RETO FINAL

Gretty María Mosquera Taborda

SofkaU

Tabla de contenido

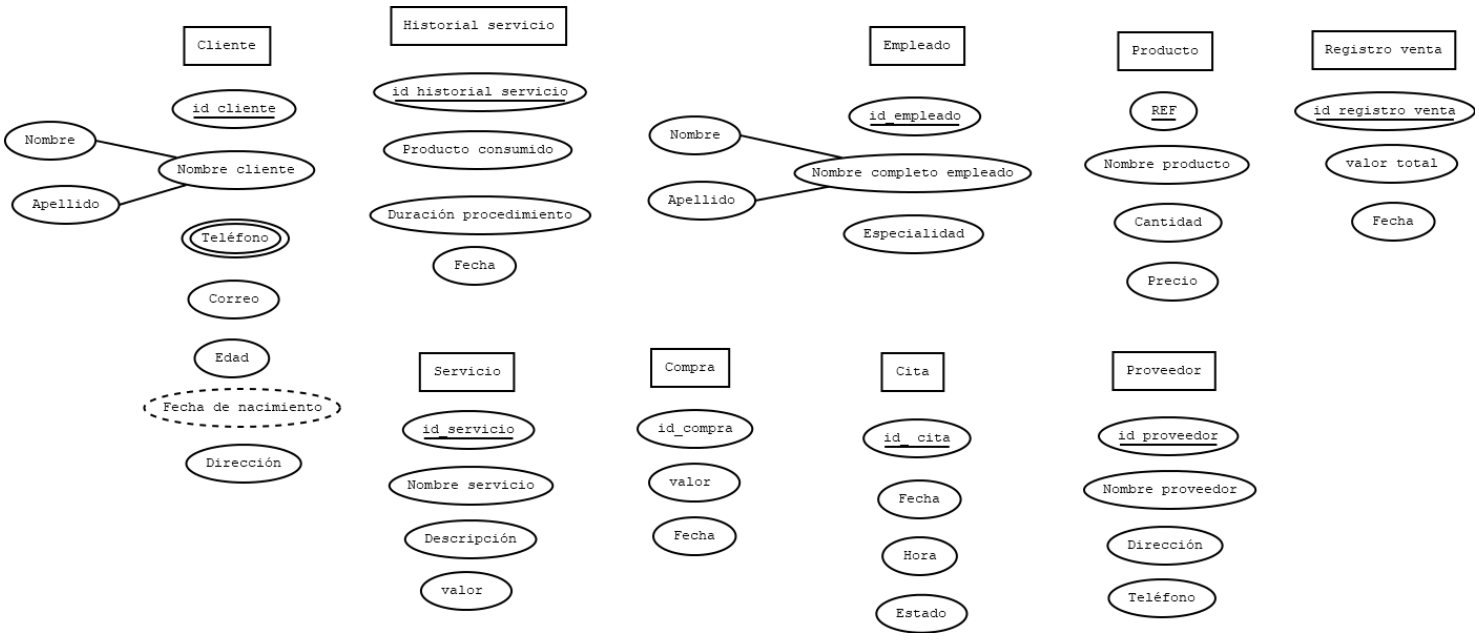
Barbería (Ejercicio A)

Una barbería desea llevar el control de sus empleados y de sus clientes, así como de los servicios que se prestan. Se desea almacenar la siguiente información:

- **Empleados**: ID, cedula, Nombre, Especialidad (Masaje, Corte, Cejas, etc.)
- **Clientes**: Datos personales (ID, cedula, Nombre, Profesión, Teléfono, correo, edad y Dirección).
- **Historial de Servicios prestados por la barbería**: Un registro para saber información del servicio prestado por un empleado a un cliente, productos consumidos, duración del procedimiento y fecha.
- **Citas**: Fecha y Hora en la que se cita al cliente y el barbero que realizará el servicio.
- **Productos vendidos por la barbería**: REF, Nombre, Cantidad y Precio.
- **Proveedor**: los productos vendidos deben tener una fuente.
- **Registro de Ventas**: Si un barbero vende un producto a un cliente, termina obteniendo una "liga" ganancia ocasional.

Construcción del modelo E-R

Se definen las siguientes entidades con sus respectivos atributos:



Entidades

- Cliente.
- Historial servicio.
- Servicio.
- Compra.
- Cita.
- Empleado.
- Producto.
- Proveedor.
- Registro venta.

***Nota*:** Adicional a las entidades y atributos que se mencionan en la descripción del taller, se cree necesario agregar:

- Una entidad servicio, esta se refiere a los servicios que puede ofrecer la barbería a sus clientes (ejemplo: afeitado de barba o corte de cabello) y la entidad de historial de servicios que ya estaba planteada, llevará un registro de los servicios que se han prestado a cada cliente en la barbería.
- Un atributo llamado estado en la entidad cita, el cual nos permitirá identificar si la cita sigue activa o fue cancelada.

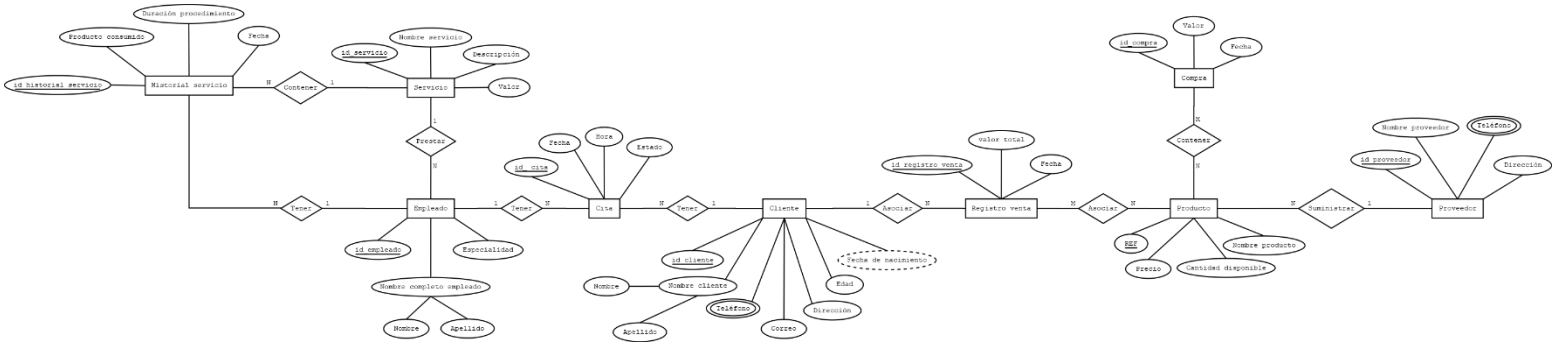
Se utilizan tres tipos de atributos

Compuestos: Este tipo de atributos se pueden descomponer en varios atributos más pequeños y simples. (En este caso el nombre completo, que podemos descomponerlo en dos atributos: *Nombre* y *apellido*).

Derivados: No es una información que se pueda recolectar directamente, sino que se puede obtener por medio de una operación o cálculo basado en otros atributos. (En este caso se agrega el atributo de *fecha de nacimiento*, ya que la *edad* no es una información que se recolecte directamente, sino que podemos calcularla por medio de la fecha de nacimiento).

Multivaluados: Estos atributos pueden tener varios valores diferentes para una sola entidad. (En este caso el teléfono, ya que una persona puede tener más de un número telefónico, por ejemplo: un número fijo y otro móvil).

Relaciones y asignación de la cardinalidad



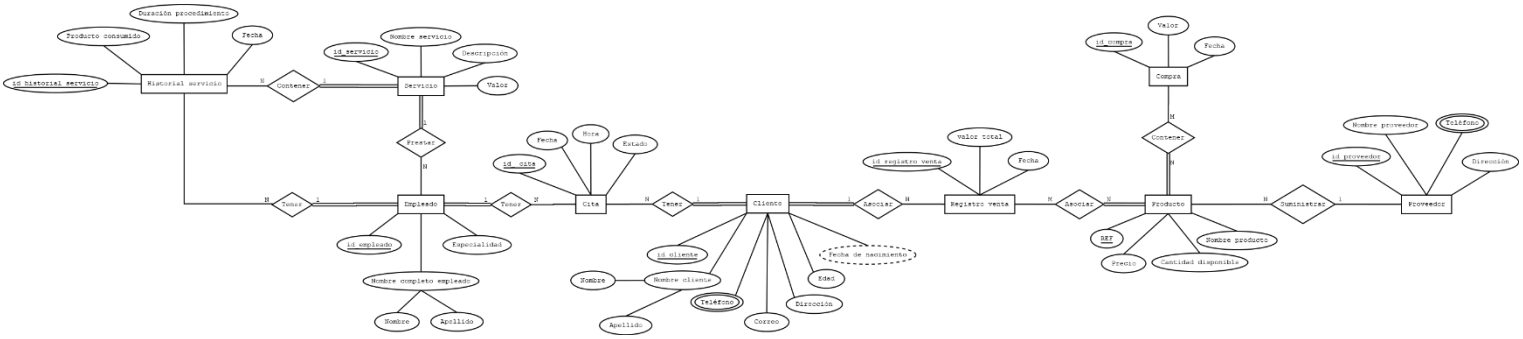
Relaciones

- Contener.
- Prestar.
- Tener.
- Asociar.
- Suministrar.

Cardinalidad

- Un **proveedor** puede *suministrar* **uno o más productos**, pero un **producto** puede ser *suministrado* por un **proveedor** (1:N).
- Una **compra** puede *contener* **uno o más productos** y un **producto** puede estar *contenido* en una o más **compras** (N:M).
- Un **registro de venta** está *asociado* a uno o más **productos** y un **producto** puede estar *asociado* a uno o más **registros de venta** (N:M).
- Un **cliente** puede tener *asociado* uno o más **registros de venta**, pero un **registro de venta** puede estar *asociado* a un **cliente** (1:N).
- Un **cliente** puede *tener* una o más **citas**, pero una **cita** solo puede estar programada para un **cliente** (1:N).
- Un **empleado** puede *tener* una o más **citas**, pero una **cita** solo puede ser programada para un **empleado** (1:N).
- Un **servicio** puede ser *prestado* por uno o más **empleados**, pero un **empleado** solo *prestará* un **servicio** en el momento de la atención (1:N).
- Un **empleado** puede *tener* uno o más **historiales de servicio**, pero un **historial de servicio** solo puede *tenerlo* un **empleado** (1:N).
- Un **servicio** puede estar *contenido* en una o más **historiales de servicio**, pero un **historial de servicio** solo lo *contiene* un **servicio** prestado por la barbería (1:N).

Resultado final



Construcción del modelo relacional

Con base al modelo E-R, se convierten las entidades en tablas con sus respectivos atributos.

tb_historial_servicio	
PK	id_historial_servicio
	producto_consumido
	duracion
	fecha
FK	id_empleado
FK	id_servicio

tb_servicio	
PK	id_servicio
	nombre_servicio
	descripcion
	valor

tb_empleado	
PK	id_empleado
	nombre_empleado
	apellido_empleado
	especialidad
FK	id_servicio

tb_cita	
PK	id_cita
	fecha
	hora
	estado
FK	id_cliente
FK	id_empleado

tb_cliente	
PK	id_cliente
	nombre_cliente
	apellido_cliente
	fecha_nacimiento
	correo
	direccion

tb_registro_venta	
PK	id_registro_venta
	valor
	fecha
FK	id_cliente

tb_producto	
PK	ref_producto
	nombre_producto
	cantidad_disponible
	precio
FK	id_proveedor

tb_proveedor	
PK	id_proveedor
	nombre_proveedor
	direccion

tb_compra	
PK	id_compra
	valor
	fecha

A causa del *atributo compuesto* se agregan dos atributos a la tabla cliente y empleado (en este caso nombre y apellido).

tb_empleado	
PK	id_empleado
	nombre_empleado
	apellido_empleado
	especialidad
FK	id_servicio

tb_cliente	
PK	id_cliente
	nombre_cliente
	apellido_cliente
	fecha_nacimiento
	correo
	direccion

A causa del atributo derivado se agrega un campo fecha_nacimiento, el cual nos permitiría si es necesario, calcular la edad del cliente.

tb_cliente	
PK	id_cliente
	nombre_cliente
	apellido_cliente
	fecha_nacimiento
	correo
	direccion

A causa del atributo multivaluado se crean dos tablas teléfono para cliente y proveedor, ya que podrían tener más de un número de teléfono (ejemplo: un número fijo y otro móvil).

tb_telefono_cliente	
PK	id_telefono_cliente
PK	telefono

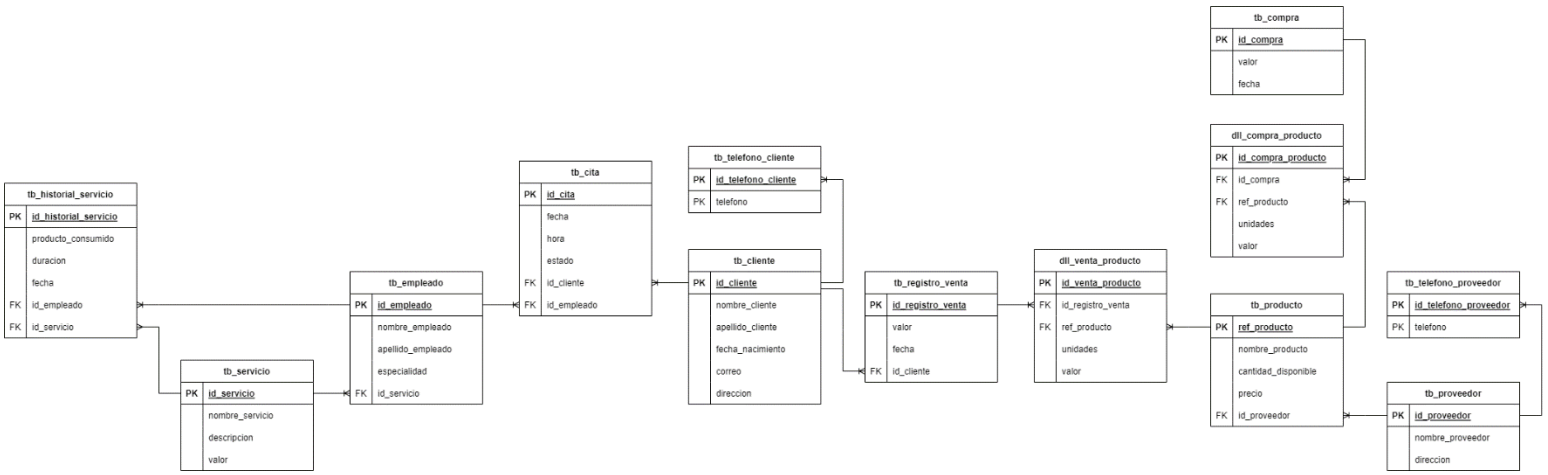
tb_telefono_proveedor	
PK	id_telefono_proveedor
PK	telefono

Se crean las tablas de detalle por el tipo de relación muchos a muchos (N:M)

dll_venta_producto	
PK	id_venta_producto
FK	id_registro_venta
FK	ref_producto
	unidades
	valor

dll_compra_producto	
PK	id_compra_producto
FK	id_compra
FK	ref_producto
	unidades
	valor

Al crear la relación entre cada una de las tablas mencionadas previamente, este es el resultado final.



Sentencias SQL

Con base al resultado del modelo relacional, se inicia la construcción del script por medio de sentencias SQL.

Creación de la base de datos

```sql
-- Creación de la base de datos barbería
CREATE DATABASE db_barberia;

-- Indica cuál será la BD a la cual se le aplicarán las siguientes consultas:
USE db_barberia;
```

Creación de una tabla

```sql
CREATE TABLE tb_servicio(
id_servicio VARCHAR(200) NOT NULL,
nombre_servicio VARCHAR(200) NOT NULL,
descripcion VARCHAR(200) NOT NULL,
valor INT NOT NULL,
PRIMARY KEY(id_servicio)
);
```

Creación de relación entre tablas (1:N)

```sql
CREATE TABLE tb_registro_venta(
id_registro_venta VARCHAR(200) NOT NULL,
valor INT NOT NULL,
fecha VARCHAR(200) NOT NULL,
id_cliente VARCHAR(200) NOT NULL,
PRIMARY KEY(id_registro_venta),
FOREIGN KEY(id_cliente) REFERENCES tb_cliente(id_cliente)
);
```

Creación de una tabla con clave compuesta

```sql
CREATE TABLE tb_telefono_cliente(
id_telefono_cliente VARCHAR(200) NOT NULL,
telefono VARCHAR(200) NOT NULL,
PRIMARY KEY(id_telefono_cliente, telefono),
FOREIGN KEY(id_telefono_cliente) REFERENCES tb_cliente(id_cliente)
);
```

Creación de tablas de detalle (N:M)

```
CREATE TABLE dll_compra_producto(
id_compra_producto VARCHAR(200) NOT NULL,
id_compra VARCHAR(200) NOT NULL,
id_producto VARCHAR(200) NOT NULL,
unidades INT NOT NULL,
valor INT NOT NULL,
PRIMARY KEY(id_compra_producto),
FOREIGN KEY(id_compra) REFERENCES tb_compra(id_compra),
FOREIGN KEY(id_producto) REFERENCES tb_producto(id_producto)
);
```

Normalización

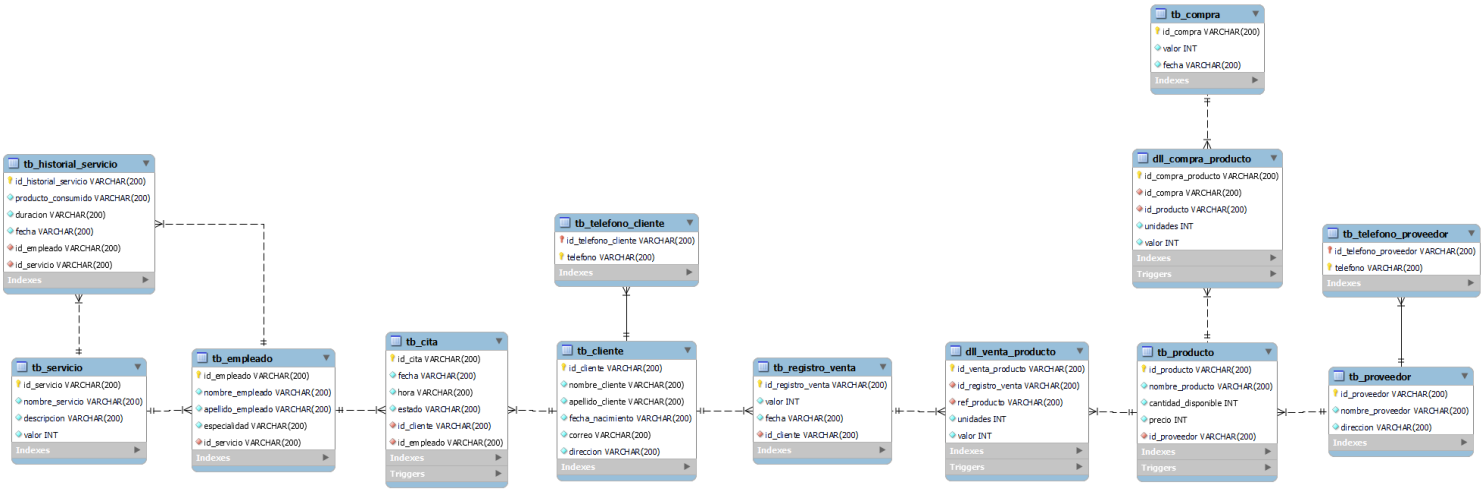
Primera forma de normalización	**Cumple**
Todos los atributos tienen valores atómicos.	✓
No hay atributos multivaluados.	✓
No deben existir registros duplicados.	✓
Las columnas repetidas deben eliminarse y colocarse agrupadas en tablas separadas bajo un contexto.	✓
Definir clave principal.	✓

Segunda forma de normalización	**Cumple**
Estar en 1FN	✓
Todos los valores de las columnas deben depender únicamente de la llave primaria de la tabla.	✓
Las tablas deben tener una única llave primaria que identifique a la tabla y que sus atributos dependen de ella, relación separada.	✓

Tercera forma de normalización	**Cumple**
Estar en 2FN.	✓
Cada atributo que no está incluido en la clave primaria no depende transitivamente de la clave primaria.	✓

Diagrama del modelo relación (EER) generado por Workbench

Un diagrama relacional representa las tablas que forman parte de la base de datos, y muestra cómo están relacionadas entre sí.



Identificar esto ayuda a comprender cómo se estructura la información en la base de datos y a entender cómo se relacionan las diferentes tablas para construir las consultas que se mostrarán a continuación:

Consultas que permiten ver la información de cada tabla o de varias tablas.

Consulta #1. Mostrar todos los empleados que trabajan en la barbería

```
230  SELECT id_empleado AS "Cédula", CONCAT(nombre_empleado, " ",apellido_empleado) AS "Nombre completo empleado"
231  FROM tb_empleado;
232
```

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

Cédula	Nombre completo empleado
1000000	Patricia Oberbrunner
1000001	Ryan Murazik
1000002	Jesse Miller
1000003	Nerissa Wehner
1000004	Otto Stoltenberg
1000005	Mason Oberbrunner
1000006	Jenice Hammes
1000007	Bonita Klocko
1000008	Shae Lang
1000009	Rosaura Zulauf
1000010	Sharilyn Wisoky
1000011	Janie Murazik
1000012	Ezra Gaylord
1000013	Kirk Nienow
1000014	Bobbi Mraz
1000015	Dong Kessler
1000016	Ping Schimmel
1000017	Tristan Cole
1000018	Margherita Fahey

Result 7

Consulta #2. Mostrar todos los clientes de la barbería y su localidad

```
234 • SELECT id_cliente AS "Cédula cliente", CONCAT(nombre_cliente, apellido_cliente) AS "Nombre completo", direccion AS "Dirección"
235   FROM tb_cliente;
```

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

Cédula cliente	Nombre completo	Dirección
1000000	CandaceRitchie	Minton
1000001	FaustoHuels	New Cris
1000002	MarlonKemmer	Port Stacyton
1000003	MarcusWitting	North Lorelei
1000004	WilmerMitchell	Hodkiewiczshire
1000005	NicholasWalter	East Darell
1000006	RalphTreutel	North Winstonshire
1000007	JolieGreenholt	Toytown
1000008	KipMitchell	Volkmanchester
1000009	MickeyGleichner	Hegmannshire
1000010	UnMcDermott	Nathanaelville
1000011	EvelynDickens	Blickmouth
1000012	ArgentinaBarton	Calistafort
1000013	DeannHackett	West Jerilynport
1000014	LaverneSauer	Rustyland
1000015	ShelbyBrakus	Port Joslynport
1000016	AlfredMcDermott	New Mellissa
1000017	ReneHilpert	North Lakieshatown
1000018	VictorLubowitz	Kizziechester

Result 10

Consulta #3. Mostrar todas las citas programadas para un empleado en especial (en este caso que contenga el id: 1000044)

```
238 • SELECT tb_empleado.id_empleado AS "Cédula del empleado",
239   CONCAT(tb_empleado.nombre_empleado, " ", tb_empleado.apellido_empleado) AS "Nombre del empleado",
240   tb_cliente.id_cliente AS "Cédula del cliente",
241   CONCAT(tb_cliente.nombre_cliente, " ", tb_cliente.apellido_cliente) AS "Nombre del cliente"
242   FROM tb_empleado
243   INNER JOIN tb_cita ON tb_cita.id_empleado = tb_empleado.id_empleado
244   INNER JOIN tb_cliente ON tb_cliente.id_cliente = tb_cita.id_cliente
245   WHERE tb_empleado.id_empleado = '1000044';
246
```

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

Cédula del empleado	Nombre del empleado	Cédula del cliente	Nombre del cliente
1000044	Delbert Powlowski	1000044	Tammara Heidenreich

Consulta #4. Mostrar el id y el nombre, de los proveedores de productos en la barbería, donde la dirección empiece por Lake

```
248 • SELECT id_proveedor AS "Id proveedor", nombre_proveedor AS "Nombre del proveedor", direccion
249   FROM tb_proveedor
250   WHERE direccion LIKE 'Lake%';
```

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

Id proveedor	Nombre del proveedor	direccion
1000000	Nathaniel	Lake Ernestoberg
1000006	Willa	Lake Maurice
1000020	Joane	Lake Lincolnshire
1000029	Claudine	Lake Allyson
1000032	Lin	Lake Ardenstad
1000035	Hanna	Lake Bradyton
1000046	Tamra	Lake Ermaberg

Consulta #5. Mostrar todos los registros de venta que existen hasta el momento

```
253 •  SELECT tb_registro_venta.id_registro_venta AS "Id registros de venta",
254    tb_registro_venta.valor,
255    tb_registro_venta.fecha,
256    tb_registro_venta.id_cliente AS "Cédula cliente",
257    CONCAT (tb_cliente.nombre_cliente, " ",tb_cliente.apellido_cliente) AS "Nombre cliente"
258    FROM tb_registro_venta
259    INNER JOIN tb_cliente ON tb_cliente.id_cliente = tb_registro_venta.id_cliente
260
```

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

Id registros de venta	valor	fecha	Cédula cliente	Nombre cliente
1000000	40208	01/12/1991	1000000	Candace Ritchie
1000001	11312	09/11/1979	1000001	Fausto Huels
1000002	14109	10/12/1978	1000002	Marlon Kemmer
1000003	45473	08/12/1977	1000003	Marcus Witting
1000004	4872	04/26/1961	1000004	Wilmer Mitchell
1000005	42791	10/29/1973	1000005	Nicholas Walter
1000006	10935	01/23/1971	1000006	Ralph Treutel
1000007	24931	07/14/1972	1000007	Jolie Greenholt
1000008	46513	04/25/1965	1000008	Kip Mitchell
1000009	42389	01/03/2000	1000009	Mickey Gleichner
1000010	35045	08/24/1964	1000010	Un McDermott
1000011	27676	03/03/1996	1000011	Evelyn Dickens
1000012	26918	06/17/1962	1000012	Argentina Barton

Result 37

Consulta #6. Mostrar nombre de los empleados y los servicios que prestan en la barbería

```
262 •  SELECT CONCAT(nombre_empleado, apellido_empleado) AS "Nombre del empleado", especialidad AS Servicio
263    FROM tb_empleado
264    ORDER BY nombre_empleado;
265
```

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

Nombre del empleado	Servicio
AnderaArmstrong	Central Specialist
BobbiMraz	Global Banking Orchestrator
BonitaKlocko	Design Executive
BritniBechtelar	Government Coordinator
ClaudeNader	Mining Designer
DaiseyWuckert	Principal Advertising Coordinator
DariusMraz	Administration Analyst

Result 39

Consulta #7. Mostrar los clientes que nacieron en el año 1999

```
267 •  SELECT CONCAT(nombre_cliente, apellido_cliente) AS "Nombre del cliente", fecha_nacimiento
268    FROM tb_cliente
269    WHERE fecha_nacimiento LIKE '%1999';
270
```

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

Nombre del cliente	fecha_nacimiento
CandaceRitchie	10/19/1999
StuartSimonis	03/28/1999
GeriWalker	11/29/1999

Consulta #8. Mostrar el precio de los productos donde su rango esté entre 5000 y 15000

```
272 • SELECT id_producto AS "Id del producto", nombre_producto AS "Nombre del producto", precio
273   FROM tb_producto
274   WHERE precio BETWEEN 5000 AND 15000;
```

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

Id del producto	Nombre del producto	precio
0000000002	Intelligent Leather Hat	5313
0000000011	Lightweight Aluminum Plate	5330
0000000014	Rustic Wool Bench	6561
0000000017	Incredible Plastic Computer	5149
0000000023	Intelligent Steel Table	5436
0000000034	Awesome Silk Car	9043
0000000039	Lightweight Copper Pants	12331
0000000043	Heavy Duty Paper Clock	13807
0000000045	Sleek Linen Wallet	7199
0000000048	Synergistic Aluminum Computer	11525
0000000053	Gorgeous Plastic Car	13553
0000000065	Practical Concrete Hat	11764
0000000067	Intelligent Bronze Gloves	6721
0000000069	Small Aluminum Bottle	8762
0000000073	Durable Silk Watch	10614
0000000084	Sleek Steel Clock	9527
0000000091	Intelligent Rubber Coat	7282

tb_producto 51 ×

Consulta #9. Mostrar los productos donde la cantidad disponible del producto sea menor a 5

```
277 • SELECT id_producto AS "Id del producto", nombre_producto AS "Nombre del producto", cantidad_disponible AS Stock
278   FROM tb_producto
279   WHERE cantidad_disponible < 5;
280
```

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

Id del producto	Nombre del producto	Stock
0000000001	Ergonomic Marble Knife	3
0000000003	Durable Wool Bottle	3
0000000013	Synergistic Cotton Knife	4
0000000022	Ergonomic Rubber Computer	4
0000000041	Intelligent Rubber Plate	1
0000000056	Lightweight Marble Table	4
0000000057	Heavy Duty Wooden Lamp	3
0000000058	Gorgeous Copper Car	3
0000000060	Synergistic Aluminum Computer	3
0000000062	Durable Iron Gloves	2
0000000090	Ergonomic Copper Coat	2
0000000094	Incredible Paper Chair	4
0000000096	Fantastic Cotton Shirt	1

Consulta #10. Mostrar el nombre de todos los clientes de manera ascendente

```
282 •  SELECT CONCAT(nombre_cliente," " ,apellido_cliente) AS "Lista de clientes"
283    FROM tb_cliente
284    ORDER BY nombre_cliente ASC
```

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

Lista de clientes
Alfred McDermott
Argentina Barton
Bertie Williamson
Candace Ritchie
Connie Wyman
Deann Hackett
Deeanna Reichel
Dick Keebler
Donny Aufderhar
Edmond Nikolaus
Effie McLaughlin
Evelyn Dickens
Fausto Huels
Felisha Hagenes
Floyd Gleason
Franklin Nikolaus
Geri Walker
Glenda Hickle

Result 63

¿Qué es una vista?

Una vista es una tabla virtual que se guarda en la base de datos, pero no como estructura sino como consultas con un nombre que la identifica y se utilizan para guardar consultas que se utilizan o ejecutan de manera frecuente.

Sintaxis para crear una vista

```
CREATE VIEW view_name AS
SELECT column1, column2,…
FROM table_name
WHERE condition;
```

Creación de vistas

Vista #1

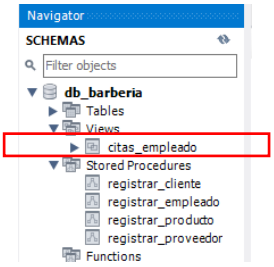
Se crea una vista la cual contiene una consulta que une la información de la tabla cita con la información de la tabla empleado, para brindar un acceso rápido de las citas que tiene asignadas cada empleado, al ser una vista nos permite filtrar por un empleado en específico (en este caso que contenga el id: 1000044).

Se utiliza la siguiente sentencia SQL para crear una vista:

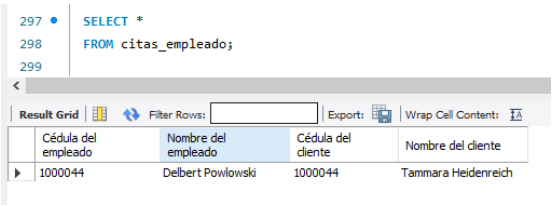
```
CREATE VIEW citas_empleado AS
SELECT tb_empleado.id_empleado AS "Cédula del empleado",
CONCAT(tb_empleado.nombre_empleado, " ", tb_empleado.apellido_empleado) AS "Nombre del empleado",
tb_cliente.id_cliente AS "Cédula del cliente",
CONCAT(tb_cliente.nombre_cliente, " ", tb_cliente.apellido_cliente) AS "Nombre del cliente"
FROM tb_empleado
INNER JOIN tb_cita ON tb_cita.id_empleado = tb_empleado.id_empleado
INNER JOIN tb_cliente ON tb_cliente.id_cliente = tb_cita.id_cliente
WHERE tb_empleado.id_empleado = '1000044';
```

```
85  13:16:25  CREATE VIEW citas_empleado AS SELECT tb_empleado.id_empleado AS "Cédula del empleado",  CONCAT(tb_empleado.nombre_empleado, " ", tb_...  0 row(s) affected
```

Una vez creada la vista, podremos visualizarla en la parte superior izquierda, en el panel de *Navigator* de Workbench, posteriormente presionar la opción *Views* para desplegar las vistas disponibles.



La información de las vistas puede ser accedida de la misma manera que se accede a la información de una tabla, utilizando la sentencia *SELECT* de la siguiente manera:



Vista #2

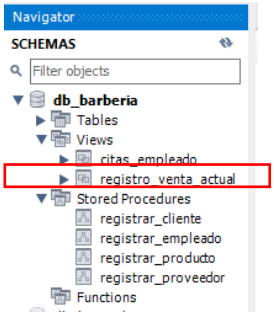
Se crea una vista la cual contiene una consulta que une la información de la tabla registro venta con la información de la tabla cliente, para brindar un acceso rápido de los registros de ventas por cliente que existen hasta el momento.

Se utiliza la siguiente sentencia SQL para crear una vista:

```
CREATE VIEW registro_venta_actual AS
SELECT tb_registro_venta.id_registro_venta AS "Id registros de venta",
tb_registro_venta.valor,
tb_registro_venta.fecha,
tb_registro_venta.id_cliente AS "Cédula cliente",
CONCAT (tb_cliente.nombre_cliente, " ",tb_cliente.apellido_cliente) AS "Nombre cliente"
FROM tb_registro_venta
INNER JOIN tb_cliente ON tb_cliente.id_cliente = tb_registro_venta.id_cliente;
```

```
86  13:19:35  CREATE VIEW registro_venta_actual AS  SELECT tb_registro_venta.id_registro_venta AS "Id registros de venta",  tb_registro_venta.valor,  tb_registro...  0 row(s) affected
```

Una vez creada la vista, podremos visualizarla en la parte superior izquierda, en el panel de *Navigator* de Workbench, posteriormente presionar la opción *Views* para desplegar las vistas disponibles.



La información de las vistas puede ser accedida de la misma manera que se accede a la información de una tabla, utilizando la sentencia *SELECT* de la siguiente manera:

```
310 •  SELECT *
311    FROM registro_venta_actual;
```

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

Id registros de venta	valor	fecha	Cédula cliente	Nombre cliente
1000000	40208	01/12/1991	1000000	Candace Ritchie
1000001	11312	09/11/1979	1000001	Fausto Huels
1000002	14109	10/12/1978	1000002	Marlon Kemmer
1000003	45473	08/12/1977	1000003	Marcus Witting
1000004	4872	04/26/1961	1000004	Wilmer Mitchell
1000005	42791	10/29/1973	1000005	Nicholas Walter
1000006	10935	01/23/1971	1000006	Ralph Treutel
1000007	24931	07/14/1972	1000007	Jolie Greenholt
1000008	46513	04/25/1965	1000008	Kip Mitchell
1000009	42389	01/03/2000	1000009	Mickey Gleichner

registro_venta_actual 66 ×

Vista #3

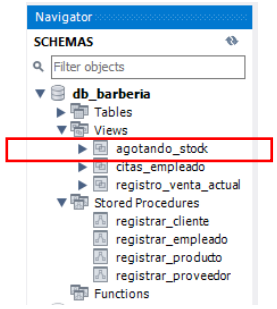
Se crea una vista la cual contiene una consulta con la información de la tabla producto, para brindar un acceso rápido de los productos donde la cantidad disponible sea menor a 5 y de esta manera identificar stock bajo en productos.

Se utiliza la siguiente sentencia SQL para crear una vista:

```
CREATE VIEW agotando_stock AS
SELECT id_producto AS "Id del producto", nombre_producto AS "Nombre del producto", cantidad_disponible AS Stock
FROM tb_producto
WHERE cantidad_disponible < 5;
```

```
87  14:31:14  CREATE VIEW agotando_stock AS  SELECT id_producto AS "Id del producto", nombre_producto AS "Nombre del producto", cantidad_disponible AS...  0 row(s) affected
```

Una vez creada la vista, podremos visualizarla en la parte superior izquierda, en el panel de *Navigator* de Workbench, posteriormente presionar la opción *Views* para desplegar las vistas disponibles.



La información de las vistas puede ser accedida de la misma manera que se accede a la información de una tabla, utilizando la sentencia *SELECT* de la siguiente manera:

```
319 •  SELECT *
320    FROM agotando_stock;
321
```

Result Grid | Filter Rows: | Export:

Id del producto	Nombre del producto	Stock
0000000001	Ergonomic Marble Knife	3
0000000003	Durable Wool Bottle	3
0000000013	Synergistic Cotton Knife	4
0000000022	Ergonomic Rubber Computer	4
0000000041	Intelligent Rubber Plate	1
0000000056	Lightweight Marble Table	4
0000000057	Heavy Duty Wooden Lamp	3
0000000058	Gorgeous Copper Car	3
0000000060	Synergistic Aluminum Computer	3
0000000062	Durable Iron Gloves	2

agotando_stock67

Vista #4

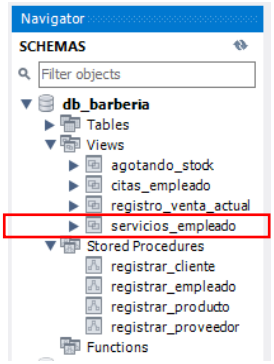
Se crea una vista la cual contiene una consulta con la información de la tabla empleado, para mostrar los datos en un formato de lista impresión (nombre concatenado con el apellido y su correspondiente especialidad).

Se utiliza la siguiente sentencia SQL para crear una vista:

```
CREATE VIEW servicios_empleado AS
SELECT CONCAT(nombre_empleado, " ",apellido_empleado) AS "Nombre del empleado", especialidad AS Servicio
FROM tb_empleado
ORDER BY nombre_empleado;
```

```
88  14:36:49  CREATE VIEW servicios_empleado AS SELECT CONCAT(nombre_empleado, " ",apellido_empleado) AS "Nombre del empleado", especialidad AS Ser...  0 row(s) affected
```

Una vez creada la vista, podremos visualizarla en la parte superior izquierda, en el panel de *Navigator* de Workbench, posteriormente presionar la opción *Views* para desplegar las vistas disponibles.



La información de las vistas puede ser accedida de la misma manera que se accede a la información de una tabla, utilizando la sentencia *SELECT* de la siguiente manera:

```
328 •  SELECT *
329    FROM servicios_empleado;
330
```

Result Grid | Filter Rows: | Export: | Wrap

Nombre del empleado	Servicio
Andera Armstrong	Central Specialist
Bobbi Mraz	Global Banking Orchestrator
Bonita Klocko	Design Executive
Britni Bechtelar	Government Coordinator
Claude Nader	Mining Designer
Daisey Wuckert	Principal Advertising Coordinator
Darius Mraz	Administration Analyst
Delbert Powlowski	Banking Officer
Dominique Pfannerstill	National Consulting Technician
Dong Kessler	International Hospitality Coordinator

servicios_empleado 68 ×

¿Qué es un procedimiento?

Es un conjunto de sentencias SQL, las cuales son agrupadas y almacenadas por un nombre particular en una BD relacional para ser usada y ejecutada en el momento que lo necesitemos donde su objetivo es realizar una tarea predeterminada como: consultas, insertar datos, actualizar, eliminar, hacer cálculos, entre otros.

Sintaxis de un procedimiento

```
CREATE PROCEDURE procedure_name
[ (parameter1 datatype [, parameter2 datatype, ...]) ]

BEGIN
   -- Cuerpo del procedimiento (aquí se incluye la lógica que se desea implementar)
END;
```

- Palabra clave **CREATE:** Especifica que se está creando un procedimiento almacenado, seguido de la palabra **PROCEDURE** y el **nombre** de dicho procedimiento.
- La sección de parámetros se especifican entre ( ) donde cada uno lleva un nombre y un tipo de dato.
- Palabra clave **BEGIN**: indica el inicio del cuerpo del procedimiento, el cual contiene la lógica especifica que se desea implementar dentro del procedimiento.
- Palabra clave **END**: indica el final del procedimiento.
- Dentro de la sección BEGIN y END se agrega el código SQL.
- En el cuerpo del procedimiento se utiliza **INSERT**, el cual nos está indicando que se va a insertar un nuevo.
- Utilizamos **CALL** para llamar a un procedimiento almacenado.
- Palabra clave **IN** antes del nombre del parámetro, nos indica que es un parámetro de entrada, es decir, que se puede pasar un valor al procedimiento para ser utilizado en su interior (cuerpo del procedimiento).
- La sentencia **DELIMITER** es útil cuando debemos definir un bloque de código que contenta varias sentencias de SQL como los procedimientos almacenados.

Creación de procedimientos

Procedimiento #1

Permite registrar un nuevo proveedor, el cual recibe los parámetros:

- id_proveedor_proc (El NIT del proveedor).
- nombre_proveedor_proc (Nombre del proveedor o empresa). direccion_proc (Dirección del proveedor).

```
DELIMITER //
CREATE PROCEDURE registrar_proveedor(
IN id_proveedor_proc VARCHAR(200),
IN nombre_proveedor_proc VARCHAR(200),
IN direccion_proc VARCHAR(200)
)
BEGIN
INSERT INTO tb_proveedor(id_proveedor, nombre_proveedor, direccion)
VALUES (id_proveedor_proc, nombre_proveedor_proc, direccion_proc);
END
//DELIMITER ;
```

Procedimiento #2

Permite registrar un nuevo producto, el cual recibe los parámetros:

- id_producto_proc → id del producto.
- nombre_producto_proc → nombre del producto.
- cantidad_disponible_proc → cantidad disponible del producto.
- precio_proc → precio del producto.
- id_proveedor_proc → id del proveedor de dicho producto.

```
DELIMITER //
CREATE PROCEDURE registrar_producto(
IN id_producto_proc VARCHAR(200),
IN nombre_producto_proc VARCHAR(200),
IN cantidad_disponible_proc INT,
IN precio_proc INT,
IN id_proveedor_proc VARCHAR(200)
)
BEGIN
INSERT INTO tb_producto(id_producto, nombre_producto, cantidad_disponible, precio, id_proveedor)
VALUES (id_producto_proc, nombre_producto_proc, cantidad_disponible_proc, precio_proc, id_proveedor_proc);
END
//DELIMITER ;
```

Procedimiento #3

Permite registrar un nuevo cliente, el cual recibe los parámetros:

- id_cliente_proc → id del cliente.
- nombre_cliente_proc → nombre del cliente.
- apellido_cliente_proc → apellido del cliente.
- fecha_nacimiento_proc → fecha de nacimiento del cliente.
- correo_proc → correo del cliente.
- direccion_proc → dirección del cliente.

*

```
DELIMITER //
CREATE PROCEDURE registrar_cliente(
IN id_cliente_proc VARCHAR(200),
IN nombre_cliente_proc VARCHAR(200),
IN apellido_cliente_proc VARCHAR(200),
IN fecha_nacimiento_proc VARCHAR(200),
IN correo_proc VARCHAR(200),
IN direccion_proc VARCHAR(200)
)
BEGIN
INSERT INTO tb_cliente(id_cliente, nombre_cliente, apellido_cliente, fecha_nacimiento, correo, direccion)
VALUES (id_cliente_proc, nombre_cliente_proc, apellido_cliente_proc, fecha_nacimiento_proc, correo_proc, direccion_proc);
END
//DELIMITER ;
```

Procedimiento #4

Permite registrar un nuevo empleado, recibe los siguientes parámetros:

- id_empleado_proc → Cédula del empleado
- nombre_empleado_proc → Primer nombre del empleado
- apellido_empleado_proc → Apellido del empleado
- especialidad_empleado_proc → Especialidad del empleado
- id_servicio_proc → Clave foránea con la tabla de servicios.

```
DELIMITER //
CREATE PROCEDURE registrar_empleado(
IN id_empleado_proc VARCHAR(200),
IN nombre_empleado_proc VARCHAR(200),
IN apellido_empleado_proc VARCHAR(200),
IN especialidad_proc VARCHAR(200),
IN id_servicio_proc VARCHAR(200)
)
BEGIN
INSERT INTO tb_empleado(id_empleado, nombre_empleado, apellido_empleado, especialidad, id_servicio)
VALUES (id_empleado_proc, nombre_empleado_proc, apellido_empleado_proc, especialidad_proc, id_servicio_proc);
END
//DELIMITER ;
```

¿Qué es un trigger?

También llamado disparador, es un objeto de base de datos el cual se activa automáticamente, dando respuesta a determinados eventos como: insertar, actualizar o eliminar datos en una tabla específica.

Sintaxis de un trigger

```
CREATE TRIGGER nombre_trigger

{BEFORE | AFTER | INSTEAD OF} {INSERT | UPDATE | DELETE}

ON nombre_tabla

[FOR EACH ROW]

[WHEN condicion]

BEGIN
-- cuerpo del trigger
END;
```

Adicional a las sentencias y conceptos mencionados anteriormente, también utilizaremos los siguientes:

- Palabra clave **CREATE:** Especifica que se está creando un trigger, seguido de la palabra **TRIGGER** y el **nombre** de dicho disparador.
- **{BEFORE | AFTER | INSTEAD OF}** nos permite especificar cuándo se activará el trigger.
- **{INSERT | UPDATE | DELETE}** sirve para especificar en qué tipo de operación se activará el trigger.
- **FOR EACH ROW** se utiliza para especificar que el cuerpo de un trigger se ejecute una vez para cada fila, lo cual permite realizar acciones específicas como actualizar, insertar o eliminar.

Creación de triggers

Trigger #1

Permite incrementar la cantidad disponible de los productos, cada que la barbería realiza una compra (al realizarse una nueva compra).

```
DELIMITER //
CREATE TRIGGER triggerCompras BEFORE INSERT ON dll_compra_producto
FOR EACH ROW
BEGIN
UPDATE tb_producto SET cantidad_disponible = cantidad_disponible + NEW.unidades
WHERE id_producto = NEW.id_producto;
END
//DELIMITER ;
```

```
228 •  INSERT INTO dll_venta_producto()
229    VALUES ('1232324', '1000000', '0000000001', 3, 7554);
230
```

Result Grid | Filter Rows: | Edit: | Export/Imp

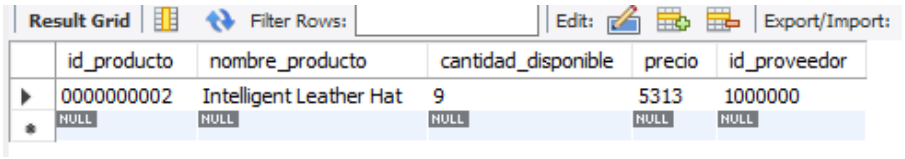
id_venta_producto	id_registro_venta	ref_producto	unidades	valor
1000046	1000044	0000000045	32	28610
1000047	1000045	0000000046	20	19425
1000048	1000046	0000000047	24	41826
1000049	1000047	0000000048	71	27252
1000050	1000048	0000000049	7	10570
1000051	1000049	0000000050	90	14829
1232324	1000000	0000000001	3	7554
NULL	NULL	NULL	NULL	NULL

dll_venta_producto3 ×

Trigger #2

Permite actualizar la cantidad disponible de los productos, cada que la barbería realiza una compra (al realizarse una nueva compra, se resta la cantidad de productos que se estén ingresando en la venta).

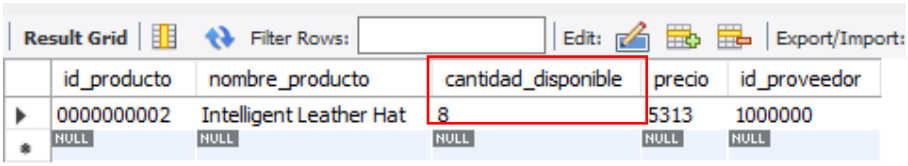
```
DELIMITER //
CREATE TRIGGER triggerVentas BEFORE INSERT ON dll_venta_producto
FOR EACH ROW
BEGIN
UPDATE tb_producto SET cantidad_disponible = cantidad_disponible - NEW.unidades
WHERE id_producto = NEW.ref_producto;
END
//DELIMITER ;
```



id_producto	nombre_producto	cantidad_disponible	precio	id_proveedor
0000000002	Intelligent Leather Hat	9	5313	1000000
NULL	NULL	NULL	NULL	NULL

```
INSERT INTO dll_venta_producto()
VALUES ('03413591', '1000001', '0000000002', 1, 23157);
```

Al ingresar un nuevo detalle de venta, se actualiza la cantidad de productos disponibles:



id_producto	nombre_producto	cantidad_disponible	precio	id_proveedor
0000000002	Intelligent Leather Hat	8	5313	1000000
NULL	NULL	NULL	NULL	NULL

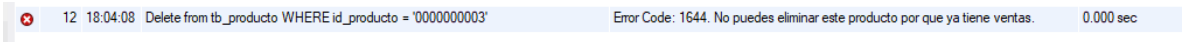
Trigger #3

Trigger que se ejecuta justo antes del intento de remover un producto, el trigger consulta el número de ventas asociadas al producto que se intenta remover y si el número es mayor que 0 el trigger muestra un mensaje de error y detiene el intento de remover el producto.

```
DELIMITER //
CREATE TRIGGER prevenir_remover_producto BEFORE DELETE ON tb_producto
FOR EACH ROW
BEGIN
    DECLARE veces_vendido INT; -- Declaramos una variable de tipo entero
    SELECT COUNT(*) INTO veces_vendido FROM dll_venta_producto WHERE ref_producto = OLD.id_producto;
    IF veces_vendido > 0 THEN
        -- La palabra reservada SIGNAL le dice a mysql que no continue con la ejecución y que
        -- arroje un error
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'No puedes eliminar este producto por que ya tiene ventas.';
    END IF;
END
//DELIMITER // ;
```

Ejemplo:

```
236 •  Delete from tb_producto WHERE id_producto = '0000000003';
```



12	18:04:08	Delete from tb_producto WHERE id_producto = '0000000003'	Error Code: 1644. No puedes eliminar este producto por que ya tiene ventas.	0.000 sec

Trigger #4

Trigger que se dispara antes de que se ejecute un nuevo registro en la tabla cita. Valida que el empleado para el que se intenta ingresar una nueva cita, no tenga ya citas para el mismo horario y fecha, si el empleado ya cuenta con citas para la hora y fecha ingresadas entonces el trigger muestra un error y no permite ejecutar la próxima consulta, que en este caso es intentar registrar una nueva cita.

```
DELIMITER //
CREATE TRIGGER prevenir_citas_dobles BEFORE INSERT ON tb_cita
FOR EACH ROW
BEGIN
    DECLARE citas_duplicadas INT;
    SELECT
        COUNT(*) INTO citas_duplicadas
        FROM tb_cita
        WHERE
            id_empleado = NEW.id_empleado AND
            fecha = NEW.fecha AND
            hora = NEW.hora;
    IF citas_duplicadas > 0 THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "La cita está duplicada";
    END IF;
END
//DELIMITER ;
```

Ejemplo:

```
257 •  INSERT INTO tb_cita(id_cita, fecha, hora, estado, id_cliente, id_empleado)
258        VALUES ('12012010', '08/09/1973', '3', 'activa', '1000004', '1000004');
259
```

⊗	23	18:26:26	INSERT INTO tb_cita(id_cita, fecha, hora,...	Error Code: 1644. La cita está duplicada

Conexión desde Java

1. Se implementa las dependencias en el archivo build.gradle

```
dependencies {
    testImplementation 'org.junit.jupiter:junit-jupiter-api:5.7.0'
    testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.7.0'
    implementation 'com.github.javafaker:javafaker:1.0.2'
    implementation 'mysql:mysql-connector-java:8.0.29'
}
```

2. Creamos una clase para la conexión a la base de datos "MySqlConnector"

```
package com.sofkau.integration.database;
import java.sql.*;

public class MySqlConnector {
    private String connectionString;
    private String dbName = "";
    private String dbUser = "";
    private String dbPassword = "";
    Connection connection;
    public MySqlConnector (String dbName, String dbUser, String dbPassword){...}
    public void connect(){...}
    public void closeConnection (){...}
    /** Permite crear consultas SELECT, INSERT, UPDATE, DELETE ...*/
    public PreparedStatement getStatement(String sql){...}
    /** Permite hacer llamadas a procedimientos almacenados ...*/
    public CallableStatement getCallable (String sql){...}
    public boolean insert(PreparedStatement statement){...}
    public boolean insert(CallableStatement procedure){...}
    public Connection getConnector() { return this.connection; }
}
```

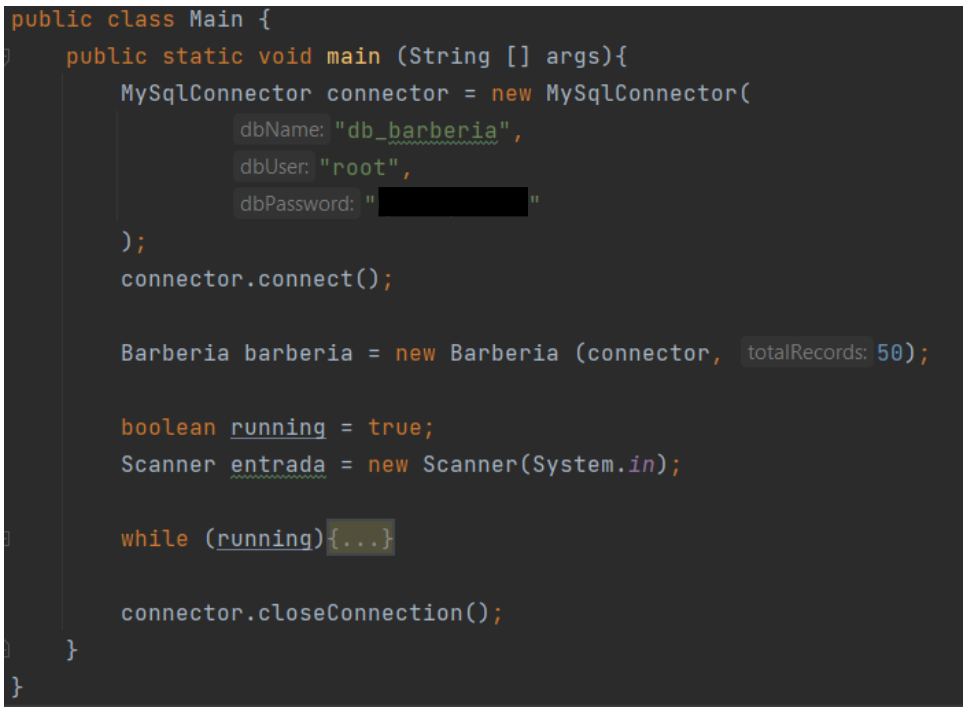
3. Se crea una *clase Barbería* donde se encuentran todos los métodos para poblar cada tabla de la BD.

```java
public class Barberia {

    MySqlConnector conn;
    private int totalRecords = 50;

    public Barberia(MySqlConnector conn, int totalRecords){...}
    public void llenarServicios (){...}
    public void llenarTelefonoProveedor(){...}
    public void llenarTelefonoCliente(){...}
    public void llenarCompra(){...}
    public void llenarCita(){...}
    public void llenarHistorialServicio(){...}
    public void llenarVenta(){...}
    public void llenarProveedores (){...}
    public void llenarProductos(){...}
    public void llenarClientes (){...}
    public void llenarEmpleados (){...}
}
```

De esta manera se insertan nuevos registros a una tabla, utilizando Java Faker.
Ejemplo de cómo se insertan los datos en la tabla teléfono proveedor.

```java
public void llenarTelefonoProveedor(){
    try{
        Faker faker = new Faker();

        String query = "INSERT INTO tb_telefono_proveedor (id_telefono_proveedor, telefono)" +
                "VALUES (?, ?);";
        PreparedStatement statement = this.conn.getStatement(query);
        int cc = 1000000;
        for (int i = 0; i < this.totalRecords; i ++){
            statement.setString( parameterIndex: 1,  x: (cc + i) + "");
            statement.setString( parameterIndex: 2, faker.phoneNumber().cellPhone());
            this.conn.insert(statement);
        }
        statement.close();
    }catch(SQLException e){
        e.printStackTrace();
    }
}
```

4. Se crea una clase main para ejecutar la aplicación, creando un menú para controlar que se puedan poblar las tablas una a una.



Para este proyecto se utilizó una librearía de Java llamada "Java Faker" la cual permite generar datos aleatorios para pruebas y simulaciones, al implementarla se identifican los siguientes aspectos:

- *Aunque hay gran variedad de registros dependiendo el tipo de dato, no se cuenta con métodos que contengan productos o especialidades con una barbería.*
- *Es fácil de usar y no requiere de mucha configuración.*
- *Ahorra tiempo a la hora de generar datos de manera automática.*

¿Está conforme con el resultado obtenido según el contexto o cree que hubiera obtenido un mejor resultado con una base de datos no relacional?

Sí, me siento conforme con el resultado ya que siento que cumple las expectativas, no considero que el uso de una base de datos haya influido en mejor o peor resultado, debido a que una BD relacional es suficiente para almacenar y gestionar la información requerida en esta actividad de la barbería. La diferencia entre una BD relacional y no relacional se puede apreciar mejor cuando hay volúmenes de información realmente altos, en este proyecto la cantidad de información era relativamente poca y en general la selección de una u otra depende mucho del contexto y de las necesidades de un sistema.