;  
-- Declarar variables para los datos de la nueva película  
SET @titulo = "Bases de datos del terror";  
SET @descripcion = "Esta es una película para la base de datos Sakila.";  
SET @ano_lanzamiento = 2024;  
SET @id_categoria = 5; -- categoría de la nueva película  
SET @id_idioma_original = 1; -- idioma inglés  
SET @duracion_renta = 3;  
SET @tarifa_renta = 2.99;  
SET @duracion_renta_ext = 7;  
SET @tarifa_renta_ext = 5.99;
```



Variables de usuario

- Otro ejemplo, esta asociado al uso de la instrucción INSERT con variables, para introducir una nueva película a la base de datos Sakila:

```
-- Insertar la nueva película en la tabla film
INSERT INTO film (title, description, release_year, language_id,
                 rental_duration, rental_rate, length, replacement_cost)
VALUES (@titulo, @descripcion, @ano_lanzamiento,
        @id_idioma_original, @duracion_renta,
        @tarifa_renta, @duracion_renta_ext,
        @tarifa_renta_ext);

-- Obtener el ID de la nueva película insertada
SET @id_nueva_pelicula = LAST_INSERT_ID();
```

Variables de usuario

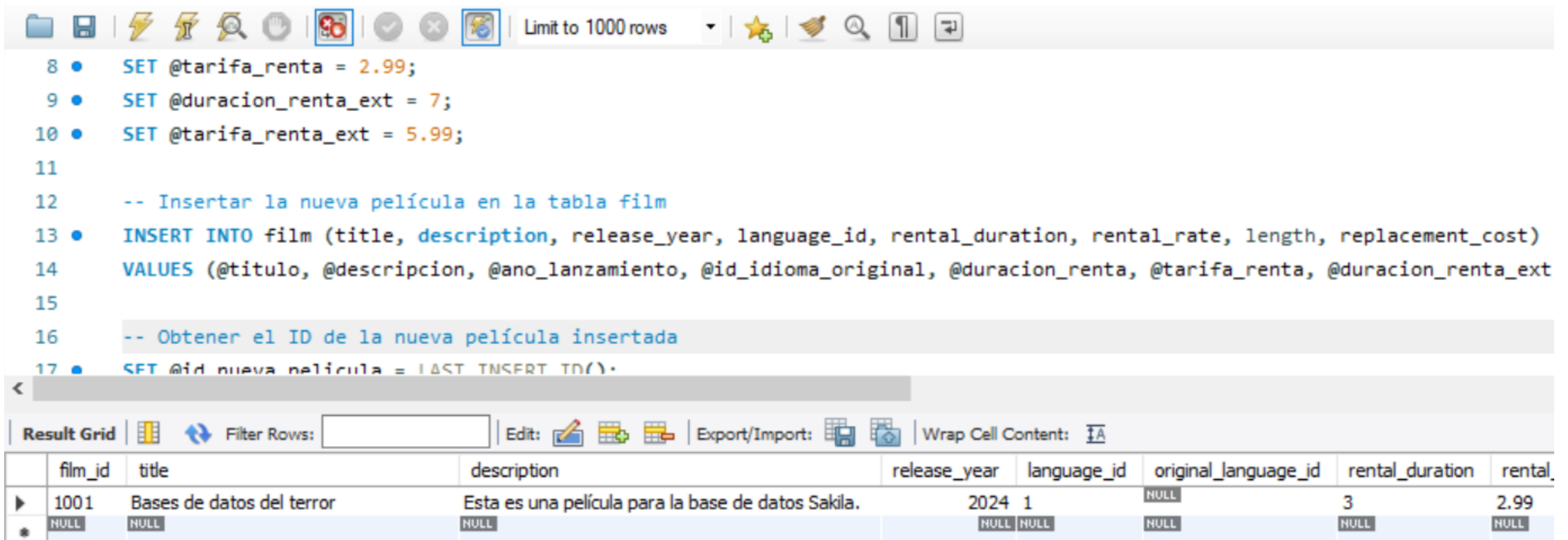
- Otro ejemplo, esta asociado al uso de la instrucción INSERT con variables, para introducir una nueva película a la base de datos Sakila:

```
-- Insertar un registro en la tabla film_category para asociar la nueva película
INSERT INTO film_category (film_id, category_id)
VALUES (@id_nueva_pelicula, @id_categoria);
```

```
-- Consultas para ver la informacion insertada
SELECT * FROM sakila.film WHERE title = "Bases de datos del terror";
SELECT * FROM sakila.film_category WHERE category_id=5;
```

Variables de usuario

- Otro ejemplo, esta asociado al uso de la instrucción INSERT con variables, para introducir una nueva película a la base de datos Sakila:



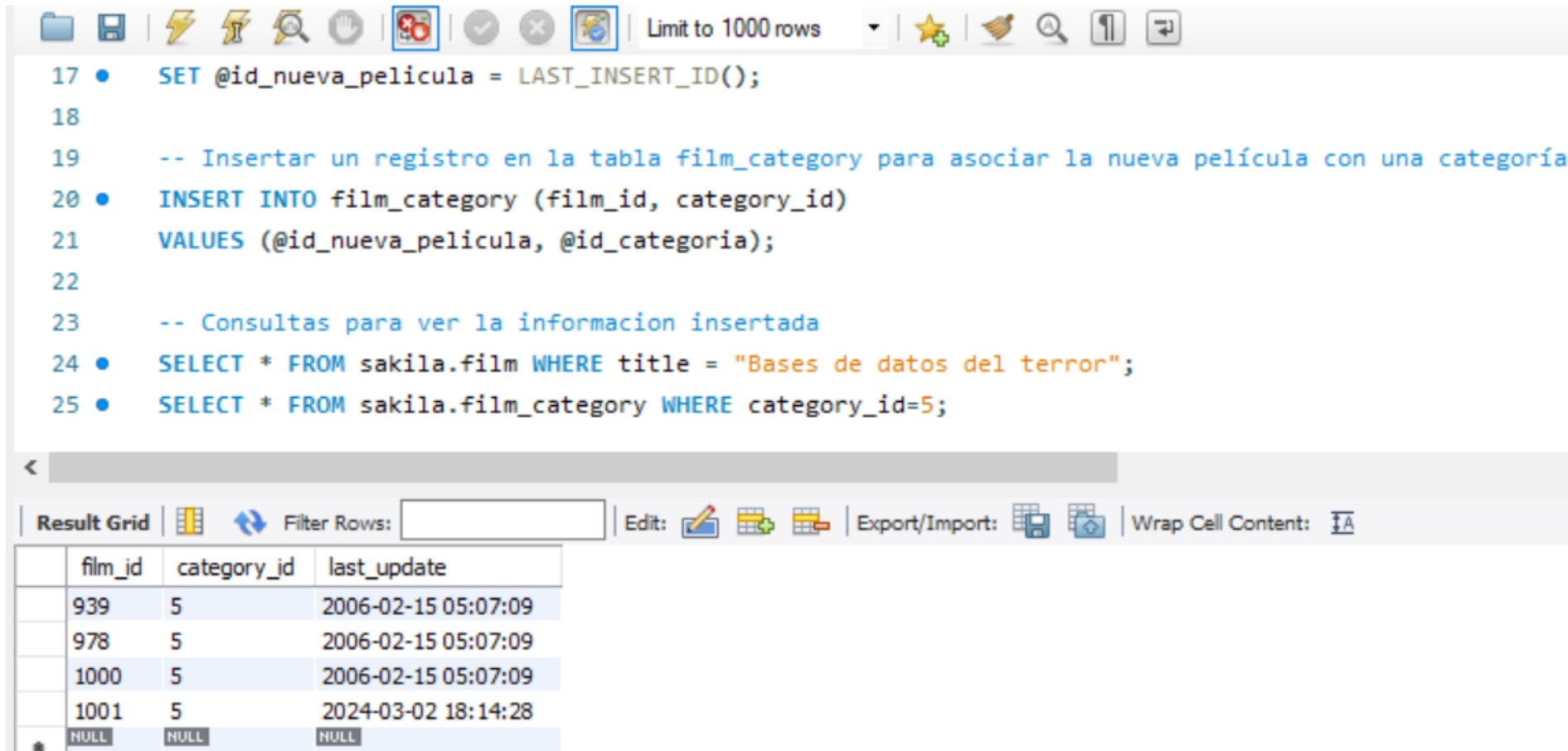
```
8 • SET @tarifa_renta = 2.99;
9 • SET @duracion_renta_ext = 7;
10 • SET @tarifa_renta_ext = 5.99;
11
12 -- Insertar la nueva película en la tabla film
13 • INSERT INTO film (title, description, release_year, language_id, rental_duration, rental_rate, length, replacement_cost)
14 VALUES (@titulo, @descripcion, @ano_lanzamiento, @id_idioma_original, @duracion_renta, @tarifa_renta, @duracion_renta_ext
15
16 -- Obtener el ID de la nueva película insertada
17 • SET @id_nueva_pelicula = LAST_INSERT_ID();
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: ☐

	film_id	title	description	release_year	language_id	original_language_id	rental_duration	rental_rate
▶	1001	Bases de datos del terror	Esta es una película para la base de datos Sakila.	2024	1	NULL	3	2.99
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Variables de usuario

- Otro ejemplo, esta asociado al uso de la instrucción INSERT con variables, para introducir una nueva película a la base de datos Sakila:



The screenshot shows a MySQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The SQL editor contains the following code:

```
17 • SET @id_nueva_pelicula = LAST_INSERT_ID();
18
19 -- Insertar un registro en la tabla film_category para asociar la nueva película con una categoría
20 • INSERT INTO film_category (film_id, category_id)
21   VALUES (@id_nueva_pelicula, @id_categoria);
22
23 -- Consultas para ver la informacion insertada
24 • SELECT * FROM sakila.film WHERE title = "Bases de datos del terror";
25 • SELECT * FROM sakila.film_category WHERE category_id=5;
```

Below the code editor is a "Result Grid" showing the output of the last query. The grid has columns for film_id, category_id, and last_update. The data is as follows:

film_id	category_id	last_update
939	5	2006-02-15 05:07:09
978	5	2006-02-15 05:07:09
1000	5	2006-02-15 05:07:09
1001	5	2024-03-02 18:14:28
NULL	NULL	NULL

Variables de usuario

- En otro ejemplo, podemos eliminar todas las películas de una categoría específica de la base de datos Sakila utilizando variables:

```
USE sakila;
```

```
-- Declarar una variable para el ID de la categoría que queremos eliminar
```

```
SET @id_categoria_eliminar = 5;
```

```
-- Eliminar todas las películas asociadas a la categoría especificada
```

```
DELETE FROM film
```

```
WHERE film_id IN (SELECT film_id
```

```
    FROM film_category
```

```
    WHERE category_id = @id_categoria_eliminar
```

```
);
```



Variables de usuario

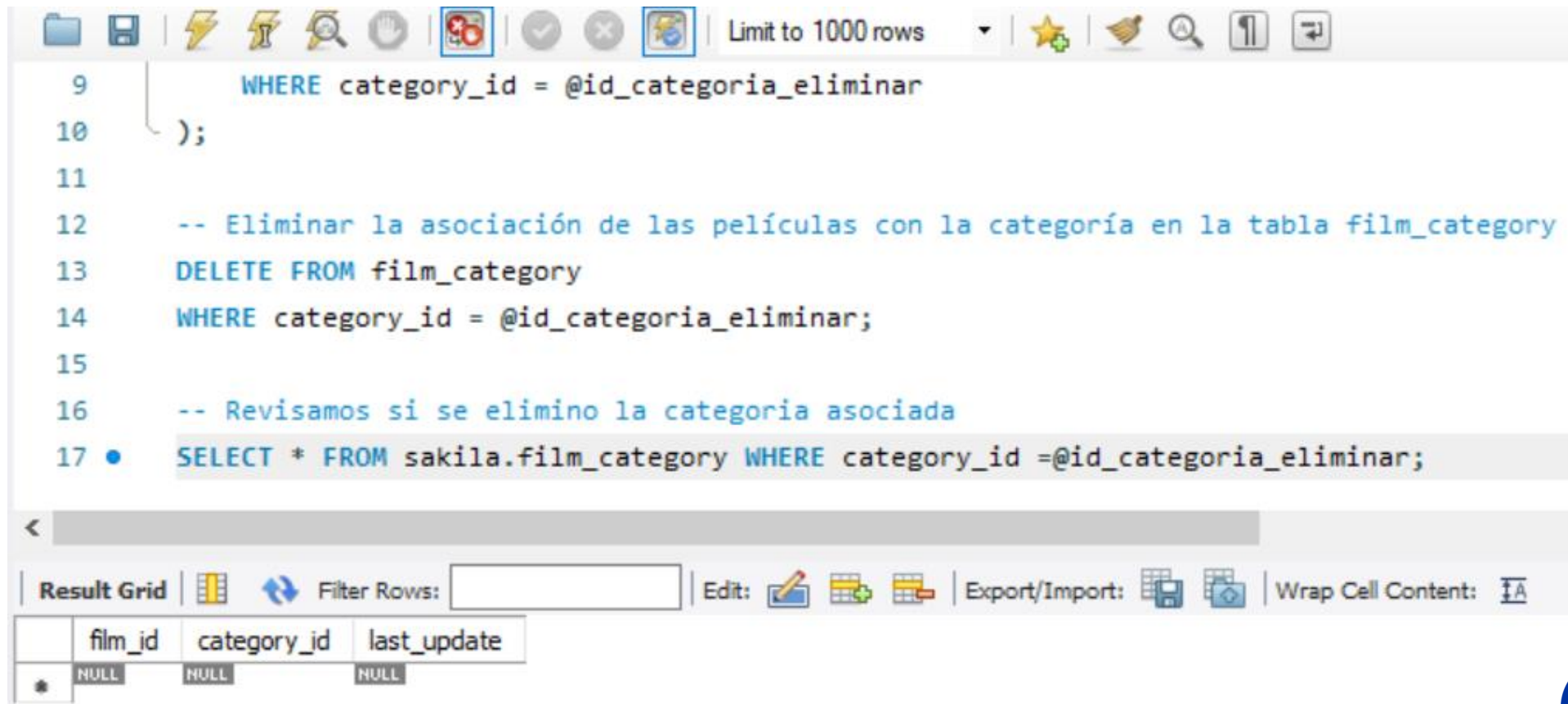
- En otro ejemplo, podemos eliminar todas las películas de una categoría específica de la base de datos Sakila utilizando variables:

```
-- Eliminar la asociación de las películas con la categoría en la tabla film_category
DELETE FROM film_category
WHERE category_id = @id_categoria_eliminar;

-- Revisamos si se elimino la categoría asociada
SELECT * FROM sakila.film_category WHERE category_id =@id_categoria_eliminar;
```

Variables de usuario

- En otro ejemplo, podemos eliminar todas las películas de una categoría específica de la base de datos Sakila utilizando variables:



```
9      WHERE category_id = @id_categoria_eliminar
10  );
11
12  -- Eliminar la asociación de las películas con la categoría en la tabla film_category
13  DELETE FROM film_category
14  WHERE category_id = @id_categoria_eliminar;
15
16  -- Revisamos si se elimino la categoria asociada
17  • SELECT * FROM sakila.film_category WHERE category_id =@id_categoria_eliminar;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content: |

	film_id	category_id	last_update
*	NULL	NULL	NULL

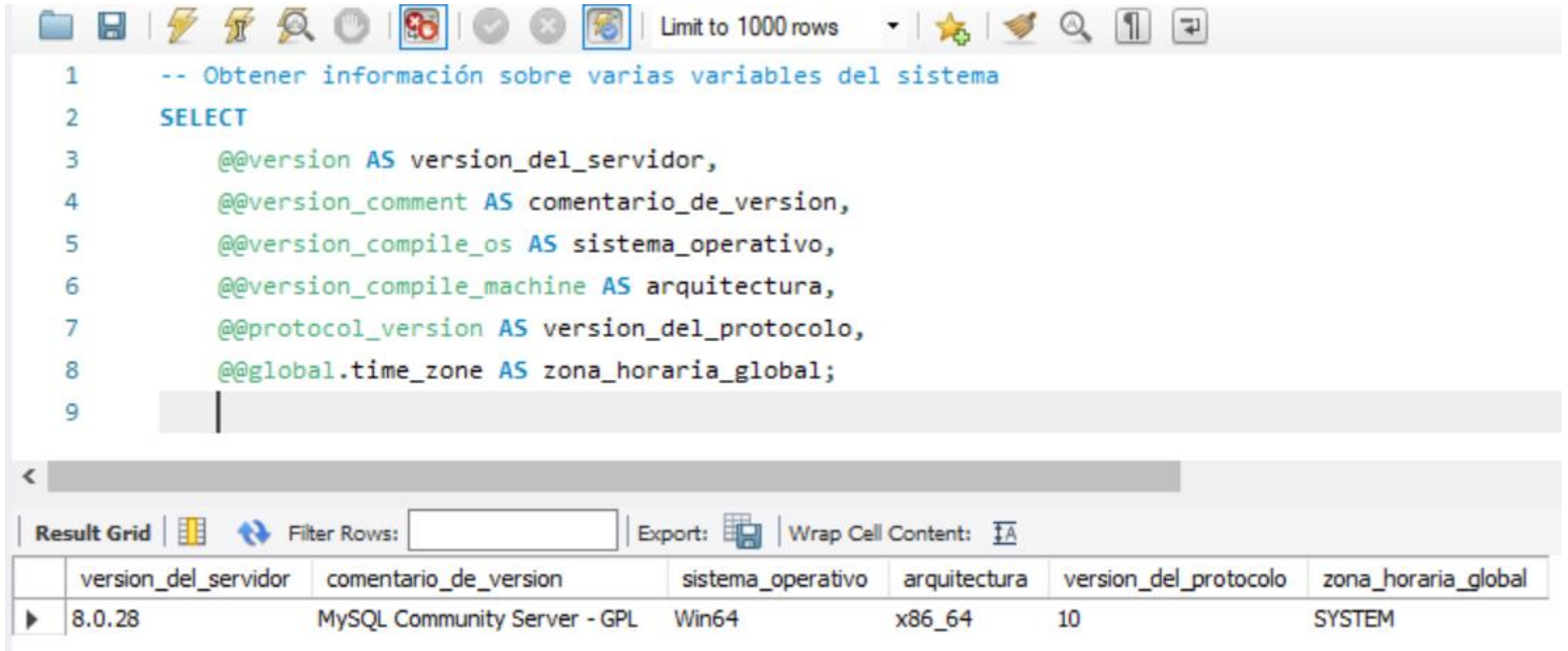
Variables de usuario

- Finalmente, veamos un ejemplo donde obtengamos la información de varias variables definidas en MySQL:

```
-- Obtener información sobre varias variables del sistema
SELECT
    @@version AS version_del_servidor,
    @@version_comment AS comentario_de_version,
    @@version_compile_os AS sistema_operativo,
    @@version_compile_machine AS arquitectura,
    @@protocol_version AS version_del_protocolo,
    @@global.time_zone AS zona_horaria_global;
```

Variables de usuario

- Finalmente, veamos un ejemplo donde obtengamos la información de varias variables definidas en MySQL:



The screenshot displays a MySQL query editor interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The query editor shows the following SQL code:

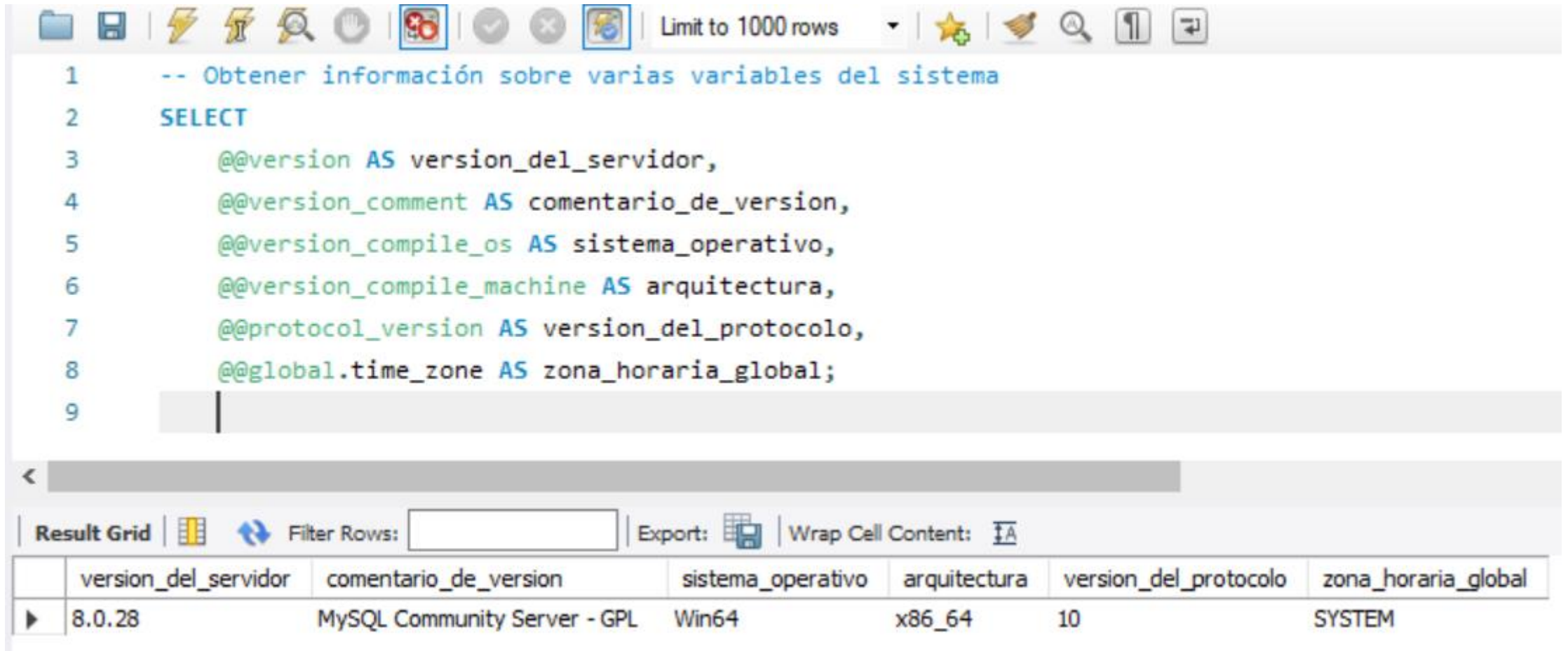
```
1  -- Obtener información sobre varias variables del sistema
2  SELECT
3      @@version AS version_del_servidor,
4      @@version_comment AS comentario_de_version,
5      @@version_compile_os AS sistema_operativo,
6      @@version_compile_machine AS arquitectura,
7      @@protocol_version AS version_del_protocolo,
8      @@global.time_zone AS zona_horaria_global;
9
```

Below the query editor is a horizontal scrollbar and a control bar with 'Result Grid', 'Filter Rows' (with an input field), 'Export', and 'Wrap Cell Content' options. The result grid below shows the output of the query:

	version_del_servidor	comentario_de_version	sistema_operativo	arquitectura	version_del_protocolo	zona_horaria_global
▶	8.0.28	MySQL Community Server - GPL	Win64	x86_64	10	SYSTEM

Variables de usuario

- Finalmente, veamos un ejemplo donde obtengamos la información de varias variables definidas en MySQL:



The screenshot displays a MySQL query editor interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The query editor contains the following SQL code:

```
1  -- Obtener información sobre varias variables del sistema
2  SELECT
3      @@version AS version_del_servidor,
4      @@version_comment AS comentario_de_version,
5      @@version_compile_os AS sistema_operativo,
6      @@version_compile_machine AS arquitectura,
7      @@protocol_version AS version_del_protocolo,
8      @@global.time_zone AS zona_horaria_global;
9
```

Below the query editor is a horizontal scrollbar and a control bar with 'Result Grid', 'Filter Rows' (with an input field), 'Export', and 'Wrap Cell Content' options. The result grid shows the output of the query:

	version_del_servidor	comentario_de_version	sistema_operativo	arquitectura	version_del_protocolo	zona_horaria_global
▶	8.0.28	MySQL Community Server - GPL	Win64	x86_64	10	SYSTEM

Eventos

- Un evento es una tarea programada que se ejecuta automáticamente en el servidor de bases de datos en un momento específico o en intervalos regulares.
- Estos eventos pueden ser utilizados para realizar diversas tareas administrativas, de mantenimiento o de procesamiento de datos de manera automática y programada, sin intervención manual por parte del usuario.
- Esto proporciona una forma conveniente de automatizar tareas recurrentes en la base de datos sin la necesidad de intervención manual continua.



Eventos

- Los eventos en MySQL se crean utilizando la sintaxis del lenguaje SQL y se almacenan en la base de datos del servidor.
- Cada evento tiene una definición que incluye información como la frecuencia de ejecución, la hora de inicio, la duración y la acción que se realizará.
- Estos eventos pueden ser activados o desactivados según sea necesario, lo que brinda flexibilidad en la gestión de tareas programadas.



Eventos

- Las tareas que se pueden realizar mediante eventos en MySQL son diversas. Por ejemplo, se pueden usar eventos para realizar copias de seguridad automáticas de la base de datos en intervalos regulares, para generar informes periódicos, para limpiar datos obsoletos o para realizar tareas de mantenimiento, como la optimización de tablas.
- Es importante tener en cuenta que, para utilizar eventos en MySQL, es necesario que el servidor de bases de datos tenga habilitado el planificador de eventos.



Eventos

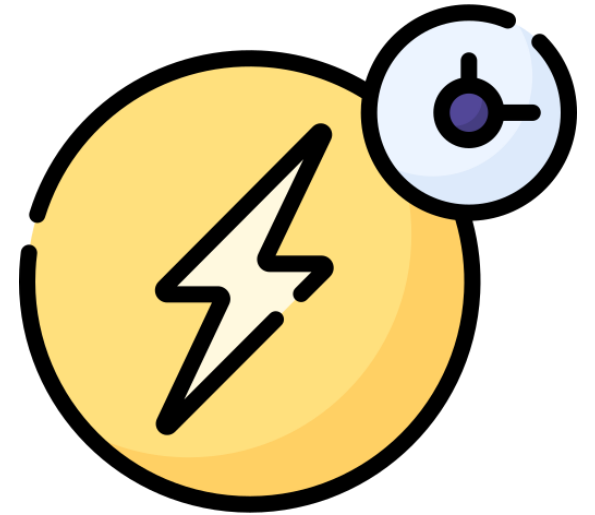
- Los eventos en MySQL ofrecen una manera conveniente de automatizar tareas periódicas en la base de datos, lo que ayuda a mejorar la eficiencia operativa y a reducir la carga de trabajo manual para los administradores y desarrolladores de bases de datos.
- Esto hace que los eventos sean una característica útil para la administración y el mantenimiento de bases de datos en entornos de producción.



Eventos

- MySQL tiene un manejador de eventos que gestiona su programación y ejecución.
- El manejador se puede activar o desactivar, pero debería estar activado de forma predeterminada.
- Para confirmar que el manejador está activado, se puede ejecutar el siguiente comando:

```
show variables like 'event_scheduler';
```




Eventos



```
1 • show variables like 'event_scheduler';
```


<

Result Grid



Filter Rows:

Export:



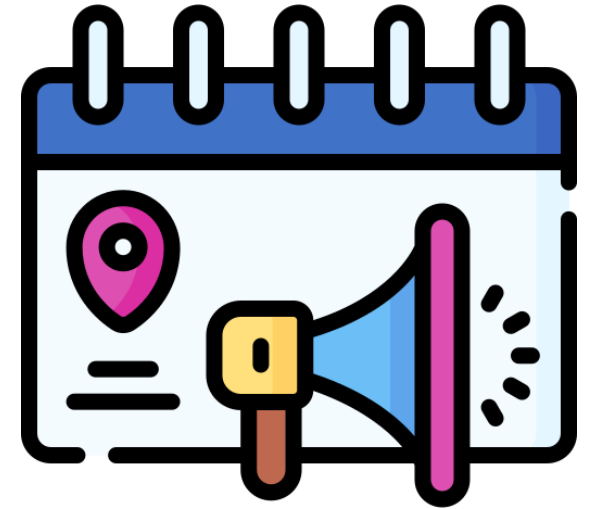
	Variable_name	Value
▶	event_scheduler	ON

Eventos

- Si el valor mostrado es OFF, el usuario (o el administrador de la base de datos) debe activar el manejador con el siguiente comando:

```
set global event_scheduler = on;
```

- Si el valor devuelto de la instrucción anterior es DISABLE, el manejador de eventos estará deshabilitado de manera predeterminada y será necesario cambiarlo en un archivo de configuración por el administrador de su base de datos.



Eventos

- Tomando en cuenta lo anterior, veamos un ejemplo: se creara un evento llamado `generate_report` sobre la base de datos `sakila`. Este evento estará programado para ejecutarse todos los días a medianoche (`STARTS CURRENT_DATE()`) y generara un informe sobre el número total de alquileres realizados en el último día:

```
USE sakila;
```

```
-- Crear la tabla informes si no existe
```

```
CREATE TABLE IF NOT EXISTS sakila.information (  
    date_creation DATE,  
    total_rental INT  
);
```



Eventos

- Tomando en cuenta lo anterior, veamos un ejemplo: se creara un evento llamado `generate_report` sobre la base de datos `sakila`. Este evento estará programado para ejecutarse todos los días a medianoche (`STARTS CURRENT_DATE()`) y generara un informe sobre el número total de alquileres realizados en el último día:

```
-- Habilitar el planificador de eventos si no está habilitado
```

```
SET GLOBAL event_scheduler = ON;
```

```
-- Crear el evento
```

```
delimiter //
```

```
CREATE EVENT IF NOT EXISTS generate_report
```

```
ON SCHEDULE EVERY 1 DAY STARTS CURRENT_DATE()
```

```
DO
```

```
BEGIN
```

```
-- Variables para almacenar el número de alquileres
```

```
SET @total_rentals := 0;
```

2

Eventos

- Tomando en cuenta lo anterior, veamos un ejemplo: se creara un evento llamado `generate_report` sobre la base de datos `sakila`. Este evento estará programado para ejecutarse todos los días a medianoche (`STARTS CURRENT_DATE()`) y generara un informe sobre el número total de alquileres realizados en el último día:

```
-- Obtener el número total de alquileres del último día
SELECT COUNT(*) INTO @total_rentals
FROM sakila.rental
WHERE rental_date >= DATE_SUB(CURDATE(), INTERVAL 1 DAY);
```

```
-- Insertar el informe en una tabla de informes
INSERT INTO sakila.information (date_creation, total_rental)
VALUES (CURDATE(), @total_rentals);
```

```
END //
```

```
DELIMITER ;
```

3

Eventos

- Para ver los eventos que se han generado en MySQL, se puede consultar la tabla del sistema llamada `information_schema.events`:

```
SELECT * FROM information_schema.events;
```

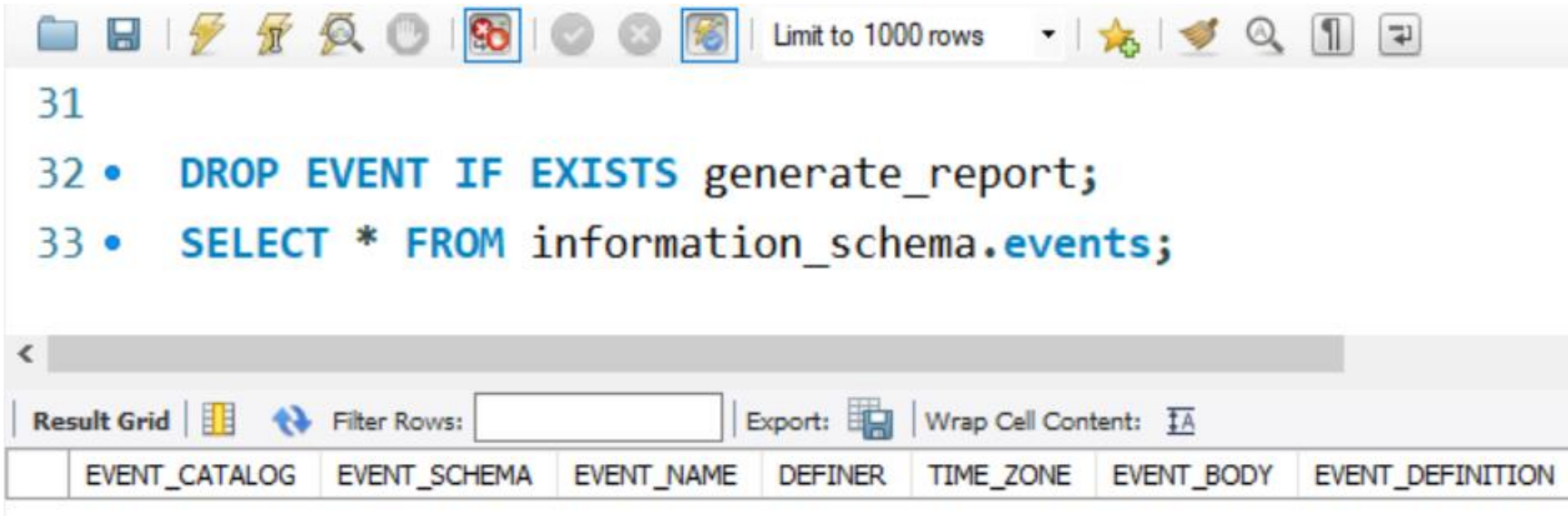
- La consulta anterior proporcionará una lista de todos los eventos programados en el servidor MySQL, incluyendo detalles como el nombre del evento, el nombre de la base de datos, la hora de inicio, la frecuencia de ejecución, entre otros.



Eventos

- Al igual que con las instrucciones revisadas previamente, se puede eliminar el evento al utilizar la siguiente instrucción en SQL:

```
DROP EVENT IF EXISTS generate_report;  
SELECT * FROM information_schema.events;
```



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and search. The main text area contains the following SQL commands:

```
31  
32 • DROP EVENT IF EXISTS generate_report;  
33 • SELECT * FROM information_schema.events;
```

Below the text area is a horizontal scrollbar. At the bottom, there is a toolbar with 'Result Grid', 'Filter Rows' (with a search input), 'Export', and 'Wrap Cell Content'. Below this toolbar is a table header with the following columns:

	EVENT_CATALOG	EVENT_SCHEMA	EVENT_NAME	DEFINER	TIME_ZONE	EVENT_BODY	EVENT_DEFINITION
--	---------------	--------------	------------	---------	-----------	------------	------------------

Eventos

En caso de tener que modificar un evento, se puede hacer de dos maneras:

- Eliminar el evento existente y crear un nuevo evento haciendo los cambios necesarios.
- Utilizar la siguiente instrucción:

```
delimiter //  
ALTER EVENT nombre_del_evento  
-- Cambia la programación según sea necesario  
ON SCHEDULE EVERY 1 DAY STARTS CURRENT_DATE()  
DO  
BEGIN  
    -- Cuerpo del evento actualizado...  
END;  
DELIMITER ;
```



Eventos

- En otro ejemplo, podemos generar un informe cada treinta minutos sobre el número total de alquileres realizados en la última hora en la base de datos sakila y guardar este informe en una tabla llamada hourly_reports:

```
USE sakila;
```

```
-- Crear la tabla de reporte
```

```
CREATE TABLE IF NOT EXISTS sakila.hourly_reports (  
    report_id INT AUTO_INCREMENT PRIMARY KEY,  
    report_date DATETIME,  
    total_rental INT  
);
```



Eventos

- En otro ejemplo, podemos generar un informe cada treinta minutos sobre el número total de alquileres realizados en la última hora en la base de datos sakila y guardar este informe en una tabla llamada hourly_reports:

```
-- Habilitar el planificador de eventos si no está habilitado
SET GLOBAL event_scheduler = ON;
```

```
-- Crear el evento
```

```
delimiter //
```

```
CREATE EVENT IF NOT EXISTS generate_hourly_report
ON SCHEDULE EVERY 30 MINUTE STARTS CURRENT_TIMESTAMP()
DO
BEGIN
```

```
-- Variables para almacenar el número de alquileres
SET @total_rentals := 0;
```

2

Eventos

- En otro ejemplo, podemos generar un informe cada treinta minutos sobre el número total de alquileres realizados en la última hora en la base de datos sakila y guardar este informe en una tabla llamada hourly_reports:

```
-- Obtener el número total de alquileres de la última hora
SELECT COUNT(*) INTO @total_rentals
FROM sakila.rental
WHERE rental_date >= DATE_SUB(NOW(), INTERVAL 1 HOUR);
```

3

```
-- Insertar el informe en la tabla de informes por hora
INSERT INTO hourly_reports (report_date, total_rental)
VALUES (NOW(), @total_rentals);
END //
DELIMITER ;
```

Eventos

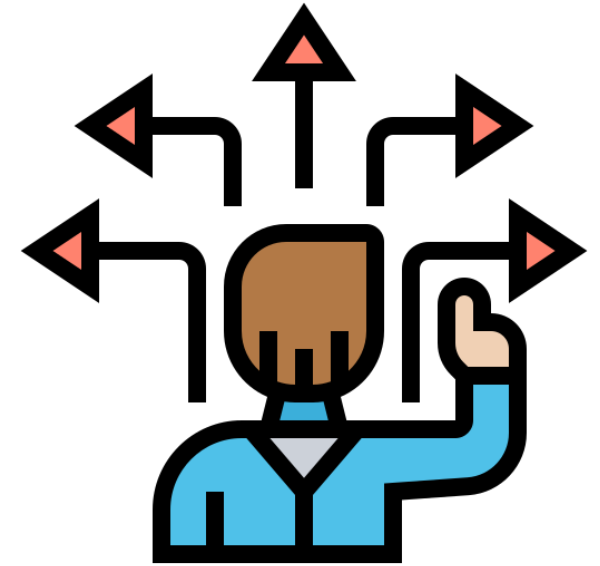
```
USE sakila;
-- Habilitar el planificador de eventos
SET GLOBAL event_scheduler = ON;

delimiter //
-- Crear el evento
CREATE EVENT IF NOT EXISTS `mark_inactive_inventory_event`
-- Se activa todos los días a medianoche
ON SCHEDULE EVERY 1 DAY STARTS CURRENT_DATE()
DO
BEGIN
    -- inactivo, los registros con cantidad disponible igual a cero
    UPDATE inventory
    SET active = 0
    WHERE quantity = 0;
END //
DELIMITER ;
```

Finalmente, en otro ejemplo crearemos un evento que se active cada día a la medianoche para realizar una tarea de mantenimiento en la tabla inventory de sakila. En este caso, la tarea será marcar como inactivo todos los registros en la tabla que tengan una cantidad disponible igual a cero.

Estructuras de decisión

- Las estructuras de decisión en MySQL son herramientas fundamentales para controlar el flujo de ejecución de una consulta o de un procedimiento almacenado basado en condiciones lógicas.
- Las estructuras de decisión permiten realizar acciones específicas según ciertas condiciones que se evalúan durante la ejecución de una consulta SQL.
- La estructura de decisión más común en MySQL es la sentencia IF. Esta sentencia permite ejecutar un bloque de código si una condición especificada es verdadera y otro bloque si es falsa.

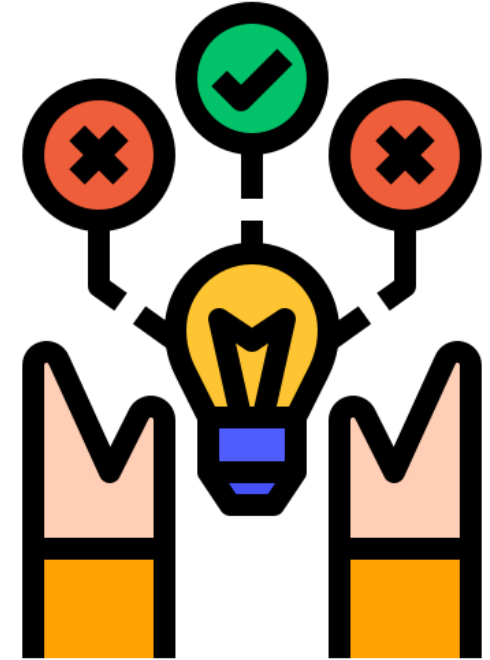


Estructuras de decisión

- Otra estructura de decisión importante es CASE. Esta instrucción permite evaluar múltiples condiciones y ejecutar un bloque de código diferente según la que se cumpla.
- Estas estructuras de decisión son esenciales en el desarrollo de aplicaciones y consultas complejas, ya que permiten adaptar el comportamiento del sistema según las condiciones específicas de los datos.
- Al utilizar estas herramientas de manera efectiva, los desarrolladores pueden crear consultas más sofisticadas y procedimientos almacenados que respondan dinámicamente a los diferentes escenarios que puedan surgir durante la ejecución del código SQL.

Estructuras de decisión

- MySQL proporciona una variedad de herramientas para realizar decisiones condicionales en consultas y procedimientos almacenados.
- Ya sea mediante el uso de estructuras de decisión básicas como IF y CASE, o combinando múltiples condiciones con operadores lógicos, MySQL ofrece la flexibilidad necesaria para implementar lógica de programación compleja en el contexto de una base de datos relacional.



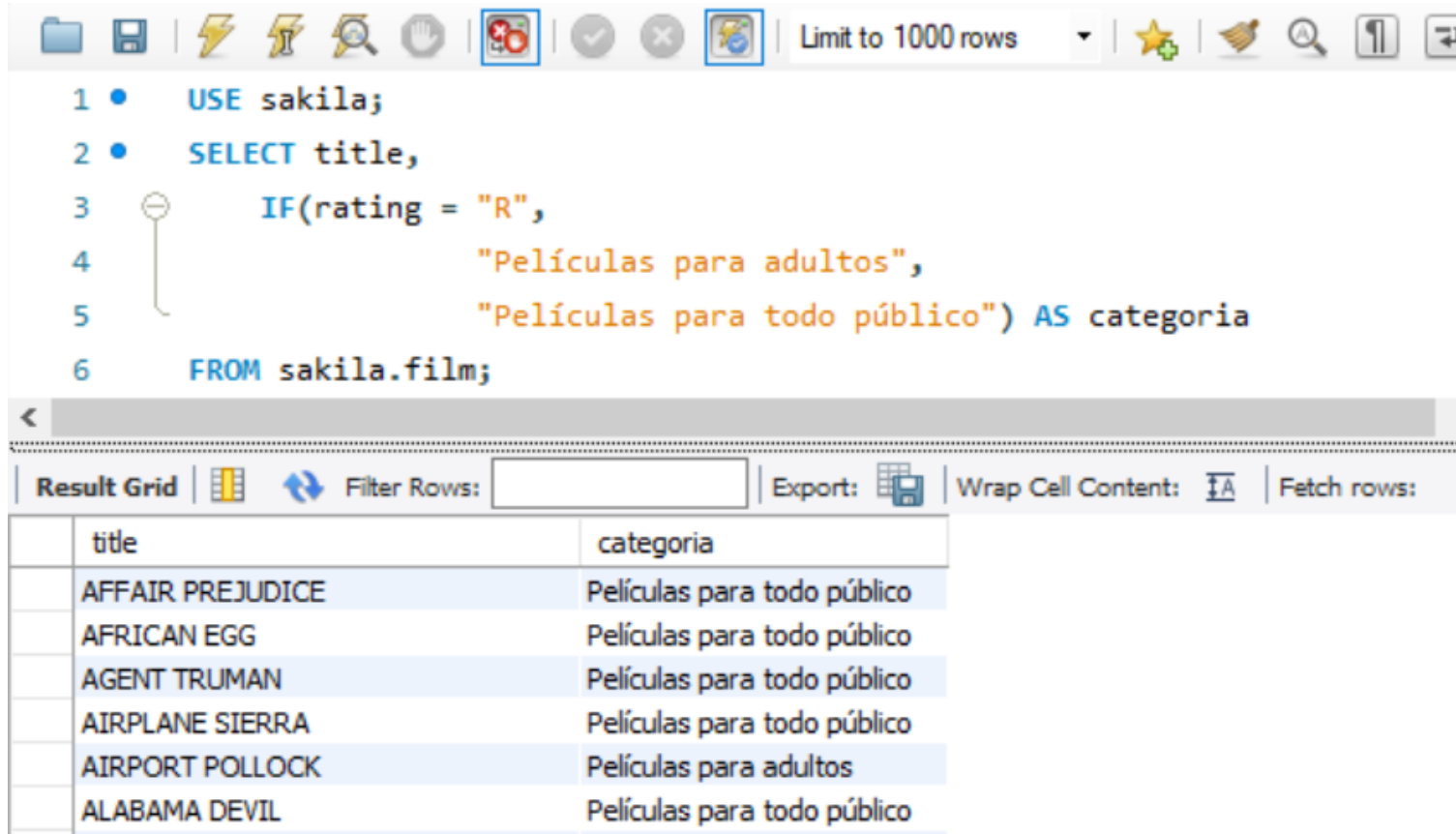
Estructuras de decisión

- Como ejemplo, supongamos que queremos seleccionar películas de la tabla film según su rating. Si el rating es igual a R, será deseable marcarlas como **Películas para adultos**, de lo contrario, las marcamos como **Películas para todo público**.

```
USE sakila;  
SELECT title,  
       IF(rating = "R",  
          "Películas para adultos",  
          "Películas para todo público") AS categoria  
FROM sakila.film;
```

Estructuras de decisión

- Como ejemplo, supongamos que queremos seleccionar películas de la tabla film según su rating. Si el rating es igual a R, será deseable marcarlas como **Películas para adultos**, de lo contrario, las marcamos como **Películas para todo público**.

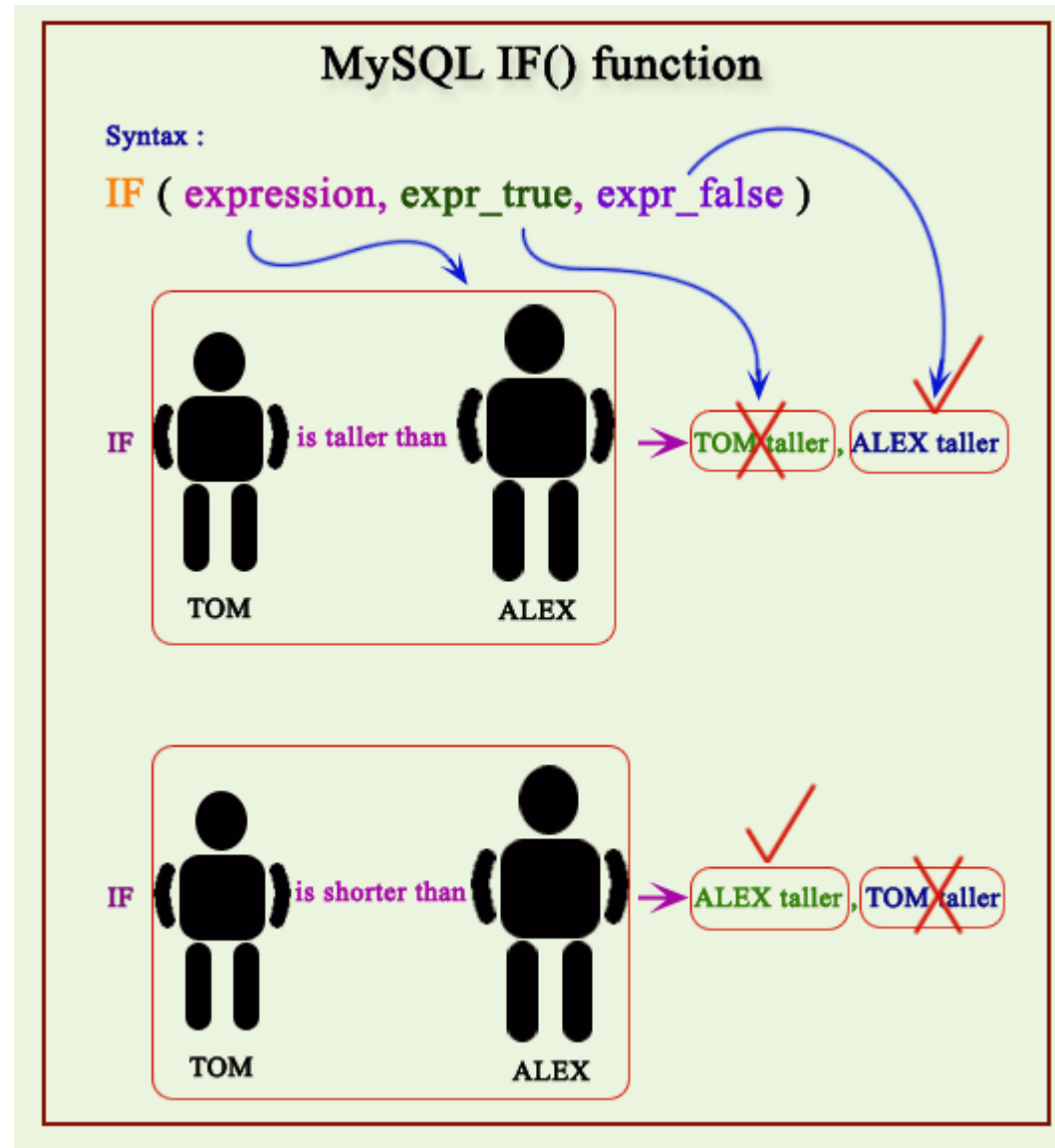


```
1 • USE sakila;
2 • SELECT title,
3     IF(rating = "R",
4         "Películas para adultos",
5         "Películas para todo público") AS categoria
6 FROM sakila.film;
```

Result Grid

title	categoria
AFFAIR PREJUDICE	Películas para todo público
AFRICAN EGG	Películas para todo público
AGENT TRUMAN	Películas para todo público
AIRPLANE SIERRA	Películas para todo público
AIRPORT POLLOCK	Películas para adultos
ALABAMA DEVIL	Películas para todo público

Estructuras de decisión



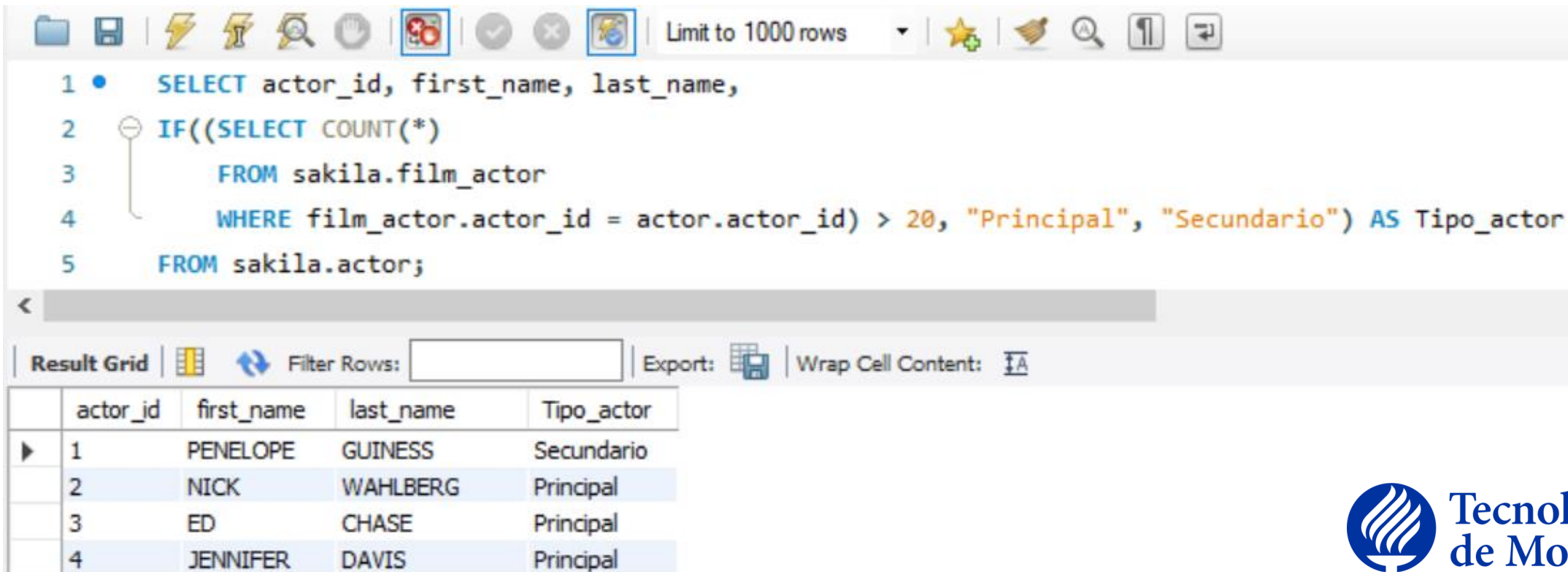
Estructuras de decisión

- En el siguiente ejemplo se seleccionan actores de la tabla actor y se determina si son actores principales o secundarios según el número de películas en las que han participado. Si un actor ha aparecido en más de veinte películas, lo consideramos un actor principal; de lo contrario, lo consideramos un actor secundario.

```
USE sakila;  
SELECT actor_id, first_name, last_name,  
IF((SELECT COUNT(*)  
    FROM sakila.film_actor  
    WHERE film_actor.actor_id = actor.actor_id) > 20, "Principal", "Secundario") AS Tipo_actor  
FROM sakila.actor;
```

Estructuras de decisión

- En el siguiente ejemplo se seleccionan actores de la tabla actor y se determina si son actores principales o secundarios según el número de películas en las que han participado. Si un actor ha aparecido en más de veinte películas, lo consideramos un actor principal; de lo contrario, lo consideramos un actor secundario.



```
1 • SELECT actor_id, first_name, last_name,  
2   IF((SELECT COUNT(*)  
3     FROM sakila.film_actor  
4     WHERE film_actor.actor_id = actor.actor_id) > 20, "Principal", "Secundario") AS Tipo_actor  
5 FROM sakila.actor;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	actor_id	first_name	last_name	Tipo_actor
▶	1	PENELOPE	GUINNESS	Secundario
	2	NICK	WAHLBERG	Principal
	3	ED	CHASE	Principal
	4	JENNIFER	DAVIS	Principal

Estructuras de decisión

- Para el siguiente ejemplo, usamos la tabla payment para calcular el nivel de membresía del cliente con el customer_id uno, basado en la cantidad total gastada por el. Si la suma de los pagos supera cien, asumimos que el cliente es miembro de oro (gold), de lo contrario, es un cliente regular.

```
USE sakila;
-- Declaración de variables
SET @precio_alquiler := 4.99; -- Precio de alquiler
SET @customer_id := 1; -- ID del cliente

-- Determinar el nivel de membresía del cliente basado en sus pagos
SELECT IF(SUM(amount) > 100, 'gold', 'regular') AS nivel_membresia
INTO @nivel_membresia
FROM sakila.payment
WHERE customer_id = @customer_id;
```



Estructuras de decisión

- Para el siguiente ejemplo, usamos la tabla payment para calcular el nivel de membresía del cliente con el customer_id uno, basado en la cantidad total gastada por el. Si la suma de los pagos supera cien, asumimos que el cliente es miembro de oro (gold), de lo contrario, es un cliente regular.

```
-- Aplicar descuento si el cliente es miembro gold
```

```
SET @descuento = IF(@nivel_membresia = 'gold', 0.1, 0);
```

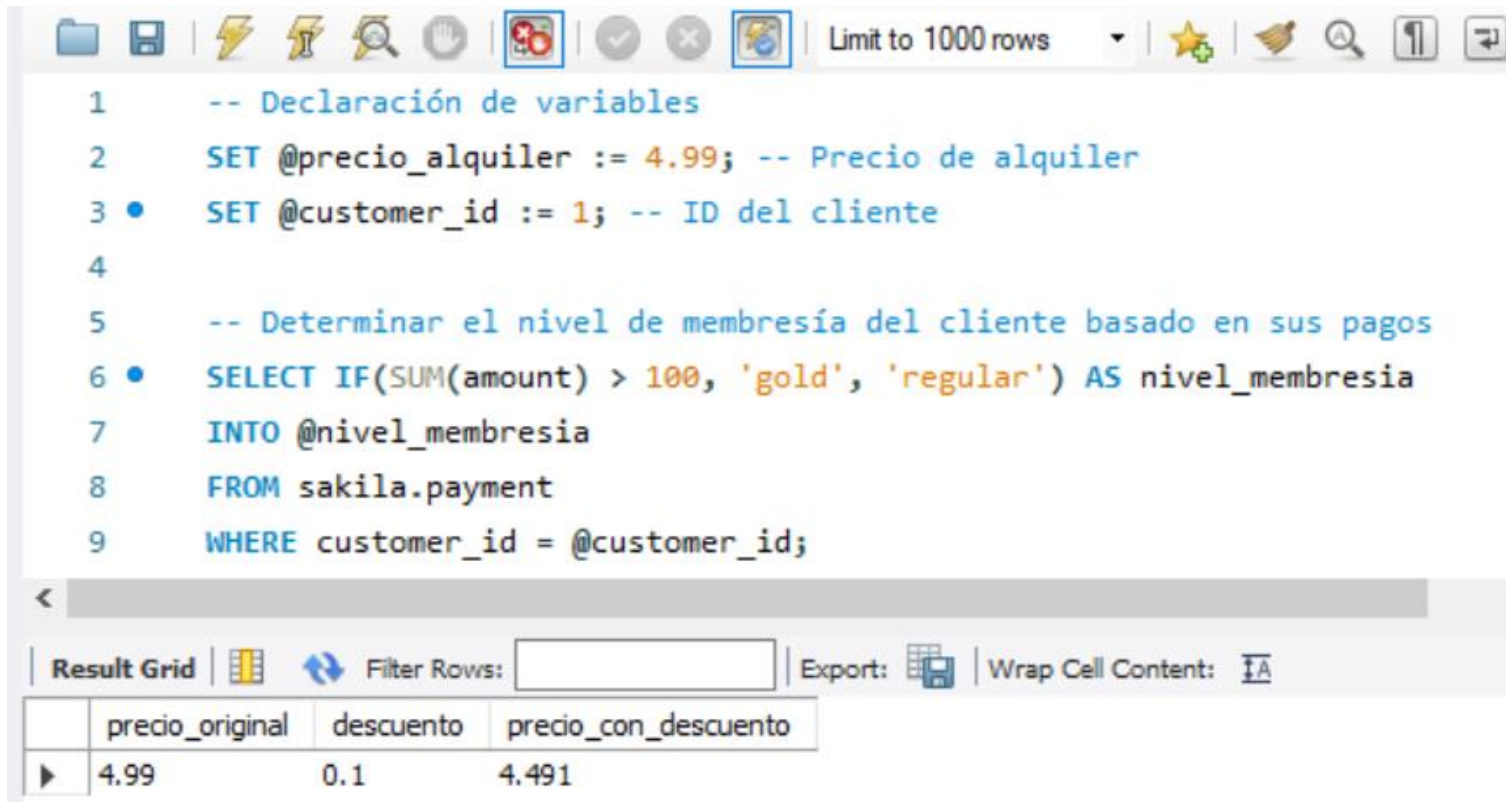
2

```
-- Calcular el precio total con o sin descuento
```

```
SELECT @precio_alquiler AS precio_original,@descuento AS descuento,  
       round(@precio_alquiler * (1 - @descuento),3) AS precio_con_descuento;
```


Estructuras de decisión

- Para el siguiente ejemplo, usamos la tabla payment para calcular el nivel de membresía del cliente con el customer_id uno, basado en la cantidad total gastada por el. Si la suma de los pagos supera cien, asumimos que el cliente es miembro de oro (gold), de lo contrario, es un cliente regular.



```
1  -- Declaración de variables
2  SET @precio_alquiler := 4.99; -- Precio de alquiler
3  • SET @customer_id := 1; -- ID del cliente
4
5  -- Determinar el nivel de membresía del cliente basado en sus pagos
6  • SELECT IF(SUM(amount) > 100, 'gold', 'regular') AS nivel_membresia
7  INTO @nivel_membresia
8  FROM sakila.payment
9  WHERE customer_id = @customer_id;
```

Result Grid

	precio_original	descuento	precio_con_descuento
▶	4.99	0.1	4.491

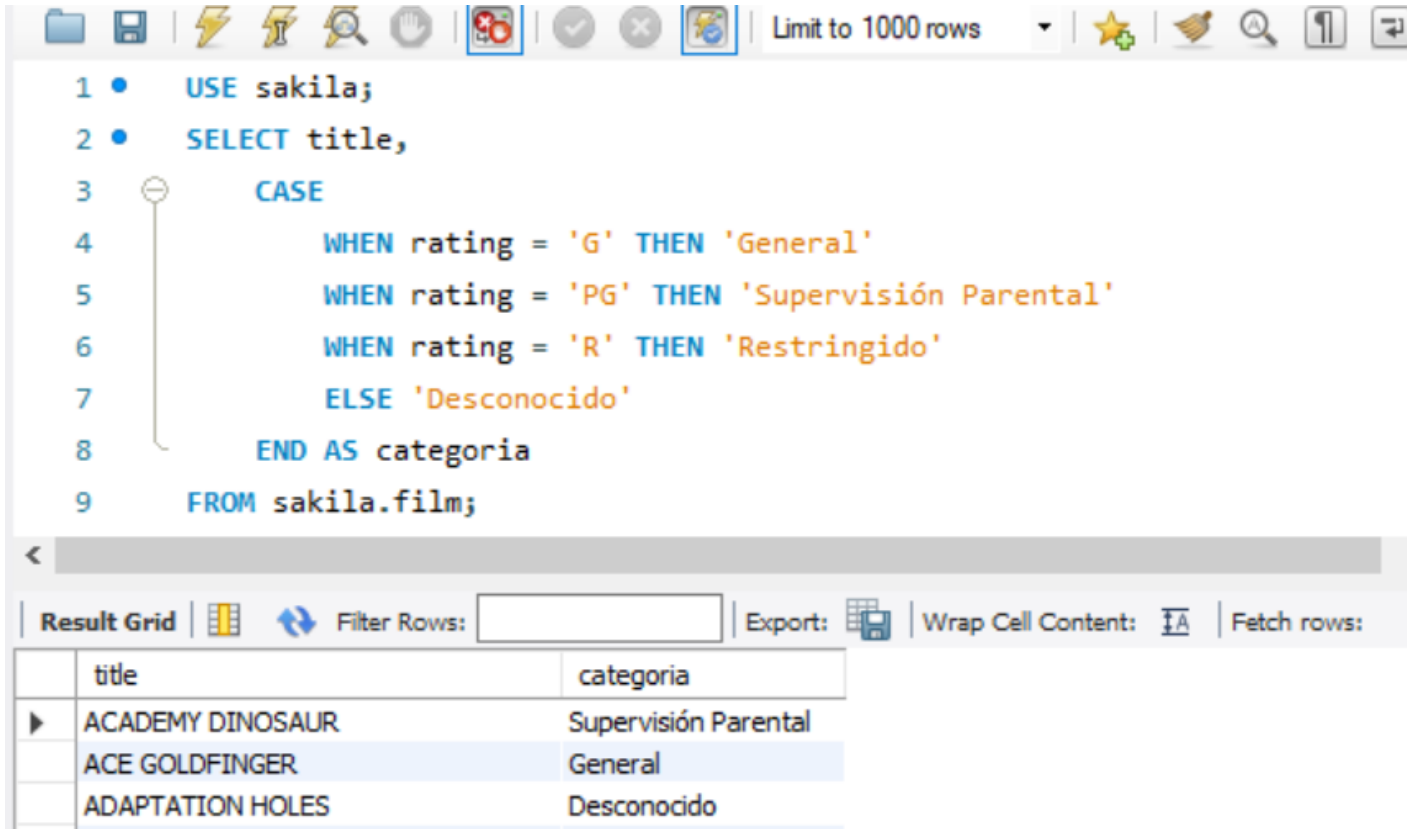
Estructuras de decisión

- En el siguiente ejemplo, vamos a seleccionar información de la tabla film y clasificar las películas en tres categorías según su rating (rating): G para General, PG para Supervisión Parental y R para Restringido.

```
USE sakila;  
SELECT title,  
       CASE  
         WHEN rating = 'G' THEN 'General'  
         WHEN rating = 'PG' THEN 'Supervisión Parental'  
         WHEN rating = 'R' THEN 'Restringido'  
         ELSE 'Desconocido'  
       END AS categoria  
FROM sakila.film;
```

Estructuras de decisión

- En el siguiente ejemplo, vamos a seleccionar información de la tabla film y clasificar las películas en tres categorías según su rating (rating): G para General, PG para Supervisión Parental y R para Restringido.



The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
1 • USE sakila;
2 • SELECT title,
3     CASE
4         WHEN rating = 'G' THEN 'General'
5         WHEN rating = 'PG' THEN 'Supervisión Parental'
6         WHEN rating = 'R' THEN 'Restringido'
7         ELSE 'Desconocido'
8     END AS categoria
9 FROM sakila.film;
```

Below the query editor, the 'Result Grid' tab is active, displaying the results of the query. The table has two columns: 'title' and 'categoria'. The results are as follows:

title	categoria
ACADEMY DINOSAUR	Supervisión Parental
ACE GOLDFINGER	General
ADAPTATION HOLES	Desconocido

Estructuras de decisión

- Para el siguiente ejemplo, utilizamos variables para determinar el idioma predominante de las películas en función del número de películas disponibles de las tablas Language y Film.

```
USE sakila;
-- Declaración de variables
SET @idioma_predominante := "Inglés";

-- Determinar el idioma predominante
SELECT
    CASE
        WHEN COUNT(*) > 100 THEN 'Inglés'
        WHEN COUNT(*) > 50 THEN 'Español'
        WHEN COUNT(*) > 20 THEN 'Francés'
        ELSE 'Otro'
    END AS idioma
INTO @idioma_predominante FROM sakila.language
INNER JOIN film ON language.language_id = film.language_id
GROUP BY language.name;
```

1

Estructuras de decisión

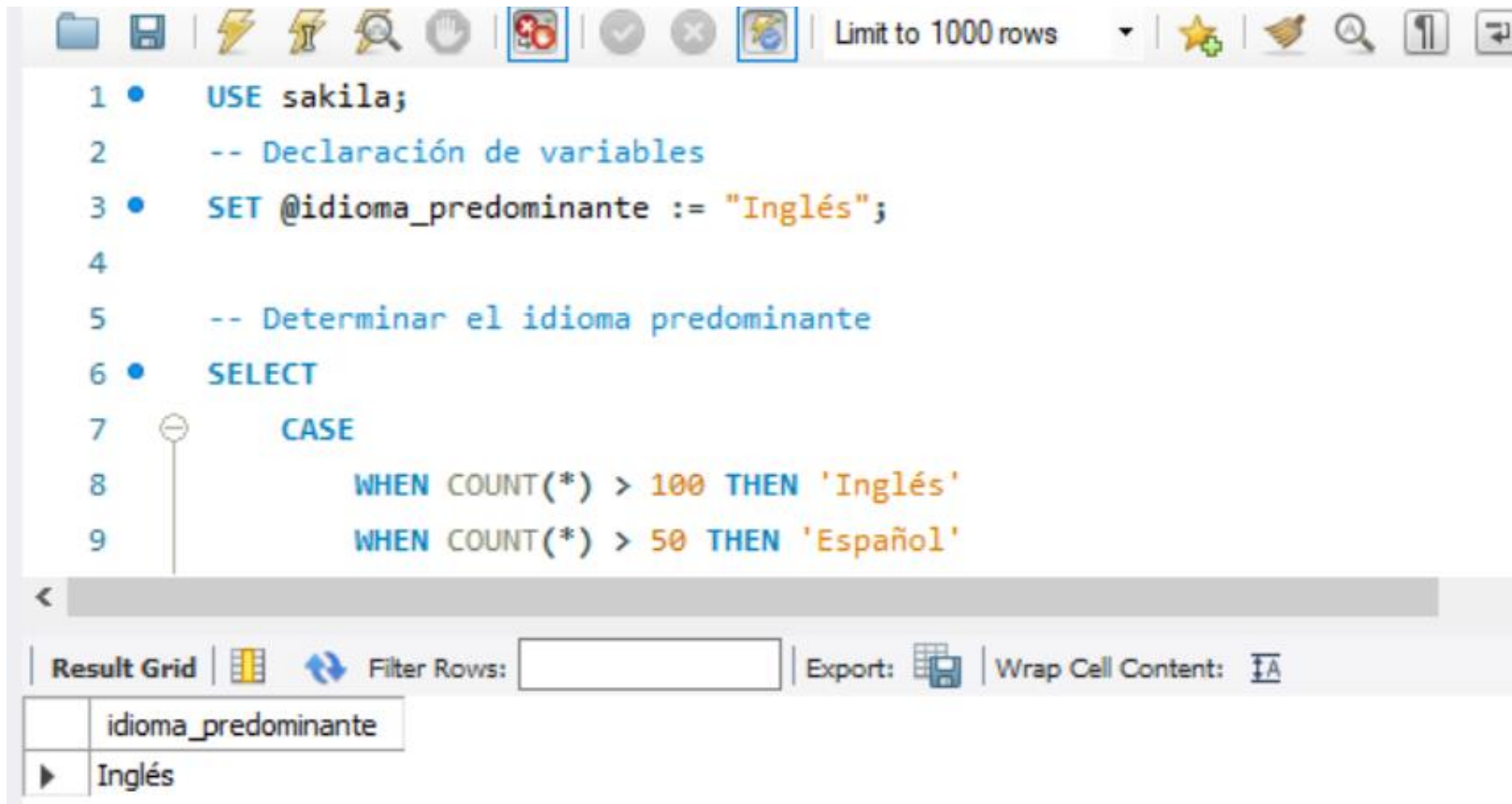
```
USE sakila;
-- Declaración de variables
SET @idioma_predominante := "Inglés";

-- Determinar el idioma predominante
SELECT
    CASE
        WHEN COUNT(*) > 100 THEN 'Inglés'
        WHEN COUNT(*) > 50 THEN 'Español'
        WHEN COUNT(*) > 20 THEN 'Francés'
        ELSE 'Otro'
    END AS idioma
INTO @idioma_predominante
FROM sakila.language
INNER JOIN film ON language.language_id = film.language_id
GROUP BY language.name;
-- Mostrar el idioma predominante
SELECT @idioma_predominante AS idioma_predominante
```

Para el siguiente ejemplo, utilizamos variables para determinar el idioma predominante de las películas en función del número de películas disponibles de las tablas language y film.

Estructuras de decisión

- Para el siguiente ejemplo, utilizamos variables para determinar el idioma predominante de las películas en función del número de películas disponibles de las tablas language y film.



```
1 • USE sakila;
2   -- Declaración de variables
3 • SET @idioma_predominante := "Inglés";
4
5   -- Determinar el idioma predominante
6 • SELECT
7     CASE
8       WHEN COUNT(*) > 100 THEN 'Inglés'
9       WHEN COUNT(*) > 50 THEN 'Español'
```

Result Grid

idioma_predominante
Inglés

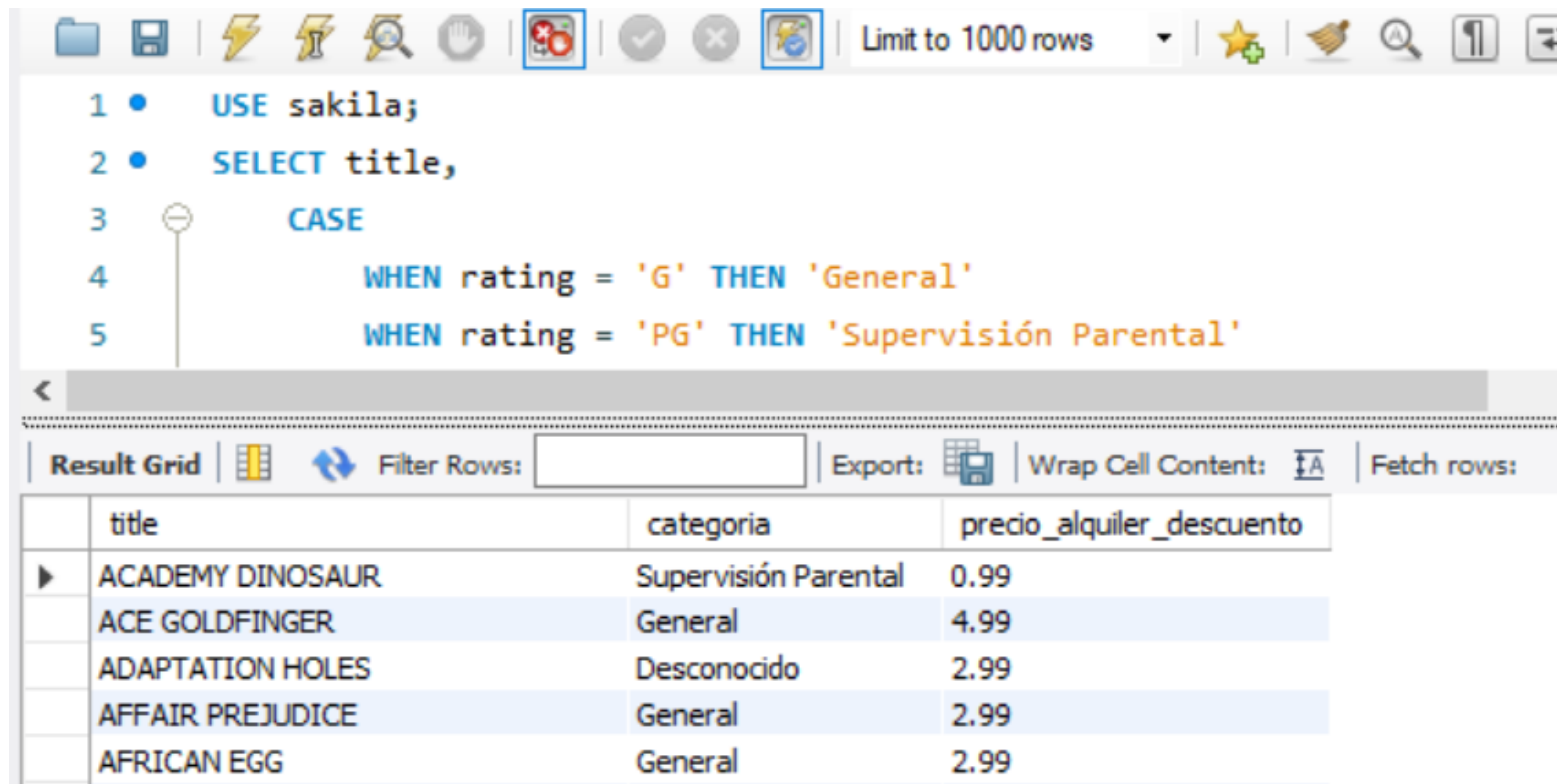
Estructuras de decisión

- Para el ultimo ejemplo, seleccionamos información de la tabla film y clasificamos las películas en tres categorías según su rating (rating): G, PG y R. Además, aplicaremos un descuento del 10% en el precio de alquiler para las películas clasificadas como R.

```
USE sakila;  
SELECT title,  
       CASE  
         WHEN rating = 'G' THEN 'General'  
         WHEN rating = 'PG' THEN 'Supervisión Parental'  
         WHEN rating = 'R' THEN 'Restringido'  
         ELSE 'Desconocido'  
       END AS categoria,  
       IF(rating = 'R', rental_rate * 0.9, rental_rate) AS precio_alquiler_descuento  
FROM  
  sakila.film;
```

Estructuras de decisión

- Para el ultimo ejemplo, seleccionamos información de la tabla film y clasificamos las películas en tres categorías según su rating (rating): G, PG y R. Además, aplicaremos un descuento del 10% en el precio de alquiler para las películas clasificadas como R.



The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

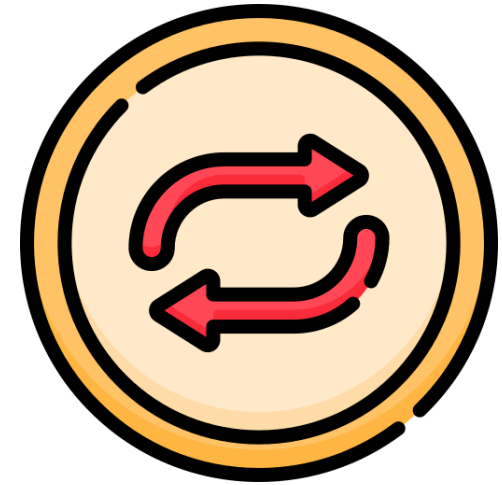
```
1 • USE sakila;
2 • SELECT title,
3 • CASE
4 •     WHEN rating = 'G' THEN 'General'
5 •     WHEN rating = 'PG' THEN 'Supervisión Parental'
```

Below the query editor, there is a 'Result Grid' section with a toolbar for filtering, exporting, and wrapping cell content. The results are displayed in a table with the following data:

	title	categoria	precio_alquiler_descuento
▶	ACADEMY DINOSAUR	Supervisión Parental	0.99
	ACE GOLDFINGER	General	4.99
	ADAPTATION HOLES	Desconocido	2.99
	AFFAIR PREJUDICE	General	2.99
	AFRICAN EGG	General	2.99

Estructuras de repetición

- En MySQL, a diferencia de muchos otros lenguajes de programación, no se cuenta con estructuras de repetición típicas como for, while, o do-while que permiten iterar sobre un conjunto de instrucciones un número determinado de veces o mientras se cumple una condición específica.
- En cambio, MySQL se centra en consultas y manipulación de datos en bases de datos, y su principal mecanismo para la iteración es el uso de cursores dentro de procedimientos almacenados.



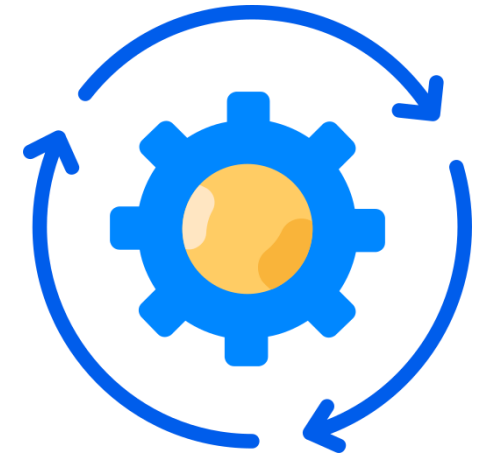
Estructuras de repetición

- Los cursores en MySQL son objetos que permiten recorrer secuencialmente los resultados de una consulta, como si fueran filas en una tabla virtual.
- Estos cursores se pueden utilizar dentro de procedimientos almacenados y funciones para procesar cada fila resultante de una consulta una a una.
- Aunque los cursores ofrecen una forma de repetición en MySQL, es importante tener en cuenta que su uso debe ser cuidadoso, ya que el procesamiento de filas puede tener un impacto significativo en el rendimiento.



Estructuras de repetición

- En lugar de utilizar bucles y estructuras de repetición directamente en consultas SQL en MySQL, es común que los desarrolladores utilicen consultas SQL más complejas y funciones agregadas para lograr el comportamiento deseado.
- Por ejemplo, mediante el uso de expresiones CASE y funciones agregadas como SUM, COUNT, y GROUP BY, se pueden realizar cálculos y operaciones de resumen de datos avanzadas sin la necesidad de bucles explícitos.



Estructuras de repetición

- Veamos un ejemplo de un procedimiento almacenado, el cual itera sobre los nombres de los actores en la tabla actor de la base de datos sakila:

```
USE sakila;
-- Crear el procedimiento almacenado
DELIMITER //

CREATE PROCEDURE iterate_actors1()
BEGIN
    DECLARE actor_name VARCHAR(45);
    DECLARE done BOOLEAN DEFAULT FALSE;
    -- Declarar un cursor para obtener los nombres de los actores
    DECLARE actor_cursor CURSOR FOR
        SELECT first_name
        FROM actor;
```

1

Estructuras de repetición

- Veamos un ejemplo de un procedimiento almacenado, el cual itera sobre los nombres de los actores en la tabla actor de la base de datos sakila:

```
-- Indicar que no hay más filas para procesar
DECLARE CONTINUE HANDLER FOR NOT FOUND
    SET done = TRUE;
-- Abrir el cursor
OPEN actor_cursor;
loop_start: LOOP -- Iniciar el bucle
    -- Obtener el siguiente nombre de actor del cursor
    FETCH actor_cursor INTO actor_name;
    -- Si no hay más nombres de actores, salir del bucle
    IF done THEN
        LEAVE loop_start;
    END IF;
```

2

Estructuras de repetición

- Veamos un ejemplo de un procedimiento almacenado, el cual itera sobre los nombres de los actores en la tabla actor de la base de datos sakila:

```
-- Mostrar el nombre del actor
    SELECT actor_name AS ActorName;
END LOOP loop_start;
-- Cerrar el cursor
CLOSE actor_cursor;
END;
//
-- Restaurar el delimitador
DELIMITER ;

CALL iterate_actors1();
```

3

Estructuras de repetición

- Veamos un ejemplo de un procedimiento almacenado, el cual itera sobre los nombres de los actores en la tabla actor de la base de datos sakila:

The screenshot shows the MySQL Workbench interface. In the background, a SQL editor contains the following code:

```
30 END;  
31 //  
32 -- Restaurar el deli  
33 DELIMITER ;  
34  
35 • CALL iterate_actors1
```

A warning dialog box is overlaid on the editor. The dialog has a yellow warning icon and the title "MySQL Workbench". The main text reads:

Maximum result count reached
No further result tabs will be displayed as the maximum number has been reached.
You may stop the operation, leaving the connection out of sync. You'll have to go to 'Query->Reconnect to server' menu item to reset the state.

Below the text, it asks: "Do you want to cancel the operation?". There are two buttons: "→ Yes" and "→ No".

In the bottom left, a "Result Grid" is visible with the following data:

ActorName
NATALIE

Transacciones

- Una transacción en MySQL es una serie de operaciones SQL que se realizan como una sola unidad lógica de trabajo.
- Lo anterior significa que todas las operaciones dentro de la transacción se ejecutan completamente o ninguna se ejecuta en absoluto.
- El procesamiento en bloque asegura que si una parte de la transacción falla, se puede revertir o deshacer todas las operaciones realizadas hasta ese punto para mantener la coherencia de los datos.



Transacciones

- MySQL utiliza el enfoque ACID (**Atomicidad**, **Consistencia**, **Aislamiento**, **Durabilidad**) para garantizar la fiabilidad de las transacciones.
- La **atomicidad** asegura que todas las operaciones dentro de una transacción se completen con éxito o se deshagan por completo si ocurre un error.
- La **consistencia** garantiza que la base de datos pase de un estado válido a otro estado válido después de que se complete una transacción.



Transacciones

- El **aislamiento** asegura que las transacciones concurrentes no interfieran entre sí.
- la **durabilidad** garantiza que una vez que se confirma una transacción, los cambios se guarden permanentemente incluso en caso de fallo del sistema.
- Las transacciones son importantes para mantener la integridad de los datos en una base de datos, especialmente en entornos donde múltiples usuarios pueden acceder y modificar los mismos datos al mismo tiempo.



Transacciones

- Las transacciones son útiles cuando se utiliza el **lenguaje de manipulación de datos** de MySQL (**DML; Data Manipulation Language**), como las instrucciones para **insertar**, **actualizar** o **eliminar**.
- Para el **lenguaje de definición de datos** (**DDL; Data Definition Language**) las instrucciones para **crear funciones**, **eliminar procedimiento almacenados** o **modificar tablas** no deben realizarse en una transacción ya que no se pueden revertir; al ejecutarlas se confirma automáticamente una transacción.



Transacciones

-- Ver la información de la tabla clientes

```
SELECT * FROM sakila.customer;
```

1

-- Iniciar la transacción

```
START TRANSACTION;
```

-- Actualizar emails de clientes

```
UPDATE sakila.customer SET email = "unknown@gmail.com"  
WHERE customer_id >= 1 AND customer_id <= 8;
```

-- Ver la información de la tabla clientes

```
SELECT * FROM sakila.customer;
```

-- **Deshacer la transacción si algo falla**
ROLLBACK;

-- Ver la información de la tabla clientes

```
SELECT * FROM sakila.customer;
```

3

2



Supongamos que queremos realizar una transacción que actualice el correo electrónico de varios clientes de la base de datos Sakila.

Transacciones

-- Ver la información de la tabla clientes

```
SELECT * FROM sakila.customer;
```

1

-- Iniciar la transacción

```
START TRANSACTION;
```

-- Actualizar emails de clientes

```
UPDATE sakila.customer SET email = "unknown@gmail.com"  
WHERE customer_id >= 1 AND customer_id <= 8;
```

2

-- Ver la información de la tabla clientes

```
SELECT * FROM sakila.customer;
```

-- **Confirmar la transacción**
COMMIT;

-- Ver la información de la tabla clientes

```
SELECT * FROM sakila.customer;
```

3



Supongamos que queremos realizar una transacción que actualice el correo electrónico de varios clientes de la base de datos Sakila.

Transacciones

- Supongamos que queremos realizar una transacción que actualice el correo electrónico de varios clientes de la base de datos Sakila:

-- Ver la información de la tabla clientes

```
SELECT * FROM sakila.customer;
```

	customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
►	1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org	5	1	2006-02-14 22:04:36	2006-02-15 04:57:20
	2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org	6	1	2006-02-14 22:04:36	2006-02-15 04:57:20
	3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org	7	1	2006-02-14 22:04:36	2006-02-15 04:57:20
	4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org	8	1	2006-02-14 22:04:36	2006-02-15 04:57:20
	5	1	ELIZABETH	BROWN	ELIZABETH.BROWN@sakilacustomer.org	9	1	2006-02-14 22:04:36	2006-02-15 04:57:20



Transacciones

```
4  -- Iniciar la transacción
5  •  START TRANSACTION;
6
7  -- Actualizar el email del cliente
8  •  UPDATE sakila.customer
9     SET email = "unknown@gmail.com"
10    WHERE customer_id >= 1 AND customer_id <= 8;
11
12 •  SELECT * FROM sakila.customer;
13
```

Supongamos que queremos realizar una transacción que actualice el correo electrónico de varios clientes de la base de datos Sakila.









<									
Result Grid Filter Rows: Edit: Export/Import: Wrap Cell Content:									
	customer_id	store_id	first_name	last_name	email	address_id	active	create_date	last_update
▶	1	1	MARY	SMITH	unknown@gmail.com	5	1	2006-02-14 22:04:36	2024-03-02 16:26:12
	2	1	PATRICIA	JOHNSON	unknown@gmail.com	6	1	2006-02-14 22:04:36	2024-03-02 16:26:12
	3	1	LINDA	WILLIAMS	unknown@gmail.com	7	1	2006-02-14 22:04:36	2024-03-02 16:26:12
	4	2	BARBARA	JONES	unknown@gmail.com	8	1	2006-02-14 22:04:36	2024-03-02 16:26:12
	5	1	ELIZABETH	BROWN	unknown@gmail.com	9	1	2006-02-14 22:04:36	2024-03-02 16:26:12
	6	2	JENNIFER	DAVIS	unknown@gmail.com	10	1	2006-02-14 22:04:36	2024-03-02 16:26:12



Transacciones

```
15
16     -- Deshacer la transacción si algo falla
17 •   ROLLBACK;
18
19     -- Ver la información de la tabla clientes
20 •   SELECT * FROM sakila.customer;
21
```

Supongamos que queremos realizar una transacción que actualice el correo electrónico de varios clientes de la base de datos Sakila.

Result Grid			Filter Rows:	<input type="text"/>	Edit:				Export/Import:			Wrap Cell Content:	
	customer_id	store_id	first_name	last_name	email		address_id	active	create_date		last_update		
▶	1	1	MARY	SMITH	MARY.SMITH@sakilacustomer.org		5	1	2006-02-14 22:04:36		2006-02-15 04:57:20		
	2	1	PATRICIA	JOHNSON	PATRICIA.JOHNSON@sakilacustomer.org		6	1	2006-02-14 22:04:36		2006-02-15 04:57:20		
	3	1	LINDA	WILLIAMS	LINDA.WILLIAMS@sakilacustomer.org		7	1	2006-02-14 22:04:36		2006-02-15 04:57:20		
	4	2	BARBARA	JONES	BARBARA.JONES@sakilacustomer.org		8	1	2006-02-14 22:04:36		2006-02-15 04:57:20		



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>