UNIVERSIDAD NACIONAL DE SAN ANTONIO ABAD DEL CUSCO

FACULTAD DE INGENIERIA ELECTRICA, ELECTRONICA, INFORMATICA Y MECANICA

ESCUELA PROFESIONAL DE INGENIERIA INFORMATICA Y DE SISTEMAS

New SQL

FUNDAMENTO Y DISEÑO DE BASE DE DATOS

DOCENTE: CARLOS RAMON QUISPE ONOFRE

ALUMNO: RONALDO TICONA JANCCO

CODIGO: 211816

CUSCO-PERU

2025

2.2. Parte práctica

2.2.1. Bases de datos con tablas semi-estructuradas

En muchos sistemas de información se tienen diseños forzados de tablas con información estructurada y semi-estructurada. Por ejemplo, en un sub sistema de almacenes de una empresa que comercializa productos de cómputo, la tabla artículo presenta información estructurada y semi-estructurada, tal la figura.

- Información estructurada.- Todos los artículos tienen los siguientes campos: Código, Descripción, Unidad Medida y Precio Unitario.
- Información semi-estructurada.- Algunos artículos tienen datos asociados a marca y modelo, otros a la capacidad del disco, memoria, tamaño de pantalla o fecha de vencimiento.

ARTÍCULOS											
Código	Descripción	Unidad Medida	Precio Unitario	Disco	Marca	Modelo	Memoria	Pantalla	Tipo	Fecha Vcto	Gramaje
101	Computador	UNI	1500	1TB	HP	XYZ	12GB				
121	Laptop	UNI	2700	1TB	LENOVO	ZetaXYZ	12GB	12"			
142	Impresora	UNI	780		EPSON				Laser		
154	Tinta	PZA	120		EPSON					31/12/2023	
155	Papel	MIL	20								60Gr

Este diseño es forzado, porque se deben considerar todos los atributos que tienen todos los artículos; luego, si a un artículo no le corresponde un determinado atributo, éste es registrado con nulo. En consecuencia, se tiene una cantidad de nulos correspondientes a los atributos que no corresponden a determinados artículos. El mayor problema de este diseño, es la flexibilidad; es decir, si se desease considerar un nuevo atributo para ciertos artículos (Por ejemplo, el número de serie), se tendría que reestructurar la tabla, con todos los inconvenientes que este hecho implicaría.

Una alternativa de solución son los denominados diseños “verticales”, que considera dos tablas, una para la parte estructurada y la otra para la parte semi-estructurada, tal la siguiente figura:

ARTÍCULOS			
Código	Descripción	Unidad Medida	Precio Unitario
101	Computador	UNI	1500
121	Laptop	UNI	2700
142	Impresora	UNI	780
154	Tinta	PZA	120
155	Papel	MIL	20

ARTÍCULOS_ATRIBUTOS		
Código	Atributo	Valor
101	Disco	1TB
101	Marca	HP
101	Modelo	XYZ
101	Memoria	12GB
121	Disco	1TB
121	Marca	LENOVO
121	Modelo	ZetaXYZ
121	Memoria	12GB
121	Pantalla	12"
142	Marca	EPSON
142	Tipo	Laser
154	Marca	EPSON
154	Fecha Vcto	31/12/2023
155	Gramaje	60gr

Si bien es una alternativa que evita el uso de valores nulos, presenta cierto nivel de complejidad en la definición de consultas y procesamiento de los datos.

La alternativa más coherente para este tipo de contextos, es el uso de SGBD que soporten también almacenar tuplas con información semi-estructurada, atributos de tipo convencional para la parte estructurada y atributos de tipo JSON para la información semi-estructurada; lo que permitiría tablas como el de la siguiente figura.

ARTÍCULO				
Código	Descripción	Unidad Medida	Precio Unitario	Datos
101	Computador	UNI	1500	{"HD": "1TB", "Marca": "HP", "Modelo": "XYZ", "Memoria": "12GB"}
121	Laptop	UNI	2700	{"HD": "1TB", "Marca": "LENOVO", "Modelo": "ZetaXYZ", "Memoria": "12GB", "Pantalla": "12"}
142	Impresora	UNI	780	{"Tipo": "Laser", "Marca": "EPSON", "Modelo": "L900"}
154	Tinta	PZA	120	{"Marca": "EPSON", "FechaVcto": "31/12/2023"}
155	Papel	MIL	20	{"Gramaje": "60gr"}

Para ilustrar este hecho, se realizará la parte práctica con el SGBD MySQL. Para lo cual, se creará una base de datos "dbventas" y como parte de ésta, la tabla "producto". Luego, se mostrará como ingresar datos estructurados y semi-estructurados; para finalmente, efectuar las consultas sobre datos de ambos tipos de estructuras.

Para crear y activar la base de datos, efectuar lo siguiente:

```
-- -----------------------------------------------------------------
--                          DBVentas
-- -----------------------------------------------------------------

CREATE database IF NOT EXISTS `dbventas` DEFAULT CHARACTER SET utf8 ;
-- -----------------------------------------------------
-- Schema dbventas
-- -----------------------------------------------------
USE `dbventas`;
```

Para crear la tabla, efectuar lo siguiente:

```
-- -----------------------------------------------------
-- Table `dbventas`.`producto`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `dbventas`.`producto` (
  `idProducto` INT NOT NULL,
  `Descripcion` VARCHAR(255) NOT NULL,
  `UnidadMedida` VARCHAR(255) NOT NULL,
  `Precio` NUMERIC(15,2) NOT NULL,
  `Datos` JSON NOT NULL,
  PRIMARY KEY (`idProducto`)
);
```

Para insertar datos a la tabla, se procede de manera convencional para los atributos estructurados; mientras que para los atributos semi-estructurados se hace uso del comando JSON_OBJECT, que permite estructurar datos en formato JSON. Efectuar lo siguiente;

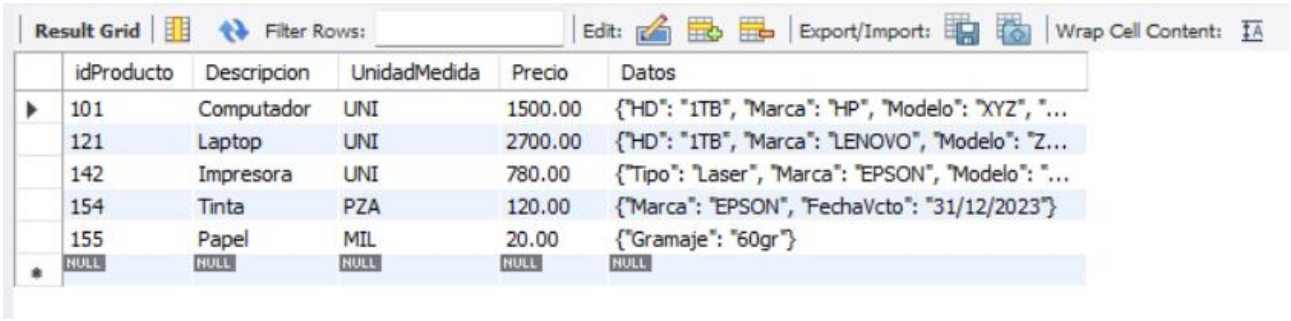
```
-- Insertar datos a la tabla producto
insert into `dbventas`.`producto`
values(101,'Computador','UNI',1500,JSON_OBJECT('Marca', 'HP',
                                               'Modelo', 'XYZ',
                                               'Memoria','12GB',
                                               'HD','1TB')),
      (121,'Laptop','UNI',2700,JSON_OBJECT('Marca', 'LENOVO',
                                           'Modelo', 'ZetaXYZ',
                                           'Memoria','12GB',
                                           'HD','1TB',
                                           'Pantalla','12')),
      (142,'Impresora','UNI',780,JSON_OBJECT('Marca', 'EPSON',
                                             'Modelo', 'L900',
                                             'Tipo','Laser')),
      (154,'Tinta','PZA',120,JSON_OBJECT('Marca', 'EPSON',
                                         'FechaVcto', '31/12/2023')),
      (155,'Papel','MIL',20,JSON_OBJECT('Gramaje', '60gr'));
```

Luego, para efectuar los procesos de consulta se procede de manera similar a las sentencias de SQL.

Por ejemplo, para consultar el contenido de la tabla producto, efectuar lo siguiente:

```
-- Seleccionar datos
select *
  from producto;
```

Se mostrará el siguiente resultado:



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

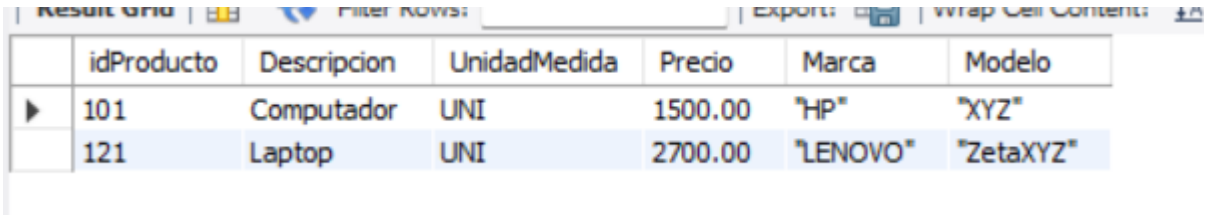
idProducto	Descripcion	UnidadMedida	Precio	Datos
101	Computador	UNI	1500.00	{"HD": "1TB", "Marca": "HP", "Modelo": "XYZ", "...
121	Laptop	UNI	2700.00	{"HD": "1TB", "Marca": "LENOVO", "Modelo": "Z...
142	Impresora	UNI	780.00	{"Tipo": "Laser", "Marca": "EPSON", "Modelo": "...
154	Tinta	PZA	120.00	{"Marca": "EPSON", "FechaVcto": "31/12/2023"}
155	Papel	MIL	20.00	{"Gramaje": "60gr"}
NULL	NULL	NULL	NULL	NULL

Como se puede apreciar, la información semi-estructurada se muestra en formato JSON.

Si se desea obtener información de la columna en formato JSON, se debe hacer uso de las funciones que ofrece el gestor para tal propósito. Por ejemplo, si sólo se desea obtener la información de los productos que tengan el atributo "HD" con el valor igual a 1TB, y mostrar las columnas de marca y modelo; entonces digitar lo siguiente:

```
-- Uso de funciones JSON
select idProducto,Descripcion,UnidadMedida,Precio, JSON_EXTRACT(Datos, '$.Marca') AS Marca, JSON_EXTRACT(Datos, '$.Modelo') AS Modelo
  from producto
  where JSON_EXTRACT(Datos, '$.HD') = '1TB';
```

Se mostrará el siguiente resultado:



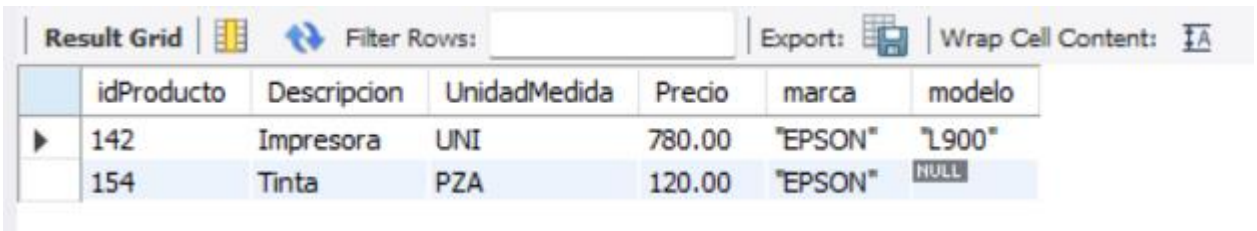
idProducto	Descripcion	UnidadMedida	Precio	Marca	Modelo
101	Computador	UNI	1500.00	"HP"	"XYZ"
121	Laptop	UNI	2700.00	"LENOVO"	"ZetaXYZ"

Notar que se está utilizando una sentencia SQL convencional, que incluye la función JSON_EXTRACT para extraer un atributo de la columna que ha sido definida como semi-estructurada.

Para considerar otras características de procesar consultas sobre la columna en formato JSON, digitar lo siguiente:

```
select idProducto,Descripcion,UnidadMedida,Precio, Datos->'$.Marca' AS marca, Datos->'$.Modelo' AS modelo
  from producto
  where (Datos->'$.Marca' is not null) and (Datos->'$.Marca' = 'EPSON');
```

Se mostrará el siguiente resultado:



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

idProducto	Descripcion	UnidadMedida	Precio	marca	modelo
142	Impresora	UNI	780.00	"EPSON"	"L900"
154	Tinta	PZA	120.00	"EPSON"	NULL

En este caso se está utilizando el operador -> que es una versión simplificada de la función JSON_EXTRACT.

2.2.2. Bases de datos con registros de auditoría

Un aspecto muy importante en la administración de bases de datos, son los procesos de auditoría, que permita identificar a los responsables de la manipulación sobre todo de datos sensibles.

Una de las aplicaciones relevantes de los "triggers" es la implementación de mecanismos que permitan efectuar procesos de auditoría. En ese sentido, a continuación se ilustrará con un ejemplo sencillo la implementación de un mecanismo de auditoría que hace uso de "triggers" e información semi-estructurada en formato JSON.

Para tal efecto, efectuar lo siguiente:

Crear la base de datos "dbcont" y como parte de ésta, la tabla "cuenta"; para lo cual digitar lo siguiente:

```
-- -----------------------------------------------------------------
--                            DBCont
-- -----------------------------------------------------------------
CREATE database IF NOT EXISTS `dbcont` DEFAULT CHARACTER SET utf8 ;
-- -----------------------------------------------------
-- Schema DBcont
-- -----------------------------------------------------
USE `dbcont`;

-- -----------------------------------------------------
-- Table `dbcont`.`cuenta`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `dbcont`.`cuenta` (
  `idCuenta` INT NOT NULL,
  `Descripcion` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`idCuenta`));
```

Crear también la base de datos "dbalmacen" y activarla; para lo cual digitar lo siguiente:

```
-- -----------------------------------------------------------------
--                            DBAlmacen
-- -----------------------------------------------------------------

CREATE database IF NOT EXISTS `dbalmacen` DEFAULT CHARACTER SET utf8 ;
-- -----------------------------------------------------
-- Schema DBAlmacen
-- -----------------------------------------------------
USE `dbalmacen`;
```

Luego, digitar las sentencias para crear las tablas de esta base de datos.

```
-- -----------------------------------------------------
-- Table `dbalmacen`.`categoria`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `dbalmacen`.`categoria` (
  `idCategoria` INT NOT NULL,
  `Descripcion` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`idCategoria`));

-- -----------------------------------------------------
-- Table `dbalmacen`.`UnidadMedida`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `dbalmacen`.`UnidadMedida` (
  `idUnidadMedida` INT NOT NULL,
  `Descripcion` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`idUnidadMedida`));
```

```
-- -----------------------------------------------------
-- Table `dbalmacen`.`producto`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `dbalmacen`.`producto` (
  `idProducto` INT NOT NULL,
  `Descripcion` VARCHAR(255) NOT NULL,
  `idCategoria` INT NULL,
  `idUnidadMedida` INT NOT NULL,
  `idCuenta` INT NOT NULL,
  PRIMARY KEY (`idProducto`),
  CONSTRAINT `fk_producto_categoria` FOREIGN KEY (`idCategoria`) REFERENCES `dbalmacen`.`categoria` (`idCategoria`),
  CONSTRAINT `fk_producto_UnidadMedida` FOREIGN KEY (`idUnidadMedida`) REFERENCES `dbalmacen`.`UnidadMedida` (`idUnidadMedida`),
  CONSTRAINT `fk_producto_Cuenta` FOREIGN KEY (`idCuenta`) REFERENCES `dbCont`.`Cuenta` (`idCuenta`)

);
-- -----------------------------------------------------
-- Table `dbalmacen`.`producto`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `dbalmacen`.`log` (
  `idLog` INT NOT NULL AUTO_INCREMENT,
  `Tabla` VARCHAR(255) NOT NULL,
  `Operacion` VARCHAR(255) NOT NULL,
  `Usuario` VARCHAR(255) NOT NULL,
  `Fecha` DateTime NOT NULL,
  `IP` VARCHAR(255) NOT NULL,
  `Datos` JSON NOT NULL,
  PRIMARY KEY (`idLog`)
);
```

En esta última tabla se registrarán todos los eventos correspondientes a las operaciones INSERT, UPDATE y DELETE efectuadas sobre cualquier tabla de ambas bases de datos. Se almacenará la información relevante de cada evento, incluyendo la tabla afectada, la operación realizada, el usuario que la llevó a cabo, la fecha y hora del evento, la dirección IP de la máquina desde donde se efectuó la operación, y los datos asociados a dicha operación. Los datos relacionados con la operación se almacenarán en formato JSON, proporcionando la flexibilidad necesaria para guardar información de cualquier tabla.

Para este propósito se debe implementar diferentes módulos.

Primero, se debe implementar un procedimiento almacenado cuya funcionalidad será registrar en la tabla “log” los datos asociados a un evento. El script correspondiente es:

```
-- ****************************************************************
-- Procedimiento para almacenar en la tabla `dbalmacen`.`Log`
-- ****************************************************************

delimiter //
CREATE PROCEDURE `dbalmacen`.`spu_Log`(Tabla varchar(255), Operacion varchar(7), Datos JSON)
BEGIN
  -- Recuperar variables de control
  set @Usuario = (SELECT LEFT(CURRENT_USER(), INSTR(CURRENT_USER(), '@') - 1));
  set @Fecha = (SELECT Now());
  set @IP = (select SUBSTRING_INDEX(host,':',1) as 'ip' from information_schema.processlist WHERE ID=connection_id());
  -- Insertar registro en la tabla Log
  INSERT INTO log(Tabla, Operacion, Usuario, Fecha, IP, Datos)
    VALUES(Tabla, Operacion, @Usuario, @Fecha, @IP, Datos);
END//
delimiter ;
```

Segundo, se debe implementar “triggers” en las tablas cuyo seguimiento se desea almacenar en la tabla “log”. Cada disparador deberá invocar al procedimiento anterior, pasando como parámetros: la Tabla que genera el evento, la operación (INSERT, UPDATE o DELETE) y los datos asociados al evento en formato JSON. Los scripts de los diferentes disparadores que se deben ejecutar son:

Disparador para la operación INSERT sobre la tabla “Cuenta” de la base de datos “dbcont”.

```
-- ****************************************************************
--       Disparadores para DBCont
-- ****************************************************************
-- ----------------------------------------------------------------
-- Disparadores para almacenar en el log, operaciones de cuenta
-- ----------------------------------------------------------------
-- INSERT
delimiter //
CREATE TRIGGER `dbcont`.`tr_Log_Cuenta_Insert` AFTER INSERT ON `dbcont`.`Cuenta`
FOR EACH ROW
BEGIN
  -- recuperar atributos y valores de la tabla de Producto
  set @Texto = JSON_OBJECT('idCuenta', NEW.idCuenta,
                           'Descripcion', NEW.Descripcion);
  -- Ejecutar procedimiento que registra operación en el log
  CALL `dbalmacen`.`spu_Log`('dbcont.cuenta', 'INSERT', @Texto);
END;//
delimiter ;
```

Disparador para la operación INSERT sobre la tabla “Producto” de la base de datos “dbalmacen”.

```
-- ****************************************************************
--               Disparadores para DBAlmacen
-- ****************************************************************
-- ------------------------------------------------------
-- Disparadores para almacenar en el log de producto
-- ------------------------------------------------------
-- INSERT
delimiter //
CREATE TRIGGER `dbalmacen`.`tr_Log_Producto_Insert` AFTER INSERT ON producto
FOR EACH ROW
BEGIN
  -- recuperar atributos y valores de la tabla de Producto
  set @Texto = JSON_OBJECT('idProducto', NEW.idProducto,
                           'Descripcion', NEW.Descripcion,
                           'idCategoria', NEW.idCategoria,
                           'idUnidadMedida',NEW.idUnidadMedida,
                           'idCuenta', NEW.idCuenta);
  -- Ejecutar procedimiento que registra operación en el log
  CALL `dbalmacen`.`spu_Log`('dbalmacen.producto', 'INSERT', @Texto);
END;//
delimiter ;
```

Disparador para la operación UPDATE sobre la tabla “producto” de la base de datos “dbalmacen”.

```
-- UPDATE

delimiter //
CREATE TRIGGER `dbalmacen`.`tr_Log_Producto_UpDate` AFTER UPDATE ON producto
FOR EACH ROW
BEGIN
  -- recuperar atributos y valores de la tabla de Producto
  set @Texto = JSON_OBJECT('idProducto', OLD.idProducto,
                           'Descripcion', OLD.Descripcion,
                           'idCategoria', OLD.idCategoria,
                           'idUnidadMedida',OLD.idUnidadMedida,
                           'idCuenta', OLD.idCuenta);
  -- Ejecutar procedimiento que registra operación en el log
  CALL `dbalmacen`.`spu_Log`('dbalmacen.producto', 'UPDATE', @Texto);
END;//
delimiter ;
```

Disparador para la operación DELETE sobre la tabla “producto” de la base de datos “dbalmacen”.

```
-- DELETE

delimiter //
CREATE TRIGGER `dbalmacen`.`tr_Log_Producto_Delete` AFTER DELETE ON producto
FOR EACH ROW
BEGIN
  set @Texto = JSON_OBJECT('idProducto', OLD.idProducto,
                           'Descripcion', OLD.Descripcion,
                           'idCategoria', OLD.idCategoria,
                           'idUnidadMedida',OLD.idUnidadMedida,
                           'idCuenta', OLD.idCuenta);
  -- Ejecutar procedimiento que registra operación en el log
  CALL `dbalmacen`.`spu_Log`('dbalmacen.producto', 'DELETE', @Texto);
END;//
delimiter ;
```

Tercero, insertar datos en las tablas de ambas bases de datos, ejecutando los siguientes scripts:

```
-- -----------------------------------------------------------------
--                         DBCont
-- -----------------------------------------------------------------
use `dbcont`;
-- Insertar datos a la tabla categoria
insert into `dbcont`.`cuenta`
values(601,'Mercaderia'), (6011,'Mercaderia tipo 1'), (6012,'Mercaderia tipo 1');

-- -----------------------------------------------------------------
--                         DBAlmacen
-- -----------------------------------------------------------------
use `dbalmacen`;
-- Insertar datos a la tabla categoria
insert into `categoria`
values(11,'Computadoras'), (15,'Laptop'), (23,'Impresoras');
-- Insertar datos a la tabla unidadmedida
insert into `unidadmedida`
values(91,'Pieza'), (92,'Unidad'), (93,'Caja');
-- Insertar datos a la tabla producto
insert into `dbalmacen`.`producto`
values(101,'Computador HP',11,92,6011),
      (105,'Coputador IBM',11,92,6012),
      (121,'Laptop Lenovo',15,92,6011),
      (142,'Impresora Epson',23,92,6012);
```

Cuarto, verificar el contenido de la tabla “log”, ejecutando el siguiente script:

```
-- Revisar la tabla Log
SELECT *
  FROM dbalmacen.log;
```

Se mostrará algo similar a:

idLog	Tabla	Operacion	Usuario	Fecha	IP	Datos
1	dbcont.cuenta	INSERT	root	2025-07-15 00:34:13	localhost	{"idCuenta": 601, "Descripcion": "Mercaderia"}
2	dbcont.cuenta	INSERT	root	2025-07-15 00:34:13	localhost	{"idCuenta": 6011, "Descripcion": "Mercaderia tipo 1"}
3	dbcont.cuenta	INSERT	root	2025-07-15 00:34:13	localhost	{"idCuenta": 6012, "Descripcion": "Mercaderia tipo 1"}
4	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6011, "idProducto": 101, "Descripcion": "Computador HP", "idCategoria": 11, "idUnidadMedida": 92}
5	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6012, "idProducto": 105, "Descripcion": "Coputador IBM", "idCategoria": 11, "idUnidadMedida": 92}
6	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6011, "idProducto": 121, "Descripcion": "Laptop Lenovo", "idCategoria": 15, "idUnidadMedida": 92}
7	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6012, "idProducto": 142, "Descripcion": "Impresora Epson", "idCategoria": 23, "idUnidadMedida": 92}

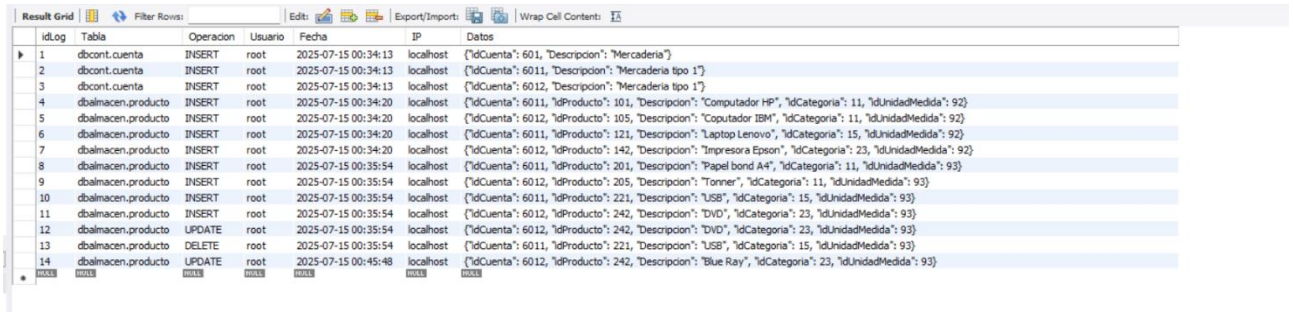
Quinto, agregar más eventos a la tabla “producto” de la base de datos “dbalmacen”, mediante la ejecución del siguiente script:

```
-- Insertar mas datos a la tabla producto
insert into `dbalmacen`.`producto`
values(201,'Papel bond A4',11,93,6011),
      (205,'Tonner',11,93,6012),
      (221,'USB',15,93,6011),
      (242,'DVD',23,93,6012);

-- Actualizar datos a la tabla producto
update `dbalmacen`.`producto`
  set Descripcion = 'Blue Ray'
  where idProducto = '242';

-- Eliminar datos de la tabla producto
delete from `dbalmacen`.`producto`
  where idProducto = '221';
```

Sexto, verificar el contenido de la tabla “log”, ejecutando el siguiente script:

```
-- Revisar la tabla Log
SELECT *
  FROM dbalmacen.log;
```

Se mostrará algo similar a:



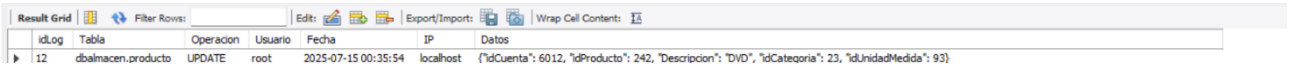
idLog	Tabla	Operacion	Usuario	Fecha	IP	Datos
1	dbcont.cuenta	INSERT	root	2025-07-15 00:34:13	localhost	{"idCuenta": 601, "Descripcion": "Mercaderia"}
2	dbcont.cuenta	INSERT	root	2025-07-15 00:34:13	localhost	{"idCuenta": 6011, "Descripcion": "Mercaderia tipo 1"}
3	dbcont.cuenta	INSERT	root	2025-07-15 00:34:13	localhost	{"idCuenta": 6012, "Descripcion": "Mercaderia tipo 1"}
4	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6011, "idProducto": 101, "Descripcion": "Computador HP", "idCategoria": 11, "idUnidadMedida": 92}
5	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6012, "idProducto": 105, "Descripcion": "Coputador IBM", "idCategoria": 11, "idUnidadMedida": 92}
6	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6011, "idProducto": 121, "Descripcion": "Laptop Lenovo", "idCategoria": 15, "idUnidadMedida": 92}
7	dbalmacen.producto	INSERT	root	2025-07-15 00:34:20	localhost	{"idCuenta": 6012, "idProducto": 142, "Descripcion": "Impresora Epson", "idCategoria": 23, "idUnidadMedida": 92}
8	dbalmacen.producto	INSERT	root	2025-07-15 00:35:54	localhost	{"idCuenta": 6011, "idProducto": 201, "Descripcion": "Papel bond A4", "idCategoria": 11, "idUnidadMedida": 93}
9	dbalmacen.producto	INSERT	root	2025-07-15 00:35:54	localhost	{"idCuenta": 6012, "idProducto": 205, "Descripcion": "Tonner", "idCategoria": 11, "idUnidadMedida": 93}
10	dbalmacen.producto	INSERT	root	2025-07-15 00:35:54	localhost	{"idCuenta": 6011, "idProducto": 221, "Descripcion": "USB", "idCategoria": 15, "idUnidadMedida": 93}
11	dbalmacen.producto	INSERT	root	2025-07-15 00:35:54	localhost	{"idCuenta": 6012, "idProducto": 242, "Descripcion": "DVD", "idCategoria": 23, "idUnidadMedida": 93}
12	dbalmacen.producto	UPDATE	root	2025-07-15 00:35:54	localhost	{"idCuenta": 6012, "idProducto": 242, "Descripcion": "DVD", "idCategoria": 23, "idUnidadMedida": 93}
13	dbalmacen.producto	DELETE	root	2025-07-15 00:35:54	localhost	{"idCuenta": 6011, "idProducto": 221, "Descripcion": "USB", "idCategoria": 15, "idUnidadMedida": 93}
14	dbalmacen.producto	UPDATE	root	2025-07-15 00:45:48	localhost	{"idCuenta": 6012, "idProducto": 242, "Descripcion": "Blue Ray", "idCategoria": 23, "idUnidadMedida": 93}

Notar, que se muestra también la información asociada a los eventos UPDATE y DELETE.

Séptimo, efectuar procesos de auditoria; por ejemplo, verificando que actualizaciones se hizo en las tablas de las bases de datos. Para este propósito ejecutar el siguiente script:

```sql
-- Recuperar las actualizaciones realizadas
SELECT *
  FROM dbalmacen.log
  WHERE Operacion = 'UPDATE';
```

Se mostrará sólo las tuplas asociadas al evento UPDATE, algo similar a:



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

idLog	Tabla	Operacion	Usuario	Fecha	IP	Datos
12	dbalmacen.producto	UPDATE	root	2025-07-15 00:35:54	localhost	{"idCuenta": 6012, "idProducto": 242, "Descripcion": "DVD", "idCategoria": 23, "idUnidadMedida": 93}

De esta forma se puede efectuar procesos de auditoría, consultando sobre información específica de la tabla “log”; por ejemplo, quién manipuló una determinada tabla, qué datos se manipuló, en qué fecha y desde qué IP se realizaron dichas manipulaciones, etc.

2.3. Investigación formativa

Elaborar una monografía sucinta de cómo almacenar información semi-estructurada en el SGBD SQL Server. Considerar, que tipo de columna se debe utilizar para este propósito y qué funciones JSON existen.