**INFORME DESARROLLO DE BASE DE DATOS**

**Autores:**

Javier Nicolas Salas

Yesi Esteban Pantoja

**Entregado a**:

Ing. Brayan Arcos

**Fecha:** 25/10/2024

[**INFORME DESARROLLO DE BASE DE DATOS** 1](#_Toc180701787)

[**Repositorio De GitHub:** 2](#_Toc180701788)

[**Resumen Ejecutivo** 2](#_Toc180701789)

[**Introducción** 3](#_Toc180701790)

[**Contexto y Motivación** 3](#_Toc180701791)

[**Alcance del Informe** 3](#_Toc180701792)

[**Objetivos** 3](#_Toc180701793)

[**Metodología** 3](#_Toc180701794)

[**Herramientas Utilizadas** 3](#_Toc180701795)

[**Procedimientos** 3](#_Toc180701796)

[**Desarrollo del Informe** 4](#_Toc180701797)

[**Descripción de la Base de Datos** 4](#_Toc180701798)

[**Tablas** 4](#_Toc180701799)

[**Diseño de Base de Datos** 6](#_Toc180701800)

[**Análisis y Discusión** 19](#_Toc180701801)

[**Interpretación de Resultados** 19](#_Toc180701802)

[**Conclusiones** 20](#_Toc180701803)

[**Recomendaciones** 20](#_Toc180701804)

[**Referencias** 20](#_Toc180701805)

**Repositorio De GitHub:**

Repositorio documentado con su estructura https://github.com/Nick0oo/DBD\_Proyecto

**Resumen Ejecutivo**

Este informe ofrece una visión detallada del proceso de desarrollo de una base de datos destinada a la gestión integral de un complejo deportivo. En él, se aborda el diseño de la base de datos, la estructura de las tablas, las consultas realizadas y el proceso de identificación de entidades fuertes y débiles. Además, se incluye el modelo entidad-relación y el proceso de normalización, proporcionando una explicación lógica y coherente de cada etapa del desarrollo realizado. Este documento busca servir como una guía clara y comprensible para quienes deseen entender el funcionamiento y la organización de la base de datos implementada.

**Introducción**

**Contexto y Motivación**

El principal motivador para nuestro equipo es el avance en el desarrollo de la base de datos de un proyecto personal que estamos llevando a cabo. Nuestro objetivo es aplicar los conocimientos adquiridos y crear un trabajo de alta calidad que nos permita avanzar de manera significativa en nuestros proyectos. Este proceso no solo nos brinda la oportunidad de consolidar nuestras habilidades técnicas, sino que también nos inspira a superar desafíos y a innovar en la gestión de datos para alcanzar resultados óptimos.**Alcance del Informe**

El informe cubre los siguientes aspectos de SQL:

* Creación y diseño de la base de datos.
* Consultas SQL.
* Entidades, Atributos, Normalización etc.
* Modelo entidad relacion.

**Objetivos**

El desarrollo y la documentación de todos los procesos realizados durante el desarrollo de nuestras bases de datos tienen como objetivo principal garantizar que estas sean completas y funcionales para el sistema que estamos desarrollando. A lo largo de esta etapa, nos hemos enfocado en crear una base de datos que no solo satisfaga los requisitos actuales, sino que también sea escalable y adaptable a futuras necesidades.

Hemos llevado a cabo un análisis exhaustivo de los requerimientos del sistema, seguido de la implementación de las tablas y relaciones necesarias para optimizar el almacenamiento y la recuperación de datos. Cada etapa del proceso ha sido meticulosamente documentada para asegurar que el sistema sea fácil de entender y mantener a largo plazo.

Además, hemos adoptado prácticas recomendadas en diseño y gestión de bases de datos, priorizando la integridad de los datos y la eficiencia en las consultas. Este enfoque nos permitirá no solo cumplir con los objetivos inmediatos del proyecto, sino también proporcionar una base sólida para el desarrollo de futuras funcionalidades y mejoras.

**Metodología**

**Herramientas Utilizadas**

* **Sistema de Gestión de Bases de Datos (DBMS):** MySQL
* **Editor de SQL:** MySQL Workbench o similar

**Procedimientos**

1. **Creación de la Base de Datos:** Se construyeron las tablas necesarias con sus relaciones y claves principales.
2. **Inserción de Datos:** Se insertaron datos de ejemplo en las tablas para proporcionar un contexto realista.
3. **Procesos:** documentar procesos de simbolización, cardinalidad, normalización
4. **Modelo entidad relación:** desarrollo del modelo entidad relacion

**Desarrollo del Informe**

**Descripción de la Base de Datos**

Esta base de datos está diseñada para la gestión integral y eficiente de un complejo deportivo a través de una aplicación móvil. Su estructura está cuidadosamente elaborada para abarcar diversos aspectos clave de la administración del complejo, facilitando un manejo detallado y preciso de las operaciones diarias.

La base de datos incluye una serie de tablas interrelacionadas que cubren múltiples facetas del funcionamiento útil y necesario para la gestión adecuada y eficaz por lo que tiene mucho contenido en el cual profundizalemos ahora

**Esquema de la Base de Datos**

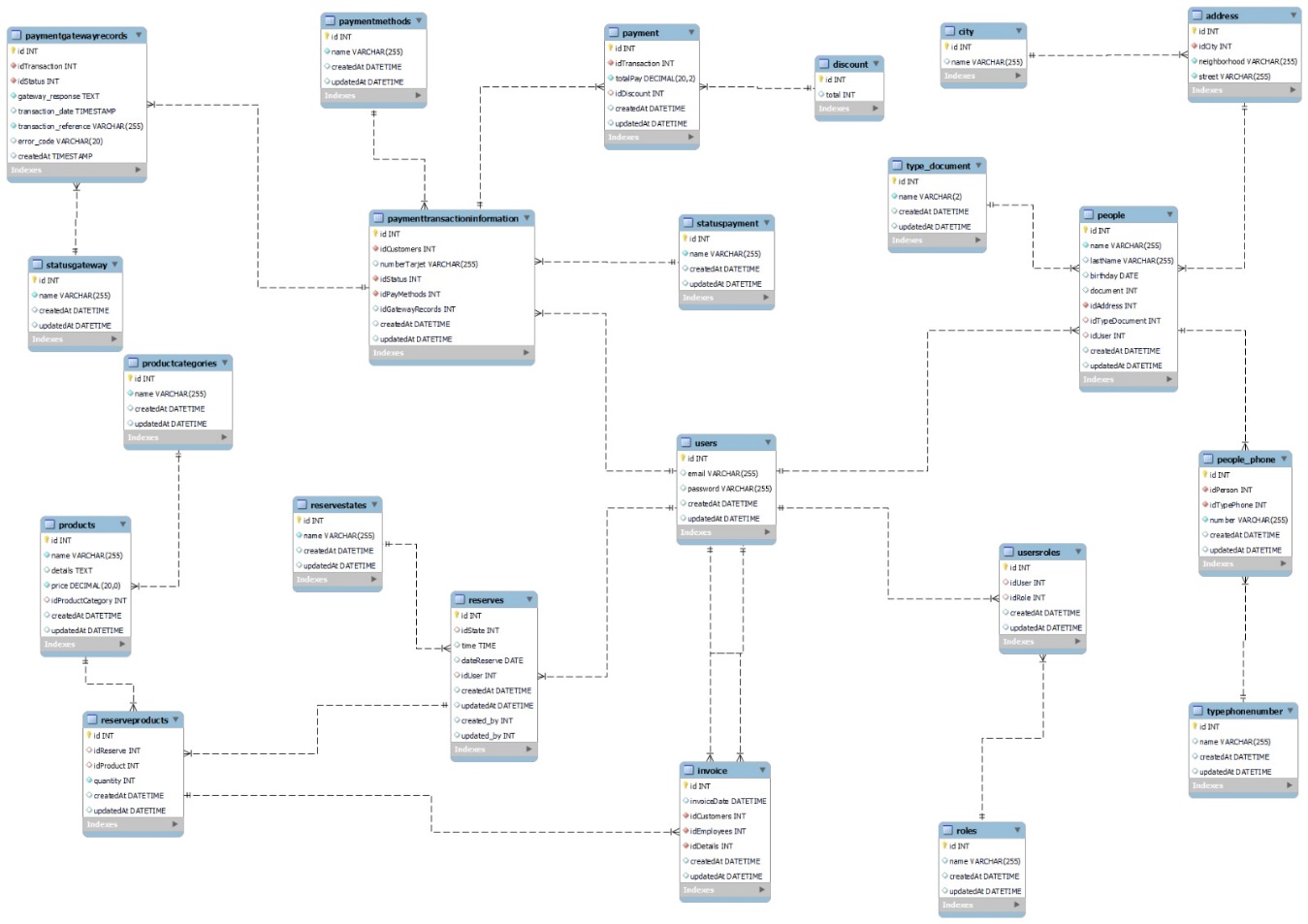
La base de datos **"proyecto"** está diseñada para gestionar la información relacionada con un **complejo deportivo** que ofrece una amplia variedad de servicios, productos y reservas. La estructura de las tablas permite gestionar tanto la información de clientes y empleados, como los productos disponibles, las reservas realizadas y las transacciones de pago. A continuación, se describen las principales tablas:

### **Tablas**

1. **type\_document:**  
   Almacena los tipos de documentos de identificación. Cada registro contiene un identificador único y una abreviatura del tipo de documento (por ejemplo, CC, PP, CE).
2. **users:**  
   Contiene los datos de los usuarios que interactúan con el sistema, como el correo electrónico y la contraseña. Se utiliza para gestionar los roles de acceso y controlar quién crea o actualiza la información.
3. **roles:**  
   Define los roles de usuario (por ejemplo, Administrador, Cliente) con un identificador único y un nombre. Cada usuario puede tener uno o más roles asignados para determinar sus permisos dentro del sistema.
4. **typePhoneNumber:**  
   Registra los tipos de números telefónicos, como Teléfono fijo o Móvil.
5. **city:**  
   Almacena las ciudades que forman parte del registro de direcciones. Cada ciudad tiene un identificador único y un nombre.
6. **address:**  
   Contiene las direcciones asociadas a clientes o empleados, incluyendo la ciudad, el barrio y la calle.
7. **people:**  
   Registra los datos personales de los individuos (clientes o empleados). Esta tabla incluye información como nombre, apellido, documento de identidad, dirección asociada y su relación con el usuario del sistema.
8. **people\_phone:**  
   Registra los números de teléfono asociados a personas. Cada registro incluye el número telefónico y el tipo de teléfono.
9. **usersRoles:**  
   Establece la relación entre usuarios y roles. Un usuario puede tener varios roles asignados en esta tabla.
10. **productCategories:**  
    Define las categorías de los productos disponibles en el sistema (por ejemplo, Ropa, Equipamiento). Cada categoría tiene un identificador único y un nombre.
11. **products:**  
    Registra los productos disponibles para reserva o venta. Cada producto tiene un nombre, descripción, precio y la categoría a la que pertenece.
12. **reserveStates:**  
    Enumera los diferentes estados en los que puede encontrarse una reserva, como Confirmada, Pendiente o Cancelada.
13. **reserves:**  
    Almacena las reservas realizadas por los usuarios, incluyendo la fecha, hora, estado de la reserva y cliente asociado. Esta tabla también rastrea quién creó y quién actualizó la reserva.
14. **paymentMethods:**  
    Define los métodos de pago disponibles, como Tarjeta de Crédito o Transferencia Bancaria.
15. **statusPayment:**  
    Registra los diferentes estados de los pagos realizados, como Completado, Pendiente o Fallido.
16. **paymentTransactionInformation:**  
    Contiene la información de las transacciones de pago, como el cliente, el método de pago, el número de tarjeta y el estado del pago.
17. **statusGateway:**  
    Enumera los posibles estados en las transacciones gestionadas por las pasarelas de pago, como Aceptado o Rechazado.
18. **paymentGatewayRecords:**  
    Almacena los registros de las pasarelas de pago, incluyendo la respuesta del gateway, la referencia de la transacción y el código de error en caso de que ocurra.
19. **discount:**  
    Registra los descuentos aplicados a las transacciones, con un identificador único y el monto total del descuento.
20. **payment:**  
    Almacena la información de los pagos realizados para las reservas, incluyendo el total pagado, la transacción relacionada y, si aplica, el descuento utilizado.
21. **reserveProducts:**  
    Detalla los productos incluidos en cada reserva, indicando el producto, la cantidad y la reserva a la que están asociados.
22. **invoice:**  
    Registra las facturas generadas para los clientes. Cada factura incluye la fecha de emisión, el cliente, el empleado que la generó y los detalles de los productos reservados.

**El código relacionado puede observarse en el repositorio del github en la pate de “Backups”**

**GRAFICO DE LA BD**:



**Diseño de Base de Datos**

**Modelo de Datos**

. El modelado de datos es una fase esencial en el diseño de la base de datos “Proyecto”, abordando aspectos fundamentales como el modelo Entidad-Relación, la normalización y la cardinalidad.

El modelo Entidad-Relación proporciona una representación visual de las entidades principales, como empleados, clientes, productos y reservas, así como de las relaciones entre ellas

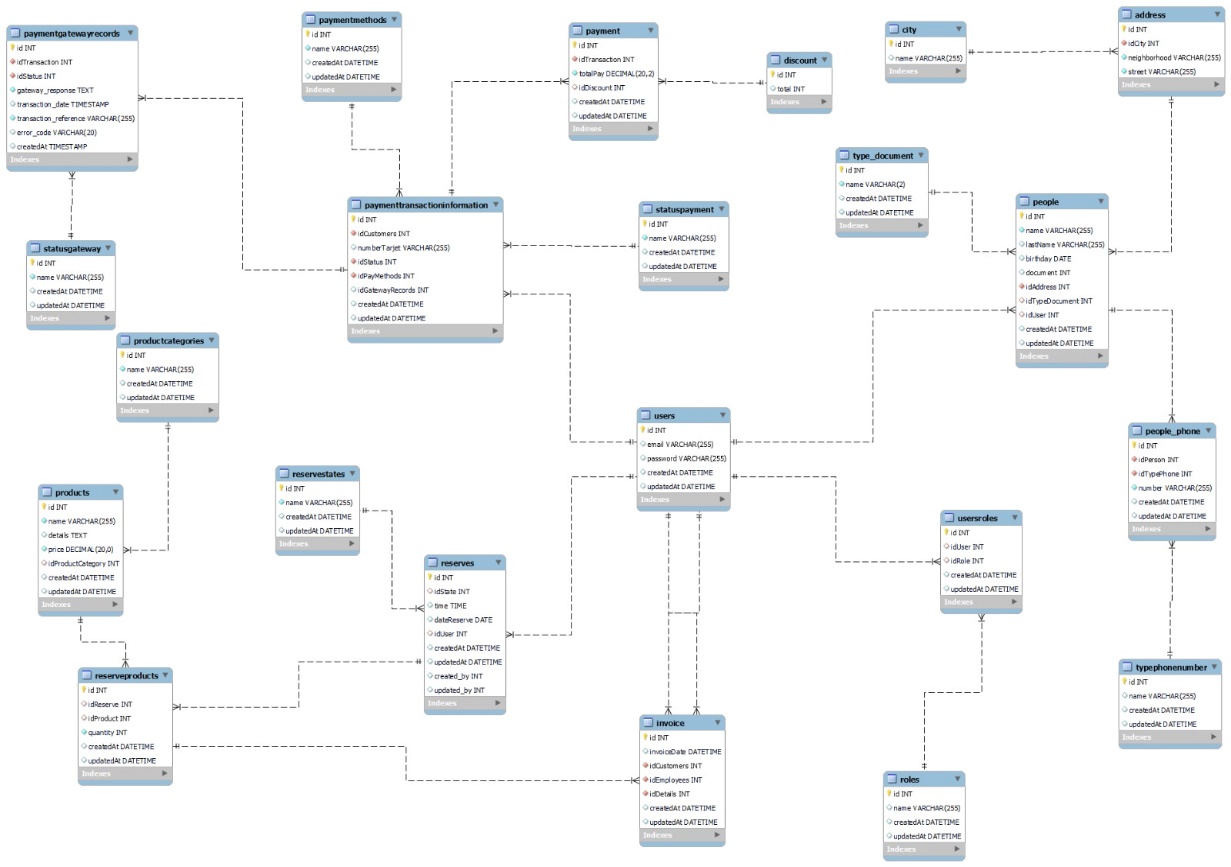
. La normalización asegura que los datos se organizan de manera eficiente, minimizando redundancias y mejorando la integridad de la información.

La cardinalidad define el número de instancias de una entidad que pueden estar asociadas con una instancia de otra entidad, estableciendo las restricciones necesarias para garantizar la precisión de las relaciones.

Asi que ahora vamos a profundizar en cada uno de ellos, ahí vamos.

#### Modelo entidad relación

Aquí se puede observar la primera parte de mi diagrama entidad relación



#### Normalizacion

**1NF (Primera Forma Normal):** Todas las tablas contienen datos atómicos sin listas ni conjuntos.

**2NF (Segunda Forma Normal):** Todas las tablas están libres de dependencias parciales; los atributos no clave dependen completamente de la clave primaria.

**3NF (Tercera Forma Normal):** No hay dependencias transitivas; cada atributo depende directamente de la clave primaria.

TODAS LAS TABLAS CUMPLEN CON LA NORMALIZACION ADECUADA PUEDE VERSE EN EL BACKUP FINAL

#### Cardinalidad

1. usersRoles (N) ---- (1) users y (N) ---- (1) roles

Descripción:

La tabla usersRoles gestiona la relación de muchos a muchos entre usuarios y roles.

Un usuario puede tener varios roles, y un rol puede estar asignado a varios usuarios.

Relación:

N a M: Gestionada a través de la tabla usersRoles.

Tablas involucradas:

users

roles

usersRoles

2. people (N) ---- (1) type\_document

Descripción:

Una persona tiene un tipo de documento asignado, mientras que un tipo de documento puede ser compartido por múltiples personas.

Relación:

N a 1: Cada persona tiene un único type\_document.

1 a N: Un type\_document puede usarse para varias personas.

3. people (N) ---- (1) address

Descripción:

Cada persona tiene una dirección registrada. Una dirección puede ser compartida por varias personas.

Relación:

N a 1: Una persona tiene una única dirección.

1 a N: Una dirección puede estar vinculada a múltiples personas.

4. address (N) ---- (1) city

Descripción:

Una dirección pertenece a una ciudad específica, y una ciudad puede contener múltiples direcciones.

Relación:

N a 1: Una dirección está asociada a una ciudad.

1 a N: Una ciudad puede tener varias direcciones.

5. people\_phone (N) ---- (1) typePhoneNumber

Descripción:

Un número telefónico tiene un tipo (ej. móvil o fijo). Un tipo de teléfono puede aplicarse a múltiples números.

Relación:

N a 1: Cada número tiene un tipo.

1 a N: Un tipo de teléfono puede aplicarse a varios números.

6. products (N) ---- (1) productCategories

Descripción:

Cada producto pertenece a una categoría. Una categoría puede tener múltiples productos.

7. reserves (N) ---- (1) users

Descripción:

Una reserva está asociada a un usuario que la realizó. Un usuario puede tener varias reservas.

8. reserves (N) ---- (1) reserveStates

Descripción:

Cada reserva tiene un estado que indica su situación. Un estado puede aplicarse a muchas reservas.

9. reserveProducts (N) ---- (1) reserves

Descripción:

Un producto reservado está asociado a una reserva específica. Una reserva puede incluir varios productos.

10. reserveProducts (N) ---- (1) products

Descripción:

Cada producto reservado se refiere a un producto del inventario.

11. paymentTransactionInformation (N) ---- (1) users

Descripción:

Una transacción de pago está vinculada a un usuario. Un usuario puede tener múltiples transacciones.

12. paymentTransactionInformation (N) ---- (1) paymentMethods

Descripción:

Cada transacción utiliza un método de pago específico. Un método de pago puede usarse en múltiples transacciones.

13. paymentTransactionInformation (N) ---- (1) statusPayment

Descripción:

Una transacción tiene un estado que refleja su situación. Un estado puede aplicarse a muchas transacciones.

14. paymentGatewayRecords (N) ---- (1) paymentTransactionInformation

Descripción:

Cada registro del gateway de pago está vinculado a una transacción. Una transacción puede tener múltiples registros de gateway.

15. paymentGatewayRecords (N) ---- (1) statusGateway

Descripción:

Un registro de gateway tiene un estado específico. Un estado puede aplicarse a múltiples registros.

16. payment (N) ---- (1) paymentTransactionInformation

Descripción:

Cada pago corresponde a una transacción. Una transacción puede involucrar múltiples pagos.

17. payment (N) ---- (1) discount

Descripción:

Un pago puede tener un descuento aplicado. Un descuento puede aplicarse a varios pagos.

18. invoice (N) ---- (1) users (clientes y empleados)

Descripción:

Cada factura está asociada a un cliente y un empleado. Un cliente o empleado puede tener múltiples facturas.

**Consideraciones de Diseño**

**Elección de llaves y entidades**

1.**type\_document**

* Llave primaria: id
* Atributos:
  + name (No nulo, Único)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

2. **users**

* Llave primaria: id
* Atributos:
  + email (Único)
  + password (Único)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

3. **roles**

* Llave primaria: id
* Atributos:
  + name (Único)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

4. **people**

* Llave primaria: id
* Atributos:
  + name (No nulo)
  + lastName
  + birthday
  + document (Único)
  + Llave foránea: idTypeDocument (Referencia a type\_document)
  + Llave foránea: idAddress (Referencia a address)
  + Llave foránea: idUser (Referencia a users)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

5. **address**

* Llave primaria: id
* Atributos:
  + Llave foránea: idCity (Referencia a city)
  + neighborhood (No nulo)
  + street (No nulo)

6. **city**

* Llave primaria: id
* Atributos:
  + name (No nulo)

7. **typePhoneNumber**

* Llave primaria: id
* Atributos:
  + name
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

8. **people\_phone**

* Llave primaria: id
* Atributos:
  + Llave foránea: idPerson (Referencia a people)
  + Llave foránea: idTypePhone (Referencia a typePhoneNumber)
  + number (No nulo)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

9. **productCategories**

* Llave primaria: id
* Atributos:
  + name (No nulo, Único)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

10. **products**

* Llave primaria: id
* Atributos:
  + name (No nulo)
  + details
  + price (No nulo)
  + Llave foránea: idProductCategory (Referencia a productCategories)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

11. **reserveStates**

* Llave primaria: id
* Atributos:
  + name (No nulo, Único)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

12. **reserves**

* Llave primaria: id
* Atributos:
  + time
  + dateReserve
  + Llave foránea: idUser (Referencia a users)
  + Llave foránea: idState (Referencia a reserveStates)
  + created\_by
  + updated\_by
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

13. **paymentMethods**

* Llave primaria: id
* Atributos:
  + name (No nulo)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

14. **statusPayment**

* Llave primaria: id
* Atributos:
  + name (No nulo)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

15. **paymentTransactionInformation**

* Llave primaria: id
* Atributos:
  + Llave foránea: idCustomers (Referencia a users)
  + numberTarjet
  + Llave foránea: idStatus (Referencia a statusPayment)
  + Llave foránea: idPayMethods (Referencia a paymentMethods)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

16. **paymentGatewayRecords**

* Llave primaria: id
* Atributos:
  + Llave foránea: idTransaction (Referencia a paymentTransactionInformation)
  + Llave foránea: idStatus (Referencia a statusGateway)
  + gateway\_response (No nulo)
  + transaction\_date
  + transaction\_reference
  + error\_code (Opcional)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)

17. **statusGateway**

* Llave primaria: id
* Atributos:
  + name (No nulo)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

18. **payment**

* Llave primaria: id
* Atributos:
  + Llave foránea: idTransaction (Referencia a paymentTransactionInformation)
  + totalPay (No nulo)
  + Llave foránea: idDiscount (Referencia a discount)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

19. **reserveProducts**

* Llave primaria: id
* Atributos:
  + Llave foránea: idReserve (Referencia a reserves)
  + Llave foránea: idProduct (Referencia a products)
  + quantity (No nulo)
  + createdAt (Por defecto, CURRENT\_TIMESTAMP)
  + updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática)

20. **invoice**

1. Llave primaria: id
2. Atributos:
   * invoiceDate
   * Llave foránea: idCustomers (Referencia a users)
   * Llave foránea: idEmployees (Referencia a users)
   * Llave foránea: idDetails (Referencia a reserveProducts)
   * createdAt (Por defecto, CURRENT\_TIMESTAMP)
   * updatedAt (Por defecto, CURRENT\_TIMESTAMP con actualización automática**)**

**Relaciones entre tablas**

* En la tabla type\_document, la columna id es referenciada por idTypeDocument en las tablas people, customers y employees.
* En la tabla users, la columna id es referenciada por idUser en la tabla people, por idCustomers en la tabla paymentTransactionInformation, por idUser en la tabla reserves, y por idEmployees en la tabla invoice.
* En la tabla address, la columna id es referenciada por idAddress en la tabla people. La columna idCity en la tabla address es referenciada por id en la tabla city.
* En la tabla products, la columna id es referenciada por idProductCategory en la tabla productCategories.
* En la tabla reserves, la columna id es referenciada por idCustomers en la tabla customers y por idState en la tabla reserveStates.
* En la tabla payment, la columna id es referenciada por idMethod en la tabla paymentMethods y por idReserves en la tabla reserves.
* En la tabla reserve\_products, la columna id es referenciada por idReserve en la tabla reserves y por idProduct en la tabla products.
* En la tabla paymentTransactionInformation, la columna id es referenciada por idCustomers en la tabla users, por idPayMethods en la tabla paymentMethods, y por idStatus en la tabla statusPayment.
* En la tabla paymentGatewayRecords, la columna id es referenciada por idTransaction en la tabla paymentTransactionInformation y por idStatus en la tabla statusGateway.

**Nombre y descripción**

El sistema de base de datos denominado “Proyecto” ha sido desarrollado con el objetivo de optimizar la gestión integral de un complejo deportivo. Este sistema está diseñado para ofrecer una solución eficiente y organizada que abarca diversas áreas clave, como la administración de empleados, clientes, productos, reservas y pagos.

El propósito principal de la base de datos es garantizar un funcionamiento rápido y eficiente, facilitando la organización y el acceso a información relevante para las operaciones del complejo deportivo. Para lograr este objetivo, “Proyecto” incluye varias tablas interrelacionadas que cubren las siguientes áreas:

**type\_document:** Almacena los diferentes tipos de documentos de identificación utilizados por empleados y clientes, asegurando que cada registro tenga una referencia adecuada.

**users:** Contiene información sobre los usuarios del sistema, incluyendo su dirección de correo electrónico y contraseña, lo que permite gestionar el acceso a la plataforma.

**roles:** Define los roles que pueden tener los usuarios en el sistema, facilitando la administración de permisos y funciones dentro del complejo.

**typePhoneNumber:** Registra los distintos tipos de números de teléfono que pueden ser asociados con las personas, como móviles y fijos, mejorando la gestión de la información de contacto.

**city:** Almacena las ciudades relacionadas con las direcciones de los empleados y clientes, contribuyendo a una mejor organización geográfica.

**address:** Registra las direcciones completas, incluyendo ciudad, barrio y calle, asegurando que la información de localización sea precisa.

**people:** Contiene información sobre las personas asociadas al complejo, tanto empleados como clientes, incluyendo datos personales, dirección y tipo de documento.

**people\_phone:** Permite asociar múltiples números de teléfono a una persona, mejorando la comunicación y el contacto.

**usersRoles:** Facilita la relación entre usuarios y roles, definiendo qué permisos tiene cada usuario en el sistema.

**productCategories:** Organiza los productos del complejo en categorías específicas, facilitando su gestión y búsqueda.

**products:** Administra el inventario de productos disponibles, incluyendo detalles sobre su nombre, descripción, precio y categoría correspondiente.

**reserveStates:** Define los distintos estados posibles para las reservas realizadas, permitiendo un seguimiento más efectivo del estado de cada reserva.

**reserves:** Facilita la gestión de las reservas, registrando información sobre el estado, fecha, hora y cliente asociado.

**paymentMethods:** Registra los métodos de pago disponibles para las transacciones realizadas por los clientes, mejorando la flexibilidad en el proceso de pago.

**statusPayment:** Almacena el estado de los pagos, permitiendo un seguimiento claro de cada transacción.

**paymentTransactionInformation:** Documenta la información de las transacciones de pago, incluyendo detalles como el número de tarjeta y el estado de la transacción.

**statusGateway:** Registra los estados posibles de las transacciones en el gateway de pagos, asegurando un seguimiento claro del proceso.

**paymentGatewayRecords:** Almacena registros de las respuestas del gateway de pagos, incluyendo información sobre la transacción y posibles errores.

**discount:** Permite gestionar descuentos aplicables a las transacciones, asegurando que los precios sean correctos.

**payment:** Registra los pagos efectuados por los clientes, incluyendo el monto total y la asociación con la transacción y descuentos aplicables.

**reserveProducts:** Detalla los productos específicos incluidos en cada reserva, especificando la cantidad y el precio, lo que permite una mejor gestión del inventario en relación con las reservas.

Este enfoque integral permite una administración más eficaz del complejo deportivo, asegurando que todas las operaciones se realicen de manera fluida y organizada.

**Análisis y Discusión**

El análisis general del proyecto de desarrollo la base de datos ha sido positivo y ha cumplido con mis expectativas. He realizado una evaluación exhaustiva y considero que todo lo implementado es funcional y adecuado. Aunque reconozco que siempre hay oportunidades para mejorar y ser más eficiente, estoy satisfecho con el trabajo realizado.

Este proyecto representa un importante primer paso en mi desarrollo profesional y personal, permitiéndome abordar mi proyecto final de manera seria y meticulosa. Estoy contento con los logros alcanzados y con la calidad del trabajo desarrollado, que sienta una sólida base para futuros avances en mi carrera y mi desarrollo de proyecto de vida.

**Interpretación de Resultados**

Los resultados obtenidos de la base de datos confirman que esta es funcional y está estructurada de manera correcta y adecuada, siguiendo las instrucciones proporcionadas. Se realizaron diversas consultas y se atendieron todas las solicitudes planteadas, obteniendo resultados consistentes y precisos.

La estructura de la base de datos cumple con los requisitos establecidos y se ha verificado su correcto funcionamiento a través de pruebas exhaustivas. Bajo mi criterio, la base de datos es adecuada y está completa, incluyendo toda la información necesaria para cumplir con su propósito de manera efectiva. Esto asegura que la base de datos no solo es operativa, sino que también es útil y capaz de soportar las operaciones del sistema de manera eficiente.

**Como funciona**

La base de datos “Proyecto” está diseñada para gestionar un complejo deportivo, donde se llevan a cabo múltiples actividades y operaciones. Su función principal es almacenar y organizar información sobre empleados, clientes, productos, reservas y pagos. Esto permite un acceso rápido y eficiente a los datos, facilitando la toma de decisiones y la administración general del complejo.

**Estructura de la Base de Datos**

La base de datos se compone de varias **tablas** interconectadas. Cada tabla almacena un tipo específico de información. A continuación, se describen las tablas más importantes y cómo interactúan entre sí.

1. **type\_document**:
   * Esta tabla almacena los diferentes tipos de documentos de identificación, como DNI, pasaporte, etc. Es importante porque ayuda a clasificar a los empleados y clientes según el tipo de documento que presentan.
2. **users**:
   * Aquí se registran los usuarios del sistema, que pueden ser empleados o administradores. Cada usuario tiene un correo electrónico único y una contraseña para acceder al sistema. Esto asegura que solo las personas autorizadas puedan manejar la información del complejo.
3. **roles**:
   * Define qué tipo de funciones pueden desempeñar los usuarios en el sistema. Por ejemplo, algunos usuarios pueden tener permisos para gestionar reservas, mientras que otros solo pueden ver información. Esto ayuda a mantener la seguridad y la organización.
4. **people**:
   * Esta tabla contiene información sobre las personas asociadas al complejo, ya sean empleados o clientes. Aquí se almacena su nombre, apellido, fecha de nacimiento, documento de identificación y dirección. Esto es crucial para gestionar la relación con los clientes y el personal.
5. **customers**:
   * Específicamente, esta tabla guarda datos de los clientes del complejo, facilitando el registro de sus reservas y pagos. Esto permite a la administración conocer a quiénes están atendiendo y cómo interactúan con el complejo.
6. **products y productCategories**:
   * Las tablas de productos y categorías de productos ayudan a gestionar el inventario disponible en el complejo. Los productos pueden ser equipos deportivos, comida y bebida, etc. Cada producto pertenece a una categoría específica, lo que facilita su búsqueda y venta.
7. **reserves**:
   * Esta tabla es fundamental para gestionar las reservas de canchas u otros servicios del complejo. Cada reserva incluye información sobre la fecha, hora y estado (si está confirmada, cancelada, etc.). Está vinculada a los clientes, lo que permite saber quién hizo cada reserva.
8. **payment**:
   * Los pagos realizados por los clientes se registran aquí. Cada pago está vinculado a una reserva específica y puede incluir descuentos si corresponde. Esto permite llevar un control financiero de las transacciones que se realizan en el complejo.
9. **reserveProducts**:
   * Cuando un cliente hace una reserva, puede incluir productos adicionales (como alimentos o equipamiento). Esta tabla registra qué productos se han reservado, en qué cantidad y a qué precio.

**Cómo Funciona el Sistema**

1. **Registro de Clientes y Empleados**:
   * Cuando una nueva persona se une al complejo, se registra en la tabla people, y se le asigna un tipo de documento de la tabla type\_document. Si es un empleado, también se agrega a la tabla users con su información de acceso.
2. **Gestión de Reservas**:
   * Los clientes pueden hacer reservas para usar las instalaciones del complejo. Se registra una nueva entrada en la tabla reserves, donde se guarda información sobre la fecha, hora y el cliente que realiza la reserva. Esto se vincula a través del ID del cliente desde la tabla customers.
3. **Proceso de Pagos**:
   * Al hacer una reserva, los clientes pueden también realizar pagos. Se crea una entrada en la tabla payment, donde se registra el método de pago utilizado (que se encuentra en la tabla paymentMethods), la cantidad pagada y si hay descuentos aplicables. Esto se relaciona directamente con la reserva correspondiente.
4. **Inventario de Productos**:
   * Durante el proceso de reserva, si un cliente decide incluir productos, se registra esta información en la tabla reserveProducts, donde se indican los productos elegidos, la cantidad y el precio. Esto permite al complejo llevar un control del inventario disponible.
5. **Seguimiento y Control**:
   * Todas las acciones en el sistema quedan registradas con marcas de tiempo, lo que permite llevar un control de cuándo se hicieron los registros y por quién. Esto es útil para auditar el funcionamiento del complejo y para resolver cualquier inconveniente que pueda surgir.

# Procedimientos Avanzados

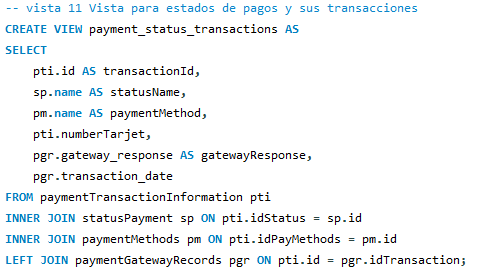
## Vistas

Una vista es como una **ventana virtual** hacia la base de datos. Es una consulta almacenada que se comporta como una tabla. Permite mostrar datos específicos o combinar información de varias tablas sin duplicar los datos reales.

**Ventajas de las vistas:**

* **Simplificación**: Ocultan la complejidad de las consultas.
* **Seguridad**: Restringen el acceso a datos sensibles al mostrar solo lo necesario.
* **Reutilización**: Se pueden usar en múltiples consultas y aplicaciones.

**Ejemplo:**



**Explicación del Código**

Esta consulta SQL crea una vista llamada payment\_status\_transactions. La vista combina información de varias tablas relacionadas con transacciones de pago, estados de pagos, métodos de pago y registros del gateway de pagos. A continuación, desglosamos cada componente para que sea más comprensible:

1. **Qué hace la vista:**

La vista proporciona un reporte consolidado que muestra:

* El ID de la transacción.
* El estado del pago.
* El método de pago utilizado.
* El número de tarjeta (parcial o completo, según cómo se guarde).
* La respuesta del gateway de pagos.
* La fecha en que se realizó la transacción.

1. **Tablas involucradas:**

* paymentTransactionInformation (pti): Contiene información básica de cada transacción, como su ID, el estado del pago y el método de pago utilizado.
* statusPayment (sp): Almacena los diferentes estados que puede tener un pago, como "Pagado", "Pendiente" o "Fallido".
* paymentMethods (pm): Lista los métodos de pago disponibles, como "Tarjeta de crédito", "PayPal", etc.
* paymentGatewayRecords (pgr): Guarda los registros relacionados con el gateway de pagos, como respuestas, errores y fechas.

1. **Cómo se combinan las tablas:**

* **INNER JOIN:** Se utiliza para combinar registros que tienen coincidencias en ambas tablas:

paymentTransactionInformation se combina con statusPayment para obtener el nombre del estado del pago.

paymentTransactionInformation también se combina con paymentMethods para obtener el nombre del método de pago.

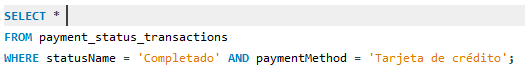
* **LEFT JOIN:** Se usa para combinar datos de paymentGatewayRecords, pero incluye las transacciones incluso si no hay registros correspondientes en el gateway (es decir, si no se procesaron por el gateway o no se recibió respuesta).

1. **Campos seleccionados en la vista:**

* **pti.id AS transactionId**: El ID único de la transacción.
* **sp.name AS statusName:** El nombre del estado del pago, obtenido de la tabla statusPayment.
* **pm.name AS paymentMethod:** El nombre del método de pago, obtenido de la tabla paymentMethods.
* **pti.numberTarjet:** El número de la tarjeta asociada a la transacción.
* pgr.gateway\_response AS gatewayResponse: La respuesta del gateway de pagos (por ejemplo, "Aprobado", "Fallido").
* **pgr.transaction\_date:** La fecha en que se procesó la transacción en el gateway.

**Ejemplo de uso:**

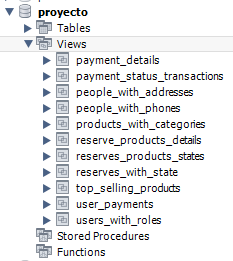
Supongamos que quieres consultar todas las transacciones junto con sus estados y métodos de pago. Una vez creada esta vista, puedes usarla así:



Una vez ejecutado su resultado se vería de la siguiente manera:



Y así es cómo aparecerían nuestros views creados:



**Ventajas de la vista:**

**Consolidación** de datos: Combina información dispersa en varias tablas, haciéndola más accesible.

**Reutilización**: La vista puede ser reutilizada en otras consultas sin necesidad de escribir uniones complejas cada vez.

**Simplificación**: Facilita el acceso a datos críticos para reportes y análisis.

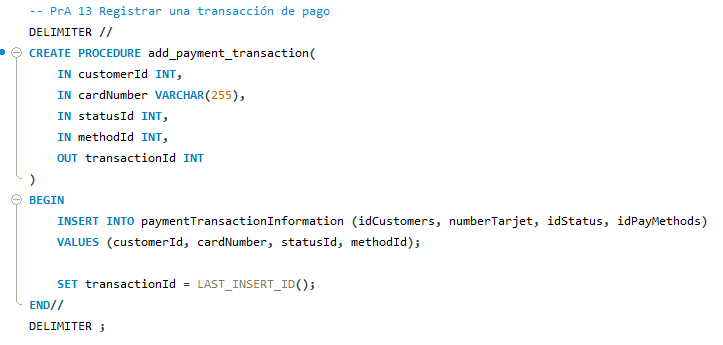
## Procedimientos Almacenados:

Son bloques de código SQL que puedes guardar en el servidor de base de datos y ejecutar cuando sea necesario. Funcionan como funciones o métodos que automatizan tareas repetitivas.

**Ventajas de los procedimientos almacenados:**

* Reutilización de código: Reducen la duplicación.
* Rendimiento: Se ejecutan en el servidor, lo que mejora la eficiencia.
* Seguridad: Pueden limitar el acceso a datos sensibles.

**Ejemplo:**



Explicación del Procedimiento Almacenado: add\_payment\_transaction

Este procedimiento almacenado está diseñado para agregar una nueva transacción de pago en la tabla paymentTransactionInformation. A continuación, desglosamos cada parte del código para que sea más fácil de entender:

1. **Propósito del Procedimiento**

* Recibe información relacionada con una transacción de pago, como el ID del cliente, el número de tarjeta, el estado del pago y el método de pago.
* Inserta un nuevo registro en la tabla paymentTransactionInformation.
* Devuelve el ID único de la transacción recién creada.

1. **Parámetros del Procedimiento**

El procedimiento tiene cinco parámetros:

* **Entrada (IN):**
* **customerId**: El ID del cliente que realiza la transacción.
* **cardNumber**: El número de la tarjeta utilizada en el pago.
* **statusId**: El estado del pago (por ejemplo, pagado, pendiente, fallido).
* **methodId**: El método de pago utilizado (como tarjeta de crédito, débito, etc.).
* **Salida (OUT):**
* **transactionId**: Almacena el ID único de la transacción recién creada. Se utiliza para que la aplicación o sistema que llame al procedimiento obtenga el ID de la nueva transacción.

1. **Cuerpo del Procedimiento**

El cuerpo tiene dos partes principales:

**A. Inserción de datos**

Inserta los valores proporcionados en la tabla paymentTransactionInformation.

Los campos en la tabla son:



* **idCustomers**: El cliente asociado con la transacción.
* **numberTarjet**: El número de la tarjeta usada.
* **idStatus**: El estado del pago (referencia a la tabla statusPayment).
* **idPayMethods**: El método de pago (referencia a la tabla paymentMethods).

**B. Recuperación del último ID insertado**



**LAST\_INSERT\_ID()** obtiene el ID del último registro que se insertó automáticamente en la tabla.

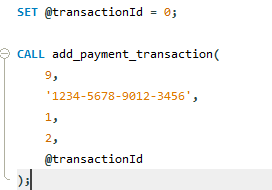
Este **ID** único se asigna al parámetro de salida **transactionId**, lo que permite devolverlo al sistema que llamó al procedimiento.

1. **Uso del Procedimiento**

Para llamar a este procedimiento, se usa el comando **CALL** con los valores de entrada y una variable para capturar la salida.

**Ejemplo de uso:**

Supongamos que un cliente con ID 9 realiza un pago con tarjeta 1234-5678-9012-3456, el estado del pago es 1 (pendiente) y el método de pago es 2 (tarjeta débito). Así se llamaría al procedimiento:



Aquí, @transactionId captura el ID de la transacción recién creada.

El comando SELECT mostrará el ID para confirmar que la operación fue exitosa.



**Ventajas del Procedimiento**

* **Automatización**: Simplifica la lógica de inserción y recuperación del ID.
* **Reutilización**: Puede ser llamado desde múltiples aplicaciones o sistemas.
* **Seguridad**: Los parámetros previenen inyecciones SQL.
* **Eficiencia**: Reduce la cantidad de consultas al combinar inserción y recuperación del ID en un solo procedimiento.

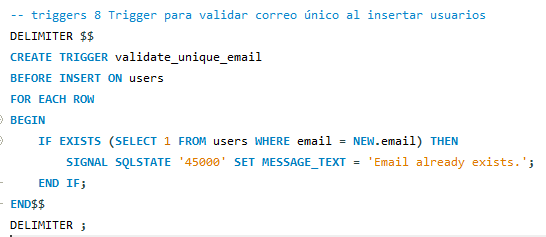
## Triggers

Un trigger es un fragmento de código que se ejecuta automáticamente en respuesta a un evento en una tabla, como una inserción, actualización o eliminación de datos.

**Ventajas de los triggers:**

* **Automatización**: Realizan tareas automáticamente.
* **Integridad de datos**: Garantizan que las reglas de negocio se cumplan.
* **Auditoría**: Registran cambios en las tablas.

**Ejemplo**:



**Explicación del Trigger: validate\_unique\_email**

Este código crea un trigger llamado validate\_unique\_email que se activa antes de insertar un nuevo registro en la tabla users. Su propósito es validar que el correo electrónico sea único para evitar duplicados en la base de datos. Si ya existe un correo igual en la tabla, el trigger lanza un error y cancela la inserción.

1. **Propósito del Trigger**

* Se asegura de que no se puedan insertar usuarios con correos electrónicos duplicados.
* Mejora la integridad de los datos al prevenir errores lógicos en el sistema.
* Ofrece un mensaje de error específico cuando se detecta un intento de insertar un correo duplicado.

1. **Componentes del Trigger**

**A. Evento del Trigger**

* **BEFORE INSERT**: El trigger se ejecuta antes de que se inserte un registro en la tabla users.
* **ON users:** Especifica que este trigger se aplica a la tabla users.

**B. Acción del Trigger**

* **FOR EACH ROW:** Indica que el trigger se ejecuta para cada fila que se intenta insertar, permitiendo validar cada nuevo registro.

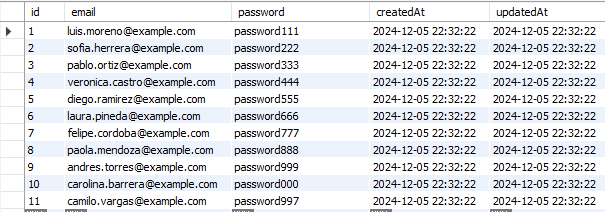
**C. Lógica del Trigger**

* **IF EXISTS:** Verifica si ya hay un registro en la tabla **users** con el mismo correo electrónico que el del nuevo registro (**NEW.email**).
* **NEW.email** representa el valor del campo email del registro que se intenta insertar.
* **SIGNAL SQLSTATE '45000':** Lanza un error personalizado con el código **SQLSTATE 45000**, que indica una condición específica definida por el usuario.
* **SET MESSAGE\_TEXT**: Proporciona un mensaje descriptivo: **"Email already exists.".**
* Si el correo ya existe, la inserción se detiene y se devuelve el error.

1. **Ejemplo de Funcionamiento**

**A. Estructura inicial de la tabla users**

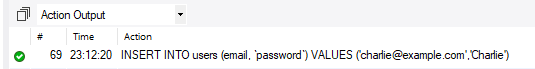
Supongamos que la tabla users contiene los siguientes registros:



**B. Intento de inserción válida**



**Resultado**: La inserción se realiza correctamente, ya que el correo charlie@example.com no está en uso.



**C. Intento de inserción duplicada**



**Resultado**: El trigger se activa y lanza el siguiente error:





La inserción no se realiza, manteniendo la integridad de la tabla.

1. **Ventajas del Trigger**

* **Integridad de datos**: Garantiza que no existan correos duplicados en la base de datos.
* **Automatización**: Simplifica la lógica, ya que no es necesario implementar esta validación en el código de la aplicación.
* **Retroalimentación inmediata**: El mensaje de error permite a los usuarios o desarrolladores entender qué sucedió y corregir el problema.

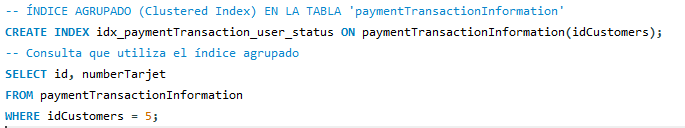
## Índices

Un índice en una base de datos es una estructura de datos asociada a una o más columnas de una tabla. Sirve para acelerar operaciones como búsquedas, filtrados, ordenamientos o uniones. Se puede comparar con el índice de un libro: en lugar de leer cada página para buscar un término, el índice te indica directamente dónde está.

**Ventajas de los índices**

* **Mayor velocidad en consultas:** Los índices hacen que las búsquedas, ordenamientos y filtrados sean mucho más rápidos.
* **Mejor rendimiento en relaciones:** Aceleran las uniones (**JOIN**) entre tablas, especialmente si las columnas relacionadas tienen índices.
* **Optimización de filtros y ordenamientos:** Las cláusulas **WHERE, ORDER BY y GROUP BY** funcionan más eficientemente con índices.

**Ejemplo 1 Índice Agrupado:**



**Explicación de Índice Agrupado (Clustered Index)**

Un índice agrupado (clustered index) organiza físicamente las filas de una tabla en el disco según el orden de una columna específica. En este caso, el índice idx\_paymentTransaction\_user\_status organiza los datos de la tabla paymentTransactionInformation por la columna idCustomers.

1. **Creación del Índice**

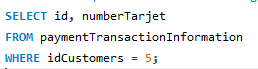


Define un índice agrupado en la columna **idCustomers**.

Las filas de la tabla se organizan físicamente en el disco basándose en los valores de **idCustomers**, mejorando el rendimiento de consultas que buscan datos específicos de esta columna.

El índice actúa como un mapa optimizado para localizar registros.

1. **Consulta que utiliza el Índice**



* **Cómo funciona con el índice agrupado:**
* La base de datos consulta directamente el índice para localizar las filas donde **idCustomers = 5**.
* Dado que las filas ya están organizadas por **idCustomers**, no es necesario realizar una búsqueda completa en la tabla.
* Esto reduce significativamente el tiempo de ejecución, especialmente en tablas grandes.
* Resultado esperado:
* Devuelve las columnas **id** y **numberTarjet** de todas las transacciones realizadas por el cliente con **idCustomers = 5**.

1. **Resultado**



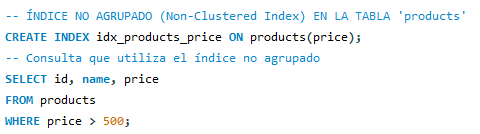
1. **Ventajas del Índice Agrupado**

* **Búsquedas rápidas:** Las consultas basadas en la columna idCustomers son mucho más rápidas, ya que la tabla está ordenada según esta columna.
* **Optimización de rangos:** Las consultas que buscan un rango de valores (BETWEEN, >, <) en **idCustomers** también se benefician del índice.
* **Acceso directo a datos:** Dado que el índice agrupado organiza físicamente los datos, las filas relacionadas están cerca unas de otras en el disco, reduciendo el número de operaciones de lectura.

1. **Limitaciones del Índice Agrupado**

* **Una sola columna:** Solo puede haber un índice agrupado por tabla, ya que organiza físicamente las filas en el disco.
* **Impacto en escrituras:** Las operaciones de inserción, actualización y eliminación en **idCustomers** pueden ser más lentas porque la base de datos necesita mantener el orden físico de las filas.
* **Tamaño del índice:** En tablas grandes, el índice agrupado puede consumir espacio adicional en el disco.

**Ejemplo 2 Índices no Agrupados:**



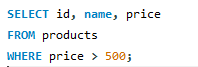
Un índice no agrupado es una estructura separada de la tabla que contiene referencias a las filas. A diferencia del índice agrupado, no afecta el orden físico de las filas en la tabla, pero almacena un mapa que acelera las búsquedas basadas en una columna o combinación de columnas.

1. **Creación del Índice**



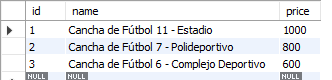
* Define un índice no agrupado en la columna price de la tabla products.
* Crea una estructura que asocia los valores de price con las ubicaciones (referencias) de las filas correspondientes en la tabla.

1. **Consulta que utiliza el Índice**



* La base de datos utiliza el índice **idx\_products\_price** para encontrar rápidamente las filas donde **price > 500.**
* Una vez que localiza las filas relevantes mediante el índice, recupera los valores de las columnas adicionales (**id y name**) directamente de la tabla.
* Este proceso, conocido como búsqueda de **clave (key lookup),** es mucho más rápido que realizar una búsqueda secuencial en toda la tabla.

1. **Resultado**



1. **Ventajas del Índice No Agrupado**

* **Búsquedas rápidas:** Acelera las consultas que filtran o buscan datos por la columna **price**.
* **Independencia del orden físico:** A diferencia del índice agrupado, el índice no agrupado no altera la organización física de la tabla. Es útil cuando necesitas índices en varias columnas.
* **Flexibilidad**: Se pueden crear múltiples índices no agrupados en diferentes columnas de una tabla, lo que permite optimizar diversas consultas.

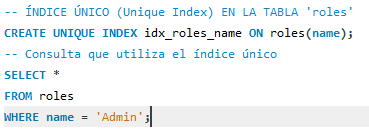
1. **Limitaciones del Índice No Agrupado**

**Espacio adicional:** Los índices no agrupados ocupan espacio en el disco, ya que son estructuras separadas.

**Impacto en escrituras:** Las operaciones de inserción, actualización o eliminación en la tabla pueden ser más lentas porque los índices deben actualizarse.

**Rendimiento en consultas complejas:** Si la consulta necesita muchas columnas que no están incluidas en el índice, el rendimiento puede ser menor debido a las búsquedas adicionales en la tabla.

**Ejemplo 3 Índices Únicos:**



Un índice único garantiza que los valores en una columna (o combinación de columnas) sean únicos en toda la tabla. Es una herramienta útil para asegurar la integridad de los datos y optimizar consultas de búsqueda.

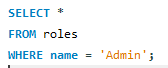
En este caso, el índice **idx\_roles\_name** asegura que no haya duplicados en la columna **name** de la tabla roles.

1. **Creación del Índice**



* Define un índice único en la columna **name** de la tabla roles.
* **Evita duplicados**: Si se intenta insertar o actualizar un registro con un valor duplicado en la columna **name**, la base de datos generará un error.
* **Optimiza búsquedas**: Las consultas que filtran datos por la columna **name** son más rápidas porque el índice actúa como un mapa optimizado.

1. **Consulta que utiliza el Índice**



**Cómo funciona el índice único con esta consulta:**

* La base de datos utiliza el índice **idx\_roles\_name** para localizar rápidamente las filas donde **name = 'Admin'**.
* Al tratarse de un índice único, la base de datos sabe que puede haber solo una fila con ese valor, lo que simplifica aún más la búsqueda.
* Esto evita que se haga una búsqueda secuencial en toda la tabla.

1. **Resultado esperado:**



1. **Ventajas del Índice Único**

* **Garantía de unicidad:** Evita errores lógicos al garantizar que los valores en la columna **name** sean únicos.Por ejemplo, no pueden existir dos roles con el nombre **Admin**.
* **Rendimiento mejorado:** Acelera búsquedas y filtrados en la columna **name**.
* **Integridad de datos:** Mejora la calidad de los datos al prevenir duplicados accidentales.
* **Soporte para claves alternativas:** Si una columna no es clave primaria, un índice único puede actuar como una clave alternativa para garantizar unicidad.

1. **Limitaciones del Índice Único**

* **Espacio adicional:** Al igual que otros índices, ocupa espacio adicional en disco.
* **Impacto en escrituras:** Inserciones o actualizaciones en la columna **name** pueden ser más lentas, ya que la base de datos verifica si el valor es único.
* **No se aplica en columnas con muchos valores nulos:** La mayoría de las bases de datos permiten múltiples valores **NULL** en una columna con índice único, lo que puede ser un comportamiento inesperado.

**Ejemplo 4 Índices Compuestos:**



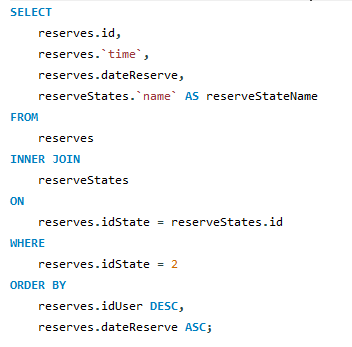
Un índice compuesto es un índice que cubre múltiples columnas en una tabla. En este caso, el índice **idx\_reserves\_state\_user\_date** se aplica a las **columnas idState, idUser y dateReserve** de la tabla **reserves**. Este tipo de índice mejora el rendimiento de consultas que filtran y ordenan datos utilizando estas columnas.

1. **Creación del Índice**



* Crea un índice compuesto que incluye las columnas idState, idUser y dateReserve.
* **Orden de prioridad:**
* El índice primero organiza los datos por idState.
* Dentro de cada idState, organiza los datos por idUser.
* Dentro de cada combinación de idState e idUser, organiza los datos por dateReserve.
* Optimiza consultas que utilicen estas columnas en el filtro (WHERE) o el ordenamiento (ORDER BY).

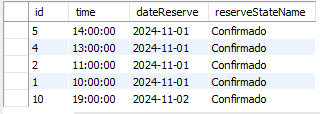
1. **Consulta que utiliza el Índice**



**Cómo funciona el índice compuesto con esta consulta:**

* **Filtro por idState:** El índice identifica rápidamente todas las filas donde idState = 2, sin necesidad de realizar una búsqueda completa en la tabla.
* **Ordenamiento por idUser y dateReserve:** Dentro de las filas filtradas (idState = 2), el índice ya tiene los datos organizados por idUser y luego por dateReserve. Esto evita que la base de datos haga un ordenamiento adicional.
* **Unión con reserveStates:** Aunque el índice no afecta directamente la tabla reserveStates, la consulta se beneficia de la optimización en la tabla reserves.

1. **Resultado esperado:**



1. **Ventajas del Índice Compuesto**

* **Rendimiento mejorado:** Acelera consultas que filtran y ordenan utilizando las columnas incluidas en el índice.
* **Optimización de múltiples columnas:** En lugar de crear índices separados para cada columna, un índice compuesto cubre escenarios donde varias columnas son relevantes para la consulta.
* **Menor uso de recursos:** Reduce la carga de la CPU y el uso de memoria al evitar búsquedas y ordenamientos completos.

# Encriptación

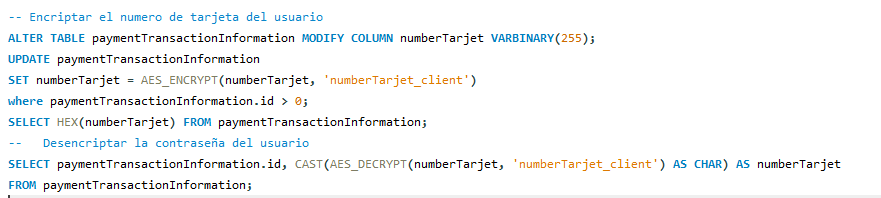
Es el proceso de convertir datos legibles (texto plano) en un formato no legible (texto cifrado) para proteger su confidencialidad. La encriptación en bases de datos protege información sensible como contraseñas, números de tarjetas, y datos personales.

**Implementación básica:**

* **Encriptar datos sensibles antes de almacenarlos:**

Usar algoritmos como AES-256 para proteger columnas específicas.

**Ejemplo:**



**Explicación del Proceso de Encriptación y Desencriptación del Número de Tarjeta**

Este fragmento de código SQL implementa la encriptación de números de tarjetas en la tabla paymentTransactionInformation para proteger la información sensible de los usuarios.

1. **Modificar la Columna para Almacenar Datos Encriptados**



* Cambia el tipo de datos de la columna numberTarjet a VARBINARY(255), necesario para almacenar datos encriptados.
* **VARBINARY**: Permite almacenar datos binarios de longitud variable, como el resultado de AES\_ENCRYPT.

1. **Encriptar los Datos Existentes**



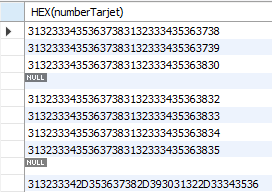
* Recorre las filas de la tabla paymentTransactionInformation y encripta los valores existentes en la columna numberTarjet usando el algoritmo AES.
* La clave de encriptación es 'numberTarjet\_client'. Es importante mantener esta clave segura y consistente para descifrar los datos.

1. **Visualizar Datos Encriptados**



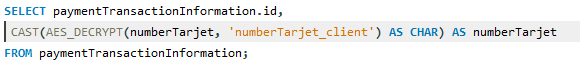
* Convierte los datos binarios encriptados en formato hexadecimal para que sean más legibles.
* Esto es útil para verificar que los datos han sido correctamente cifrados.

1. **Resultado Ejemplo:**



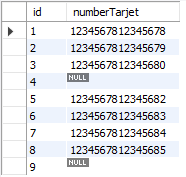
**Nota:** Los espacios NULL es debido a que en esa posición no tienen tarjeta registrada.

1. **Desencriptar los Datos**



* Usa AES\_DECRYPT para desencriptar los valores en la columna numberTarjet utilizando la clave 'numberTarjet\_client'.
* CAST(... AS CHAR): Convierte los datos binarios desencriptados en texto legible.

1. **Resultado Ejemplo:**



**Consideraciones de Seguridad**

1. **Almacenamiento seguro de claves:**

* La clave 'numberTarjet\_client' debe mantenerse segura y no almacenarse en texto plano en el código o base de datos.
* Usar un gestor de claves o variables de entorno para protegerla.

1. **Encriptación en tránsito:** Asegúrate de usar conexiones seguras (SSL/TLS) entre la aplicación y la base de datos.
2. **Permisos restringidos:** Limitar el acceso a la columna encriptada solo a usuarios o servicios autorizados.
3. **Rotación de claves:** Si es necesario cambiar la clave de encriptación, desencripta los datos con la clave antigua, vuelve a cifrarlos con la nueva, y actualiza la base de datos.

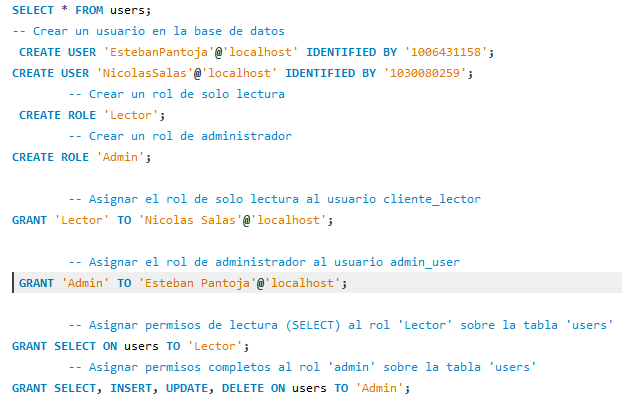
**Ventajas de Este Enfoque**

* **Protección de datos sensibles**: Los números de tarjeta están protegidos incluso si la base de datos es comprometida.
* **Cumplimiento normativo**: Cumple con estándares como PCI DSS que exigen la protección de datos financieros.
* **Flexibilidad**: Puedes desencriptar datos cuando sea necesario para procesarlos o mostrarlos en entornos seguros.

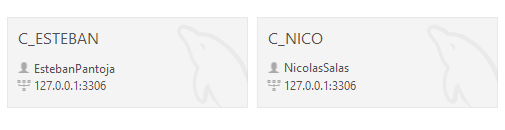
# Usuarios Y Roles

Un **rol** es un conjunto de privilegios que puedes asignar a varios usuarios, facilitando la gestión de permisos.

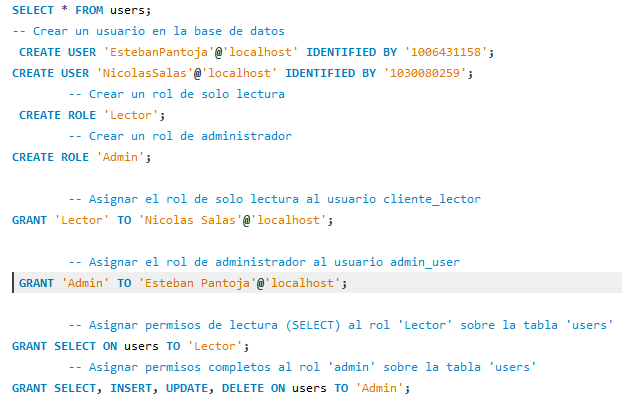
1. **Crear un Rol**



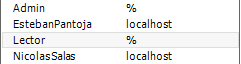
* Crea dos usuarios en el servidor MySQL:
* EstebanPantoja con la contraseña '1006431158'.
* NicolasSalas con la contraseña '1030080259'.
* 'localhost' indica que los usuarios solo pueden conectarse desde la máquina local, es decir, no desde una IP externa.
* De esta manera se verán:

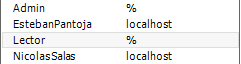


1. **Crear Roles**

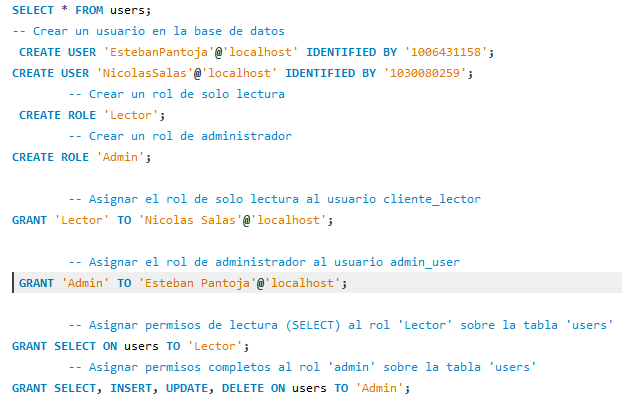


* Crea dos roles en la base de datos:
* Lector: Un rol con permisos limitados (en este caso, solo de lectura).
* Admin: Un rol con permisos completos, como administrador.
* Los roles en MySQL son conjuntos de privilegios que puedes asignar a los usuarios, lo que facilita la gestión de permisos a gran escala.
* De esta manera se verán los roles



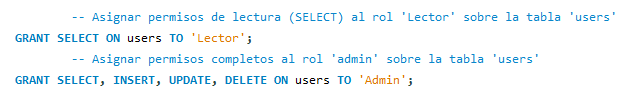


1. **Asignar Roles a Usuarios**



* Asigna el rol de Lector al usuario NicolasSalas, otorgándole permisos solo para leer datos.
* Asigna el rol de Admin al usuario EstebanPantoja, otorgándole permisos de administración.
* Consideraciones:
* Los roles permiten agrupar permisos. En este caso, NicolasSalas solo podrá realizar acciones de lectura en la base de datos, mientras que EstebanPantoja podrá realizar cualquier acción, incluida la escritura y eliminación de datos.

1. **Asignar Permisos a los Roles**



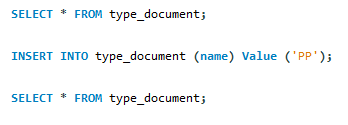
* **Para el rol Lector:** Se le otorgan permisos de solo lectura (SELECT) sobre la tabla users. Esto significa que los usuarios asignados a este rol pueden leer los datos de la tabla users, pero no pueden modificarlos.
* **Para el rol Admin:** Se le otorgan permisos completos (SELECT, INSERT, UPDATE, DELETE) sobre la tabla users. Esto permite a los usuarios asignados al rol Admin ver, insertar, actualizar y eliminar registros en la tabla users.
* **Consideraciones:**
* SELECT: Permite leer los datos de las tablas.
* INSERT: Permite agregar nuevos registros a las tablas.
* UPDATE: Permite modificar registros existentes en las tablas.
* DELETE: Permite eliminar registros de las tablas

Al asignar estos privilegios a los roles, se facilita la gestión de permisos sin necesidad de asignar permisos a cada usuario individualmente.

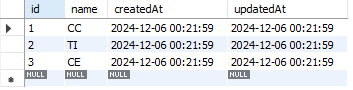
1. **Permisos de un usuario**

**Ejemplo 1:**

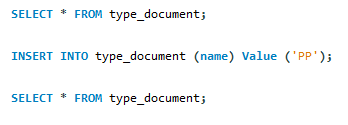
* Vamos a ver qué tipo de documentos tenemos registrados



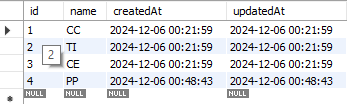
* Cómo podemos ver tenemos los siguientes tipos de documentos ya registrados



* Ahora vamos a insertar un nuevo tipo de documento



* Cómo podemos ver ahora tenemos 4 tipos de documentos agregados a nuestra base de datos



**Ejemplo 2:**

* Ahora intentaremos modificar un atributo de una tabla



* Cómo podemos ver no podemos modificar el atributo debido a que no tenemos los permisos para hacer la modificación





**Conclusiones**

El desarrollo de estas bases de datos representó un desafío significativo, pero sumamente enriquecedor para nuestro equipo. A lo largo de este proyecto, enfrentamos y superamos diversos retos, aplicando las mejores prácticas y enfoques para lograr soluciones eficientes y bien estructuradas. Este proceso no solo nos permitió consolidar nuestros conocimientos, sino que también avanzó considerablemente nuestro desarrollo profesional.

El proyecto nos brindó una valiosa experiencia en la creación y gestión de bases de datos, estableciendo una sólida base sobre la cual podemos construir y expandir en el futuro. Estamos convencidos de que los aprendizajes adquiridos y las habilidades desarrolladas durante este proyecto nos preparan adecuadamente para enfrentar futuros desafíos en el campo del desarrollo de bases de datos. Esto nos permitirá avanzar en nuestro desarrollo y llevar a cabo proyectos de mayor envergadura con éxito, especialmente en relación con nuestros futuros proyectos de grado y los proyectos personales en los que se basa este desarrollo.

**Recomendaciones**

Por el momento, se requiere un uso básico de la base de datos, lo que implica descargar los scripts necesarios para su funcionamiento y realizar pruebas. Los detalles específicos se encuentran en el repositorio de GitHub correspondiente. Para asegurar un funcionamiento adecuado, es importante descargar el último backup disponible y los scripts asociados, garantizando así que el sistema opere de manera efectiva y eficiente.