Ivana Sánchez Pérez

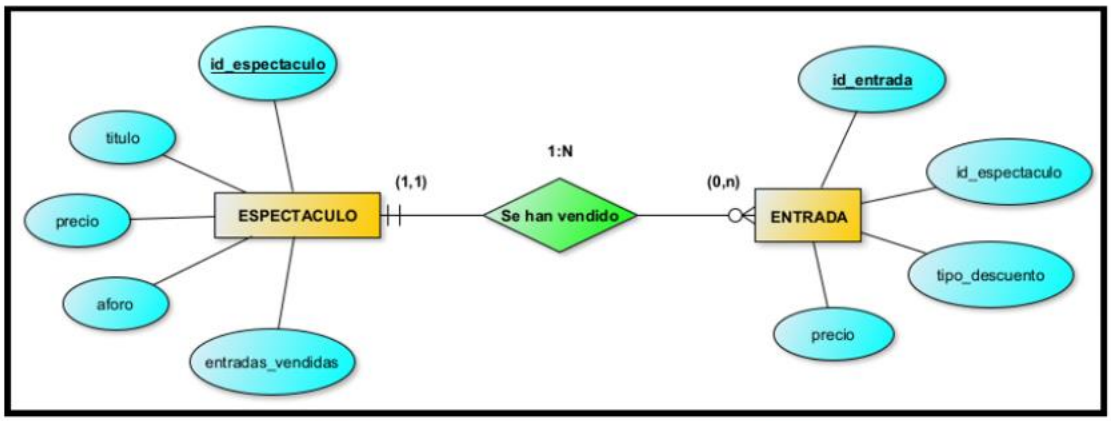
2º ASIR

Contenido

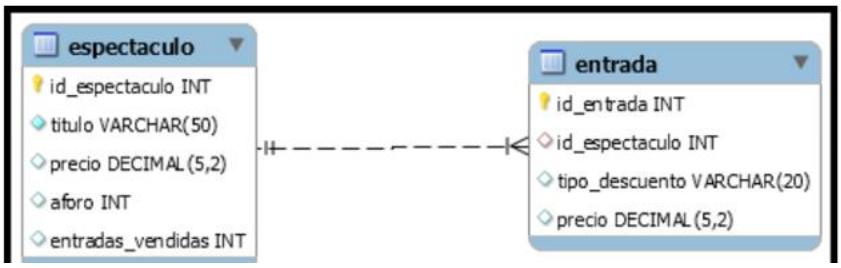
íNDICE

1.- Introducción

En el presente trabajo se muestra el ejercicio 3 de la asignatura ASGB donde se describe un caso práctico en el que la empresa BK Programación se plantean cómo automatizar la administración sobre el gestor de bases de datos de un proyecto de venta de entradas online según el diagrama-Entidad Relación y modelo relacional siguiente:



Sebastián López. *Diagrama E/R sistema de venta de entradas online* (CC BY)



2.- Creación de un trigger

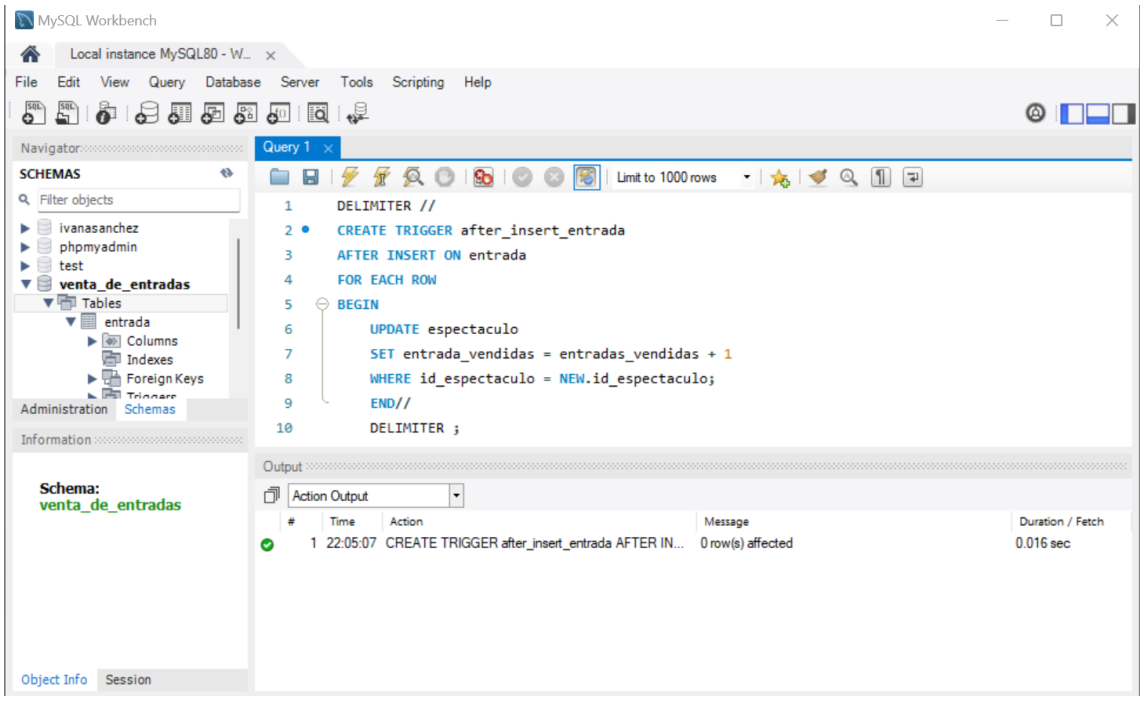
El trigger que vamos a crear, actualizará la columna de entradas_vendidas en la tabla espectáculo cada vez que se inserte una nueva entrada en la tabla entrada, incrementando el contador correspondiente al espectáculo al que pertenece la nueva entrada.

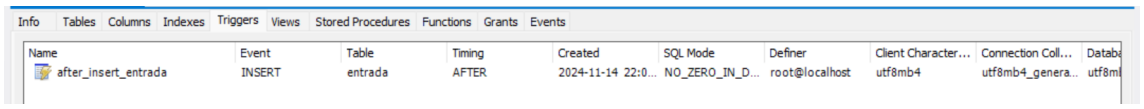
Comenzamos descargando el archivo .txt que se nos proporciona en el enunciado de la tarea y, cambiándole la extensión a sólo .sql. entramos en el terminal de MYSQL y lo importamos para crear nuestra base de datos. Hacemos Commit; para guardar.

```
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> DROP DATABASE venta_de_entradas;
Query OK, 2 rows affected (0.21 sec)

mysql> CREATE DATABASE venta_de_entradas;
Query OK, 1 row affected (0.00 sec)

mysql> USE venta_de_entradas;
Database changed
mysql>
mysql> CREATE TABLE espectaculo (
    ->   id_espectaculo INTEGER AUTO_INCREMENT,
    ->   titulo VARCHAR(50) NOT NULL,
    ->   precio DECIMAL(5,2),
    ->   aforo INTEGER,
    ->   entradas_vendidas INTEGER,
    ->   PRIMARY KEY (id_espectaculo)
    -> )ENGINE=InnoDB;
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> CREATE TABLE entrada (
    ->   id_entrada INTEGER AUTO_INCREMENT,
    ->   id_espectaculo INTEGER,
    ->   tipo_descuento VARCHAR(20),
    ->   precio DECIMAL(5,2),
    ->   PRIMARY KEY (id_entrada)
    -> )ENGINE=InnoDB;
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> ALTER TABLE entrada
    -> ADD CONSTRAINT entrada_FK_espectaculo
    -> FOREIGN KEY (id_espectaculo) REFERENCES
    -> espectaculo(id_espectaculo)
    -> ON DELETE RESTRICT
    -> ON UPDATE CASCADE;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO espectaculo (id_espectaculo,titulo,precio,aforo,entradas_vendidas) VALUES
    ->                          ('001','Mamma Tuya',25,200,0),
    ->                          ('002','El Rey Pelón',40,250,0),
    ->                          ('003','Romea y Julieto',20,10,7);
Query OK, 3 rows affected (0.01 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

```
mysql> SHOW TABLES;
+-----------------------------+
| Tables_in_venta_de_entradas |
+-----------------------------+
| entrada                     |
| espectaculo                 |
+-----------------------------+
2 rows in set (0.01 sec)

mysql>
```

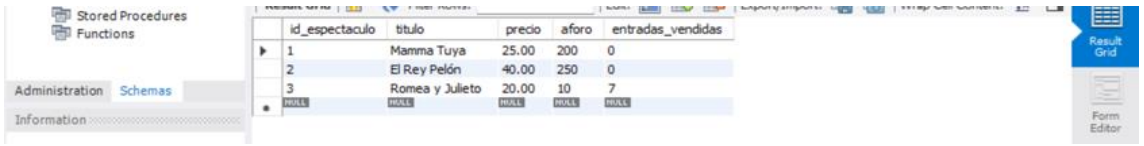
```
mysql> SELECT * FROM espectaculo;
+----------------+-----------------+--------+-------+-------------------+
| id_espectaculo | titulo          | precio | aforo | entradas_vendidas |
+----------------+-----------------+--------+-------+-------------------+
|              1 | Mamma Tuya      |  25.00 |   200 |                 0 |
|              2 | El Rey Pelón    |  40.00 |   250 |                 0 |
|              3 | Romea y Julieto |  20.00 |    10 |                 7 |
+----------------+-----------------+--------+-------+-------------------+
3 rows in set (0.01 sec)

mysql>
```



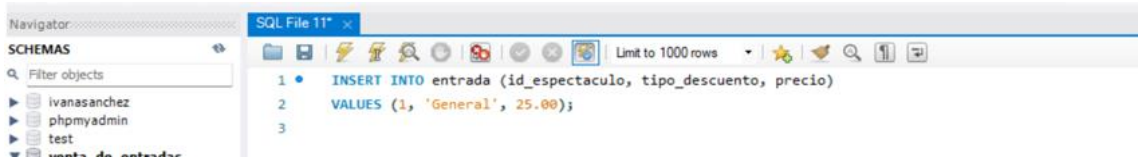


2.1.- COMPROBACIONES

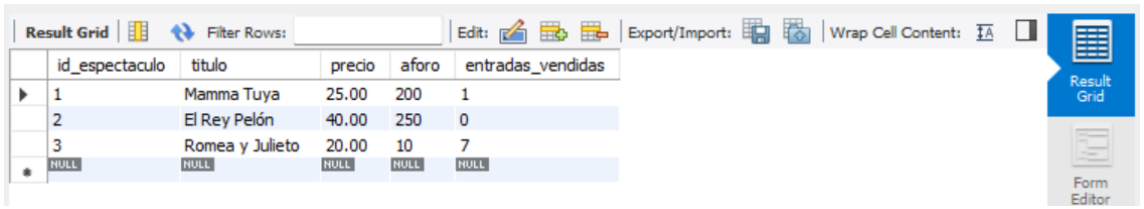
a) SELECT DE LA TABLA ESPECTÁCULO



b) INSERT DE LA TABLA ENTRADA

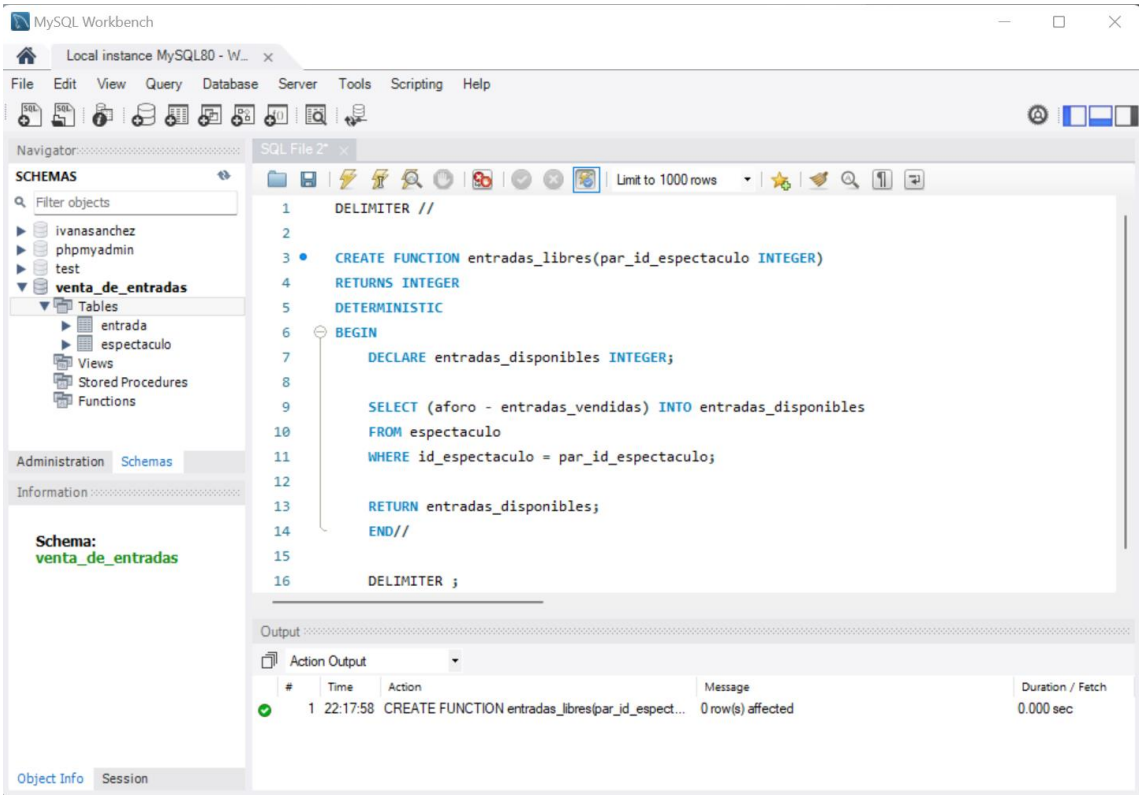


c) NUEVO SELECT DE LA TABLA ESPECTÁCULO



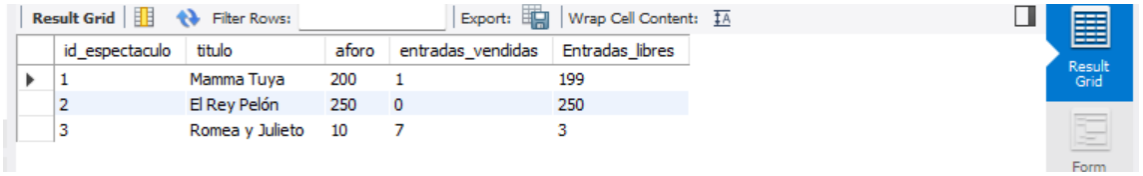
3.- Creación de la función “entradas_libres()”

Vamos a crear una función llamada entradas_libres que, al tomar como parámetro el ID de un espectáculo, va a devolver el número de entradas libres para dicho espectáculo. El cálculo se basará en el aforo total de éste y la cantidad de entradas ya vendidas.



3.1.- COMPROBACIONES

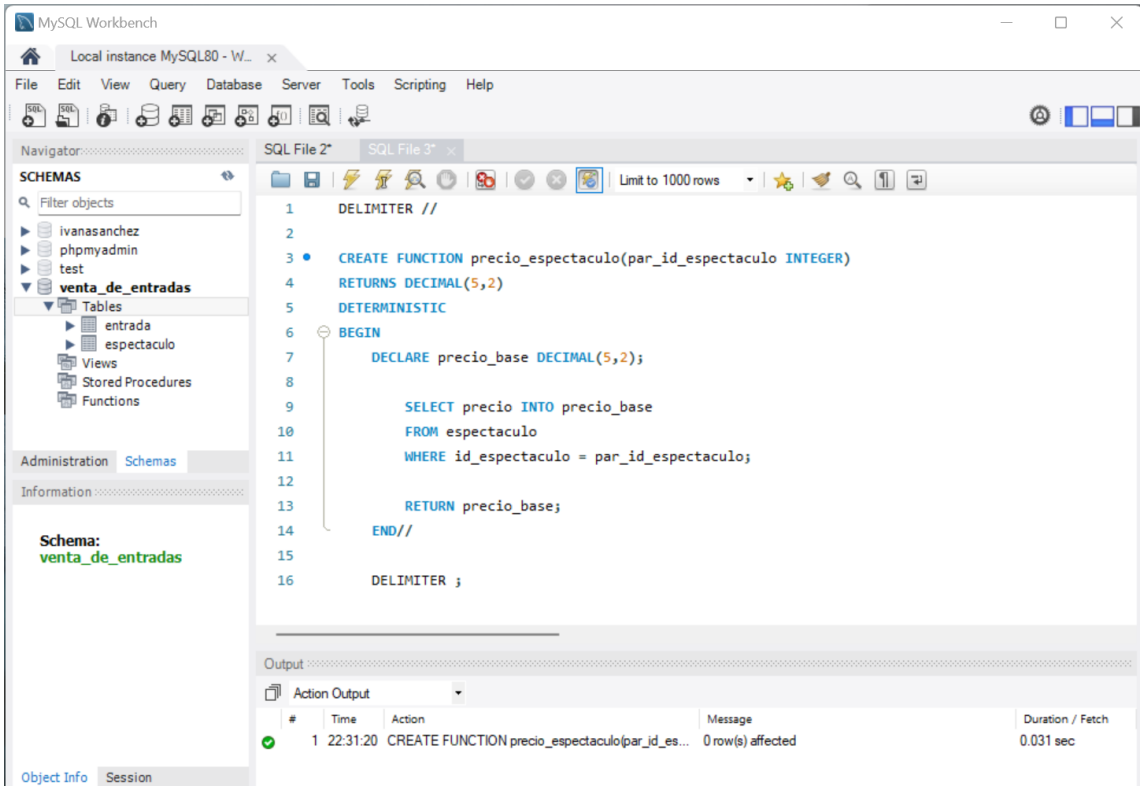
a) SELECT entradas_libres (id_espectaculo)



id_espectaculo	titulo	aforo	entradas_vendidas	Entradas_libres
1	Mamma Tuya	200	1	199
2	El Rey Pelón	250	0	250
3	Romea y Julieto	10	7	3

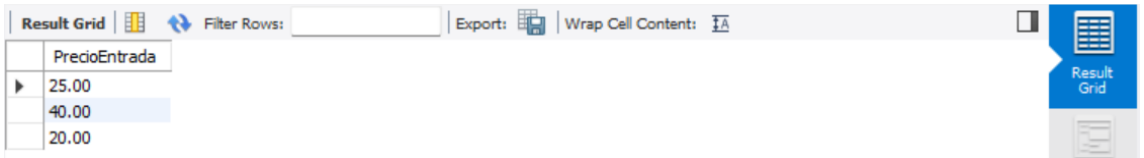
4.- Creación de la función “precio_espectáculo()”

Esta función toma como parámetros el ID de un espectáculo y devuelve el precio asociado a ese espectáculo. La función realiza una consulta a la tabla espectáculo para obtener el precio correspondiente al espectáculo identificado por el ID proporcionado.



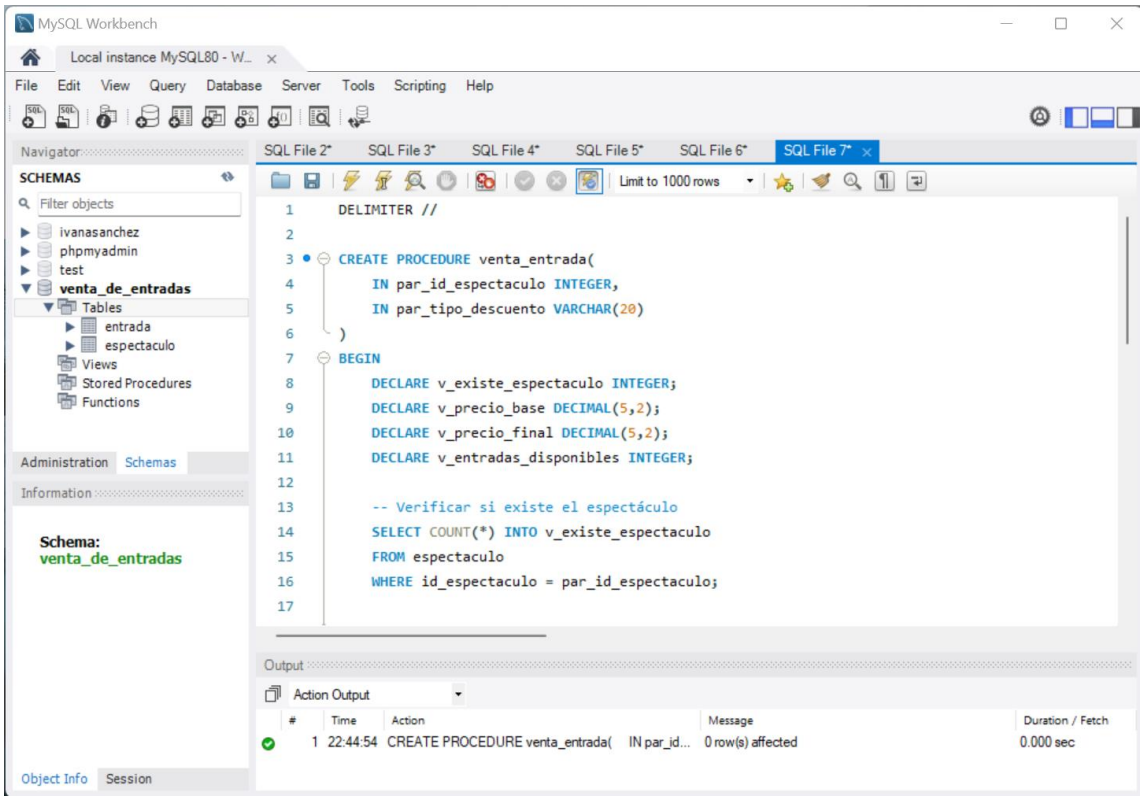
4.1.- COMPROBACIONES

a) SELECT precio_espectaculo(id espectáculo)



5.- Procedimiento “venta_entrada”

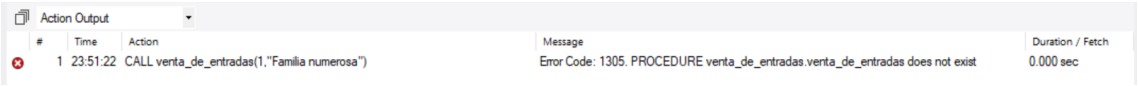
Procedimiento que permite vender entradas para espectáculos, teniendo en cuenta la existencia del espectáculo, la validez del tipo de descuento y la disponibilidad de entradas libres en el aforo del espectáculo.



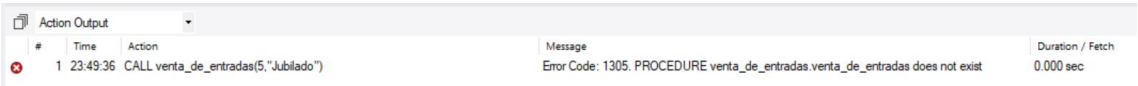
5.1.- COMPROBACIONES

a) REALIZACIÓN DE TODAS LAS LLAMADAS

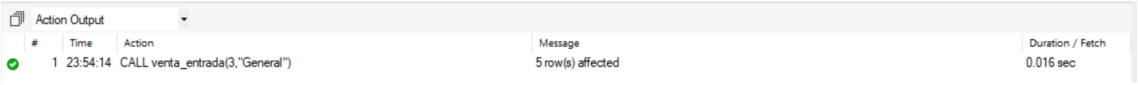
- CALL FAMILIA NUMEROSA



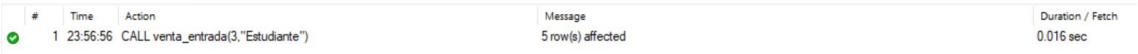
- CALL JUBILADO



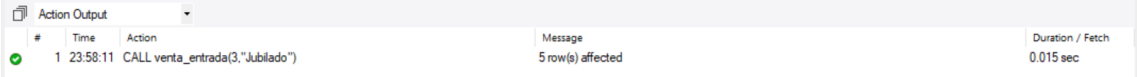
- CALL GENERAL



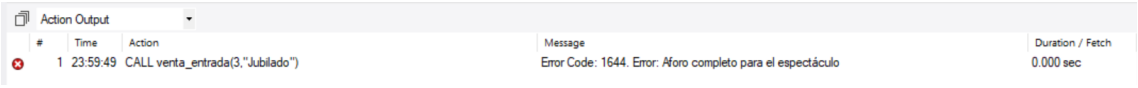
- CALL ESTUDIANTE



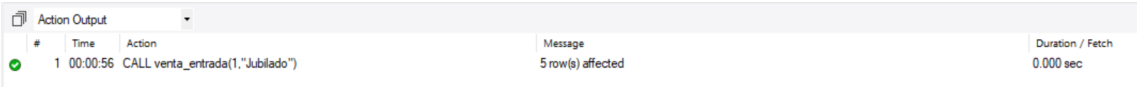
- CALL JUBILADO



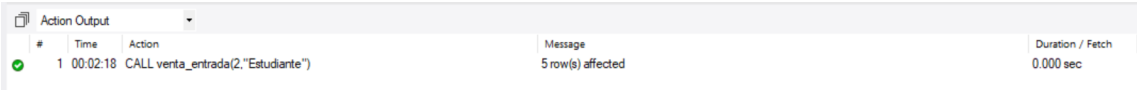
- CALL JUBILADO



- CALL JUBILADO

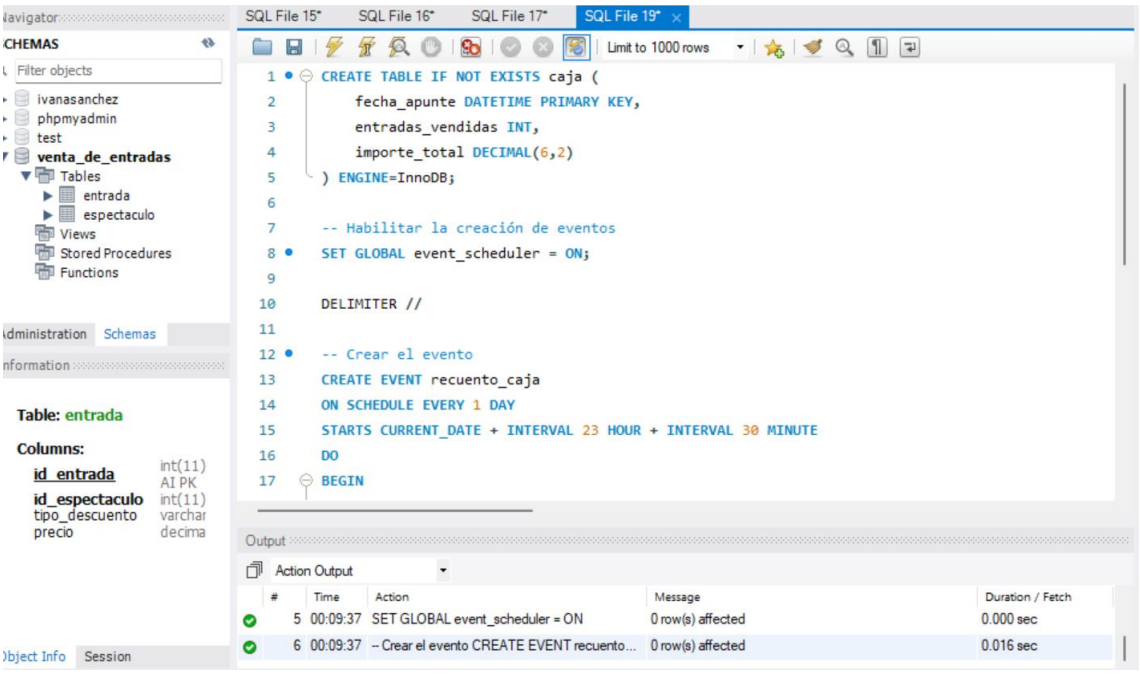


- CALL ESTUDIANTE



6.- Programar un evento: Recuento diario de caja

Como indica el enunciado, hemos creado un evento que se ejecuta cada 24 horas a las 23:30h y que realiza un recuento diario de las entradas vendidas, actualiza la columna entradas_vendidas a cero, elimina todas las filas de la tabla entrada y luego agrega una nueva fila en la tabla llamada caja con la información del recuento diario.



6.1.- COMPROBACIONES

Para hacer las comprobaciones, modificamos el script para que se ejecute cada 5 minutos.

```
DELIMITER //

DROP EVENT IF EXISTS recuento_caja//

CREATE EVENT recuento_caja
ON SCHEDULE EVERY 5 MINUTE
STARTS CURRENT_TIMESTAMP
DO
BEGIN
    DECLARE v_entradas_total INT;
    DECLARE v_importe_total DECIMAL(6,2);

    -- Calcular totales antes de borrar
    SELECT COUNT(*), SUM(precio)
    INTO v_entradas_total, v_importe_total
    FROM entrada;
```

a) SELECT DE ESPECTÁCULO, ENTRADA Y CAJA ANTES EVENTO

- ESPECTACULO

```
SELECT 'ESPECTACULOS' as tabla;
SELECT * FROM espectaculo;
```

id_espectaculo	titulo	precio	aforo	entradas_vendidas
1	Mamma Tuya	25.00	200	0
2	El Rey Pelón	40.00	250	0
3	Romea y Julieto	20.00	10	0

- ENTRADAS

```
SELECT 'ENTRADAS' as tabla;
SELECT * FROM entrada;
```

id_entrada	id_espectaculo	tipo_descuento	precio

- **CAJA**

```
SELECT 'CAJA' as tabla;
SELECT * FROM caja;
```

fecha_apunte	entradas_vendidas	importe_total
2024-11-15 00:15:51	7	156.00
2024-11-15 00:20:51	0	0.00
2024-11-15 00:25:51	0	0.00
NULL	NULL	NULL

b) HACEMOS UN SHOW EVENTS

Db	Name	Definer	Time zone	Type	Execute at	Interval value	Interval field
venta_de_entradas	recuento_caja	root@localhost	SYSTEM	RECURRING	NULL	5	MINUTE

c) SELECT DE ESPECTÁCULO, ENTRADA Y CAJA DESPUÉS DEL EVENTO

Tendremos los mismos resultados, pues no hemos realizado ningún cambio. Vuelvo a modificar el script para que tenga su configuración de 24 horas.

```
DELIMITER //

DROP EVENT IF EXISTS recuento_caja//

CREATE EVENT recuento_caja
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_DATE + INTERVAL 23 HOUR + INTERVAL 30 MINUTE
DO
BEGIN
    -- Mismo código del evento...
END//

DELIMITER ;
```

#	Time	Action	Message	Duration / Fetch
8	00:34:17	DROP EVENT IF EXISTS recuento_caja	0 row(s) affected	0.000 sec
9	00:34:17	CREATE EVENT recuento_caja ON SCHED...	0 row(s) affected	0.000 sec

7.- PROPUESTA NUEVA FUNCIONALIDAD

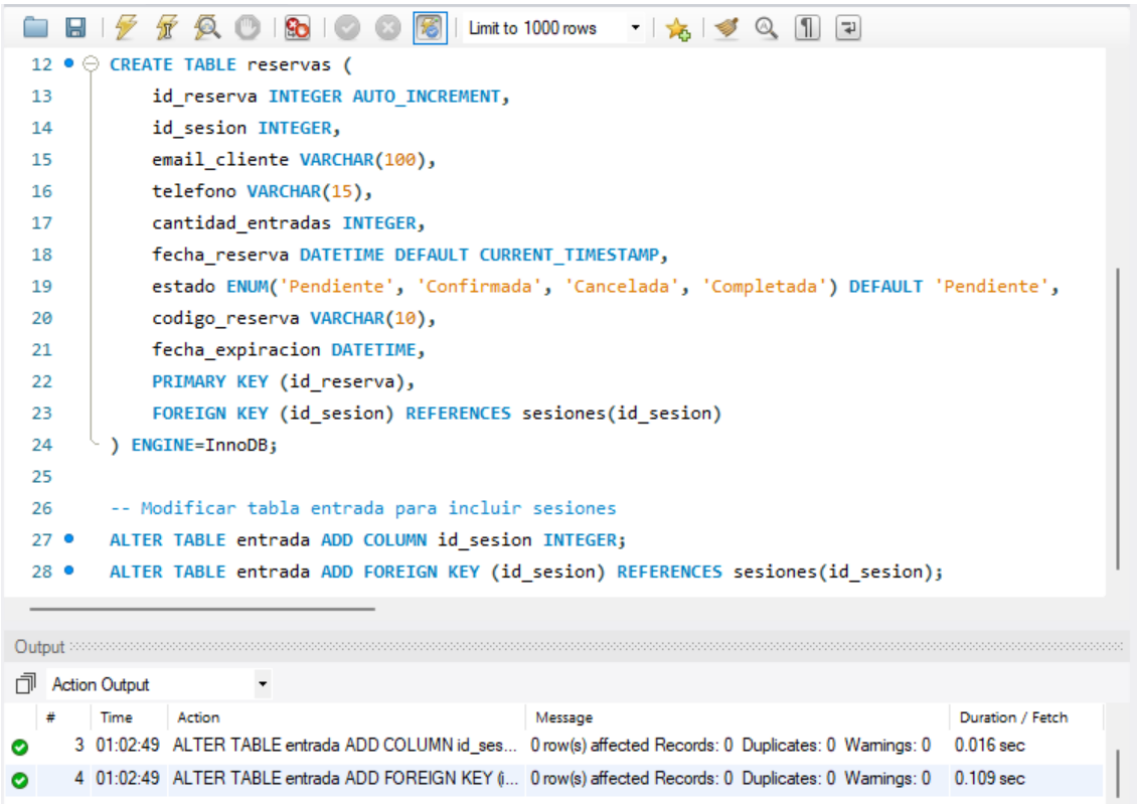
Me he decidido por crear un sistema de reservas anticipadas con gestión de sesiones múltiples por espectáculo. Se trata de una ampliación que permite una gestión de sesiones múltiples (distintas fechas y horas) y llevar un control de ellas (Programada, En venta, Finalizada y Cancelada)

Con este sistema de reservas los clientes podrán reservar entradas proporcionando email y teléfono, tendrán un código único que expirará en 24 horas y, además, contarán con un sistema del estado de su reserva: Pendiente, Confirmada, Cancelada y Completada.

Por último, se ha creado una gestión automática de limpieza de reservas expiradas, de disponibilidad y de generación de códigos de reservas.

7.1.- EXPLICACIÓN

Para que funcione esta ampliación, lo primero que he creado son las 2 tablas que nos van a hacer falta: sesiones y reservas, modificando para ello la tabla entrada para incluir sesiones.



Después de crear las tablas, nos toca los procedimientos para: crear una nueva sesión, validar que el espectáculo existe, validar que la fecha es futura, crear una reserva, obtener id_espectaculo, verificar disponibilidad, generar un código único, crear la reserva por 24 horas, devolver el código de reserva, confirmar una

reserva, obtener datos de la reserva, validaciones, actualizar estado y crear las entradas correspondientes.

```sql
DELIMITER //

-- Procedimiento para crear una nueva sesión
CREATE PROCEDURE crear_sesion(
    IN p_id_espectaculo INTEGER,
    IN p_fecha_hora DATETIME
)
BEGIN
    -- Validar que el espectáculo existe
    IF NOT EXISTS (SELECT 1 FROM espectaculo WHERE id_espectaculo = p_id_espectaculo) THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'El espectáculo no existe';
    END IF;

    -- Validar que la fecha es futura
    IF p_fecha_hora <= NOW() THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La fecha debe ser futura';
    END IF;
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
2	01:12:58	CREATE PROCEDURE crear_reserva(IN p...	0 row(s) affected	0.016 sec
3	01:12:58	CREATE PROCEDURE confirmar_reserva(...	0 row(s) affected	0.016 sec

- **CONFIRMACIÓN RESERVAS**

SQL File 32* | SQL File 33* | SQL File 34*

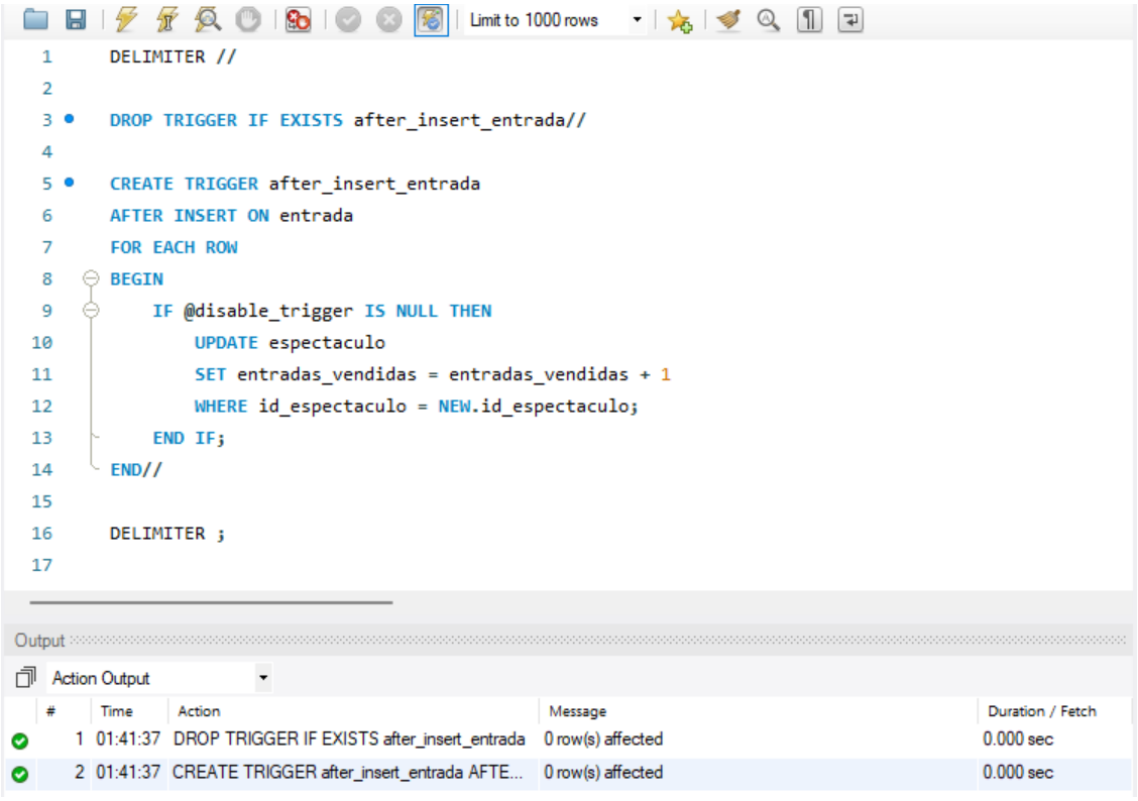
```sql
DELIMITER //

DROP PROCEDURE IF EXISTS confirmar_reserva//

CREATE PROCEDURE confirmar_reserva(
    IN p_codigo_reserva VARCHAR(10)
)
BEGIN
    DECLARE v_id_reserva INTEGER;
    DECLARE v_id_sesion INTEGER;
    DECLARE v_cantidad INTEGER;
    DECLARE v_estado VARCHAR(20);
    DECLARE v_fecha_exp DATETIME;
    DECLARE v_id_espectaculo INTEGER;
    DECLARE v_precio_base DECIMAL(5,2);

    -- Obtener datos de la reserva
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	01:38:54	DROP PROCEDURE IF EXISTS confirmar_rese...	0 row(s) affected	0.015 sec
2	01:38:54	CREATE PROCEDURE confirmar_reserva(I...	0 row(s) affected	0.016 sec

- **Modificación del trigger para que respete la variable de control**



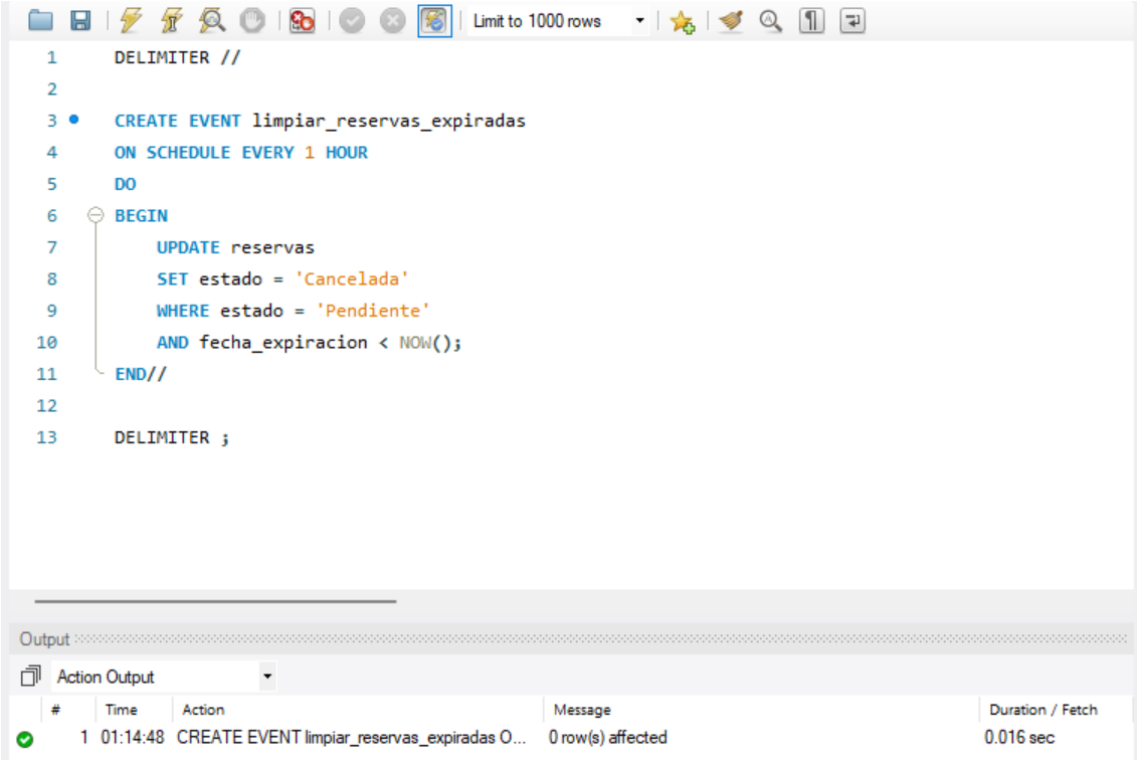
```
DELIMITER //

DROP TRIGGER IF EXISTS after_insert_entrada//

CREATE TRIGGER after_insert_entrada
AFTER INSERT ON entrada
FOR EACH ROW
BEGIN
    IF @disable_trigger IS NULL THEN
        UPDATE espectaculo
        SET entradas_vendidas = entradas_vendidas + 1
        WHERE id_espectaculo = NEW.id_espectaculo;
    END IF;
END//

DELIMITER ;
```

Output

Action Output

#	Time	Action	Message	Duration / Fetch
1	01:41:37	DROP TRIGGER IF EXISTS after_insert_entrada	0 row(s) affected	0.000 sec
2	01:41:37	CREATE TRIGGER after_insert_entrada AFTE...	0 row(s) affected	0.000 sec

- **Por último, crearemos la limpieza de las reservas expiradas**



```
DELIMITER //

CREATE EVENT limpiar_reservas_expiradas
ON SCHEDULE EVERY 1 HOUR
DO
BEGIN
    UPDATE reservas
    SET estado = 'Cancelada'
    WHERE estado = 'Pendiente'
    AND fecha_expiracion < NOW();
END//

DELIMITER ;
```

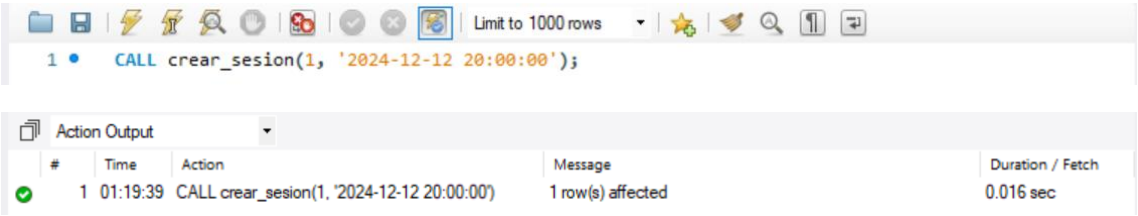
Output

Action Output

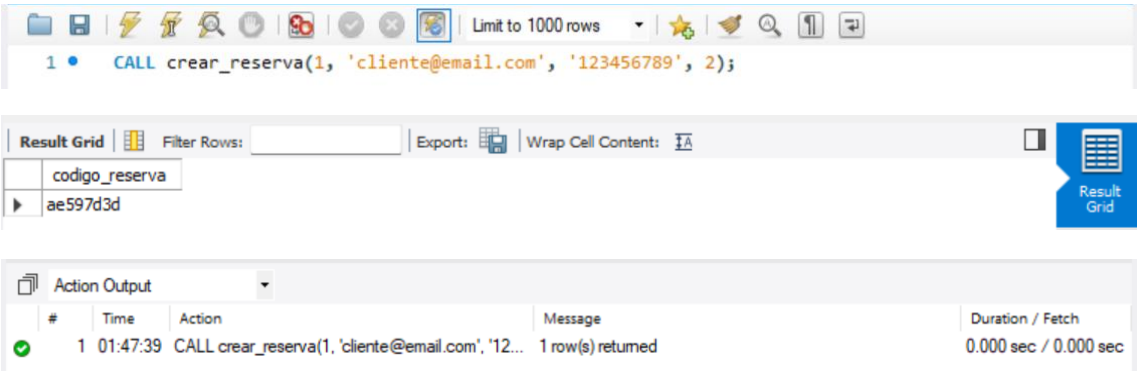
#	Time	Action	Message	Duration / Fetch
1	01:14:48	CREATE EVENT limpiar_reservas_expiradas O...	0 row(s) affected	0.016 sec

7.2.- COMPROBACIÓN

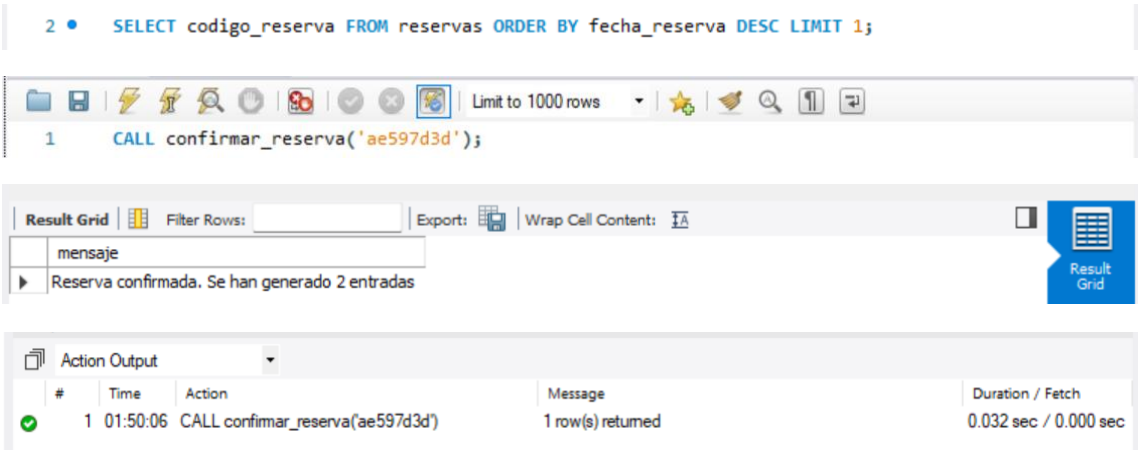
- CREAR UNA SESIÓN



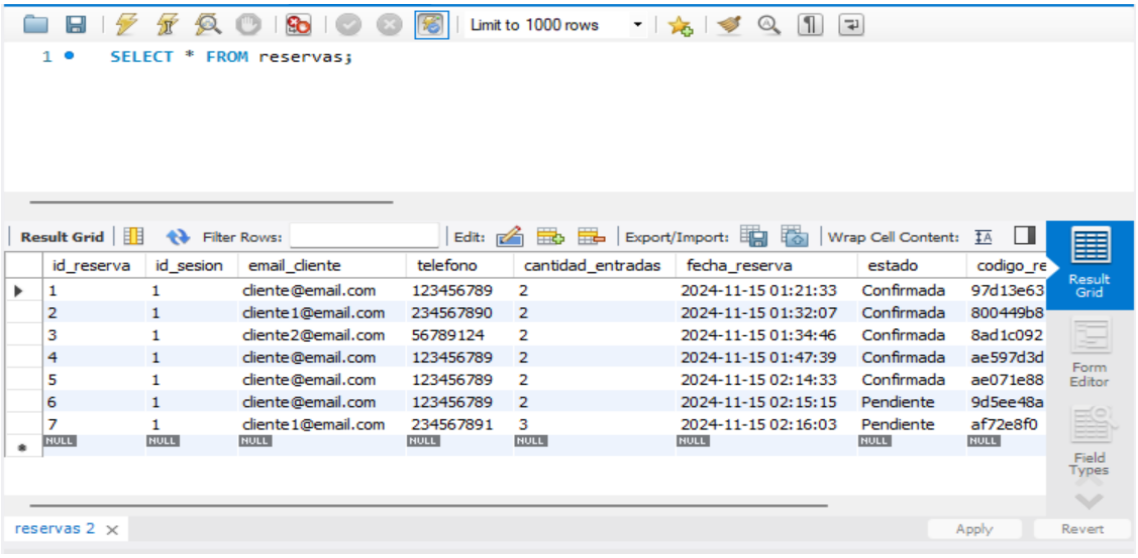
- CREAR UNA RESERVA



- CONFIRMAR UNA RESERVA



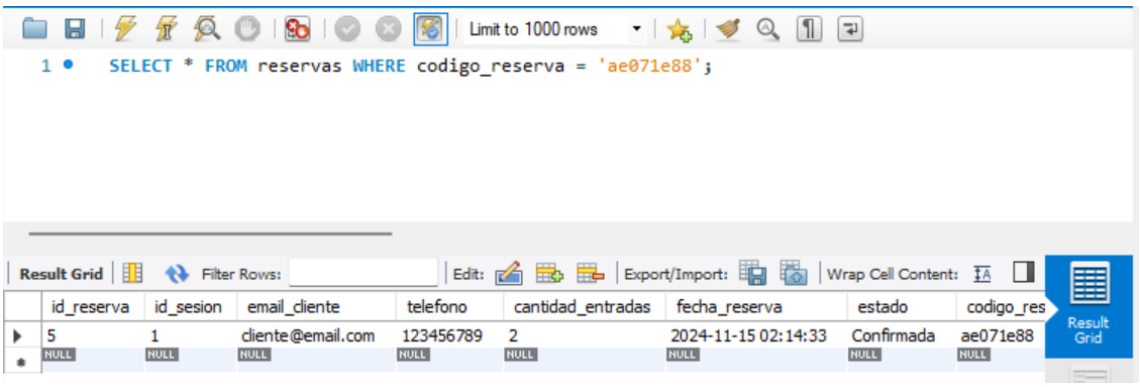
- **VER RESERVAR CONFIRMADAS**



```
SELECT * FROM reservas;
```

Result Grid

id_reserva	id_sesion	email_cliente	telefono	cantidad_entradas	fecha_reserva	estado	codigo_re
1	1	cliente@email.com	123456789	2	2024-11-15 01:21:33	Confirmada	97d13e63
2	1	cliente1@email.com	234567890	2	2024-11-15 01:32:07	Confirmada	800449b8
3	1	cliente2@email.com	56789124	2	2024-11-15 01:34:46	Confirmada	8ad1c092
4	1	cliente@email.com	123456789	2	2024-11-15 01:47:39	Confirmada	ae597d3d
5	1	cliente@email.com	123456789	2	2024-11-15 02:14:33	Confirmada	ae071e88
6	1	cliente@email.com	123456789	2	2024-11-15 02:15:15	Pendiente	9d5ee48a
7	1	cliente1@email.com	234567891	3	2024-11-15 02:16:03	Pendiente	af72e8f0

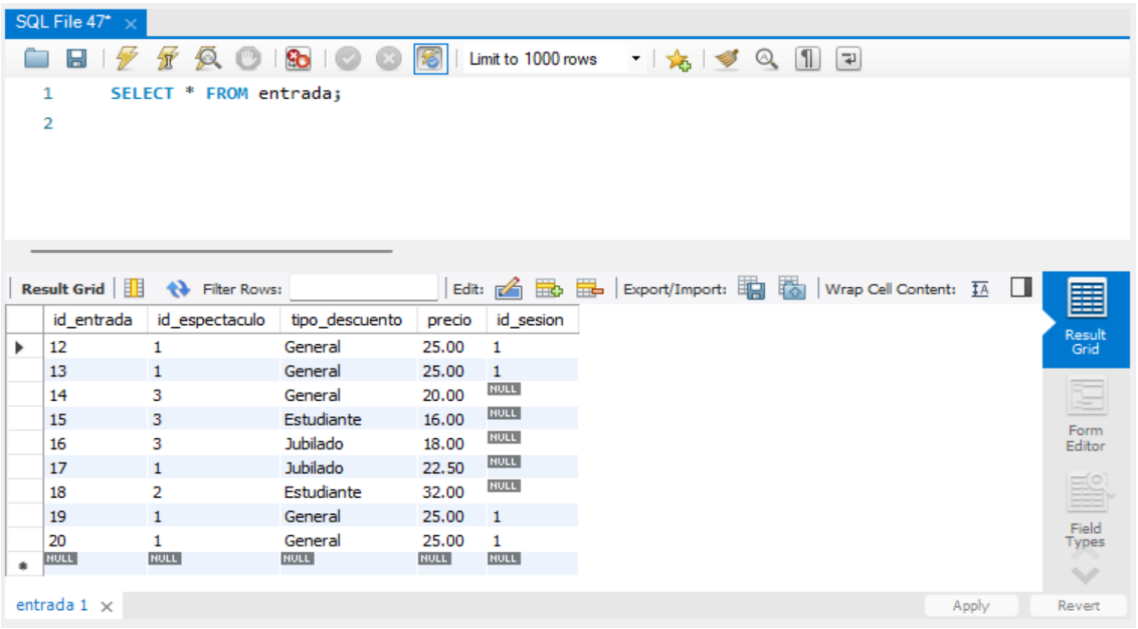


```
SELECT * FROM reservas WHERE codigo_reserva = 'ae071e88';
```

Result Grid

id_reserva	id_sesion	email_cliente	telefono	cantidad_entradas	fecha_reserva	estado	codigo_res
5	1	cliente@email.com	123456789	2	2024-11-15 02:14:33	Confirmada	ae071e88

- **VER ENTRADAS GENERADAS**


SQL File 47*
Limit to 1000 rows
1 SELECT * FROM entrada;
2
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
id_entrada id_espectaculo tipo_descuento precio id_sesion
12 1 General 25.00 1
13 1 General 25.00 1
14 3 General 20.00 NULL
15 3 Estudiante 16.00 NULL
16 3 Jubilado 18.00 NULL
17 1 Jubilado 22.50 NULL
18 2 Estudiante 32.00 NULL
19 1 General 25.00 1
20 1 General 25.00 1
NULL NULL NULL NULL NULL
Result Grid
Form Editor
Field Types
entrada 1
Apply
Revert