



Universidad
Nacional
de San Martín

Trabajo Práctico de Bases de Datos 2024

Docentes:

Denise Martin

Diego Mirarchi

Pablo Núñez Monzón

Integrantes:

Abelardo, Gaston Ezequiel

Dragonetti, Maria Cecilia

Jotallan Calvetti, Ruben Gabriel

Kovinchich, Mariel Nadine

Pavon Gomez, Rodrigo Nicolas

Introducción.....	1
Parte 1: Análisis y Limpieza de Datos.....	1
1.1 Inspección de la información. Organización y Normalización de Datos:.....	1
1.2 Documentación.....	1
Parte 2: Selección y Justificación del Motor de Base de Datos.....	2
2.1 Investigación y Selección de Motor Relacional:.....	2
2.2 Evaluación Comparativa con una Base de Datos No Relacional:.....	3
Parte 3: Diseño de Modelo de Datos y Creación del Esquema.....	6
3.1 Diseño del Modelo Entidad-Relación:.....	6
3. 2 Implementación del Esquema en SQL:.....	9
Parte 4: Carga y Manipulación de Datos.....	15
4.1 Carga Inicial de Datos.....	15
4.1.1 Importar los datos organizados en las tablas correspondientes.....	15
4.1.2 Verificar la integridad de los datos y la conformidad con las restricciones.....	18
4.2 Consultas Básicas y Reportes:.....	20
4.3 Consultas Avanzadas y Optimización:.....	29
Parte 5: Procedimientos, Triggers y Estrategias de Automatización.....	35
5.1 Análisis Teórico de Procedimientos Almacenados:.....	35
5.1.1 Investigación.....	35
5.1.2 Propuesta Teórica de Procedimiento.....	35
5.1.3 Documentación de procesos almacenados.....	36
5.2 Análisis Teórico de Triggers.....	37
5.2.1 Investigación.....	37
5.2.2 Propuesta de Trigger.....	37
5.3.3 Documentación de Trigger.....	37
5.3 Propuesta de Estrategias de Automatización.....	38
5.3.1 Investigación.....	38
5.3.2 Propuesta de Estrategia de Automatización.....	38
5.3.3 Documentación de las estrategias de automatización.....	41
5.4.4 Conclusión de estrategias de automatización.....	41
6. Buenas prácticas aplicadas.....	41
7. Conclusion.....	41

Introducción

Este trabajo práctico tiene como objetivo diseñar e implementar una base de datos para gestionar información académica y personal de estudiantes y docentes de manera eficiente. El proyecto abarca todas las etapas del ciclo de vida de una base de datos, desde el análisis inicial de los datos hasta la implementación y optimización. Se seleccionó MySQL 8 como motor debido a su rendimiento, escalabilidad y soporte para relaciones complejas.

El propósito principal es proporcionar una solución robusta y escalable, con consultas optimizadas y herramientas automatizadas que aseguren integridad y disponibilidad de los datos.

Parte 1: Análisis y Limpieza de Datos

1.1 Inspección de la información. Organización y Normalización de Datos:

Se Adjunta el archivo excel con los datos estandarizados y separados por columna:

<https://docs.google.com/spreadsheets/d/1JSe3YxugDEbMQJHkQ1zvH8NryUynOJGULnuM63sJTk/edit?gid=0#gid=0>

1.2 Documentación

Para la limpieza y organización de datos en primera instancia se duplicó el archivo excel entregado. Después se pasó a identificar las diferentes categorías que podían desprender del campo “Descripción”.

Se agregaron las columnas:

1. **Materias:** Se completa como texto separado por coma (,) las materias que están cursando las personas.
2. **Trabajo:** Algunas personas ingresaron su puesto de trabajo, pero no especificaron la empresa o rubro. Otros especificaron el rubro o empresa, pero no el puesto. Por lo que se optó por completarlo como SI/NO si la persona está trabajando de manera formal.
3. **Experiencia en base de datos relacionales:** Se completa con SÍ/NO. Hubo personas que expresaron “tengo conocimientos mínimos de Bases de Datos”, en esos casos se tomó la experiencia como NO.
4. **Experiencia en base de datos no relacionales:** Se completa con SÍ/NO. Solo se clasificó con SI a las personas que dijeron explícitamente haber tenido experiencia con bases no relacionales.
5. **Ubicación:** Se estandarizaron las localidades para que no se repitan, aquellas que sean de Capital Federal se agruparon dentro de CABA. Aquellos que no especificaron localidad se autocompleta con “Bs As”, ya que contemplamos que los estudiantes deben ser de la provincia para poder concurrir..
6. **Hobbie:** Los hobbies se agruparon para estandarizar. Había algunos que por ejemplo nombraban “gimnasio”, “gym”, “entrenar” y se agrupó todo dentro de Gimnasio. Lo

mismo para personas que les gustaba el manga o el anime, se agruparon dentro de la categoría MANGA/ANIME. Quienes no agregaron hobby se autocompleta con “-”

7. **Tipo Mascota:** Se completa con texto expresando el tipo de mascota y cantidades que tiene, separados por una “,”. No se especifica género. Las personas que no tienen ninguna mascota se autocompleta con “-”. Las personas que no especificaron qué tipo de mascota es, pero si la cantidad se autocompleta con “Indefinido”.
8. **Nombre Mascota:** Se completa con los nombres de las mascotas, separado por “,”. Los valores de los nombres no están ordenados. En las tablas los nombres se autocompleta con “Sin nombre”
9. **Música:** Se completa con el género o banda que la persona recomendó, las personas que no recomendaron se autocompleta con “-”
10. **Series:** Se completa con series o películas que la persona recomendó, las personas que no recomendaron se autocompleta con “-”

Además de estos ya existían ciertos campos:

1. **DNI:** Se completa con el DNI de la persona, debe ser único. Las personas que no tienen cargado el dato del DNI se autocompleta con NULL porque es un dato que no se puede inventar.
2. **Nombre y Apellido:** Se separó en 2 campos diferentes.
Nombre: Se completa con el dato de nombre de los estudiantes, aquellos que no tengan cargado este dato se autocompleta con NULL.
Apellido: Es un dato obligatorio, se completa solamente con el apellido del estudiante.
3. **Email:** Es un dato obligatorio, se completa con el dato del mail de la persona.
4. **Grupo:** Se completa con el nombre del grupo al que pertenece la persona, solo puede ser el nombre perteneciente a uno de los 9 grupos o del grupo DOCENTE.
5. **Rol:** Se completa con uno de los 5 roles que puede contener los grupos para estudiantes. El dato es obligatorio para las personas que pertenezcan a un grupo. En caso de ser docente se completa con alguno de los 3 roles posibles (Titular, ayudante primera, ayudante segunda)

Parte 2: Selección y Justificación del Motor de Base de Datos

2.1 Investigación y Selección de Motor Relacional:

- **Elección de motor:** Decidimos utilizar MySQL versión 8 como el motor de base de datos principal, porque es uno de los motores más estables y flexibles que hay. Además, es conocido por ser una opción confiable y ampliamente utilizada, lo cual se adapta bien para este proyecto.
- **Ventajas del motor seleccionado:** MySQL ofrece varias ventajas pero las principales ventajas a la hora de elegirlo para nuestro proyecto fueron las siguientes:

- Adecuación a relaciones complejas: Se destaca por su capacidad de manejar relaciones entre tablas, haciéndolo de forma eficiente, lo que para nuestro proyecto es vital para gestionar los datos, como estudiantes, materias y etc
 - Capacidad del sistema al responder a un trabajo: El motor de almacenamiento InnoDB(de código abierto que maneja MySQL) permite un rendimiento rápido incluso cuando hay grandes volúmenes de datos, lo que es vital para manejar múltiples usuarios concurrentes sin ningún problema.
- **Cualidades en términos de rendimiento, organización y escalabilidad:**
- Analizando estas 3 cualidades podemos determinar que se convierte en una opción sólida y confiable ya que facilita y organiza las consultas complejas.
- La calidad en términos de **rendimiento** se basa en poder ejecutar consultas complejas rápidamente gracias a su capacidad de optimizar operaciones, como la creación de índices y el uso de caché de consultas.
- La **organización de los datos** en tablas permite interrelacionar mediante claves foráneas, lo que garantiza que los datos estén bien estructurados, evitando redundancias y facilitando el acceso rápido a la información.
- En cuanto la **escalabilidad** es escalable tanto verticalmente (aumentando recursos en el mismo servidor) como horizontalmente (distribuyendo datos en múltiples servidores), lo que le permite adaptarse al crecimiento del sistema sin sacrificar el rendimiento.

■ **Condicionamientos o limitaciones:**

Cabe mencionar que MySQL tiene limitaciones, ya que requiere configurarlo correctamente para que todo vaya rápido, y cuando la base de datos es muy grande, puede ser un poco más pesado. Pero con mantenimiento, estos problemas se pueden manejar.

2.2 Evaluación Comparativa con una Base de Datos No Relacional:

- **Descripción Alternativa con una Base de Datos No Relacional:**
Para explorar una alternativa no relacional, tomamos como ejemplo **MongoDB**, una base de datos orientada a documentos. A diferencia de las bases de datos relacionales, **MongoDB** utiliza documentos en lugar de tablas tradicionales. Estos documentos son conjuntos de datos que pueden tener estructuras flexibles y diferentes entre sí, lo que significa que no es necesario que todos los documentos sigan el mismo formato. Esta flexibilidad permite realizar operaciones CRUD sin afectar a todos los documentos, lo que facilita la adaptación de los datos sin necesidad de reestructurar toda la base de datos.
- **Justificación de la Alternativa No Relacional:**
Para hablar de justificación tenemos que hablar de los beneficios por la cual se pone encima de la relacional y serian :

La **flexibilidad en el almacenamiento** permite manejar información que no sigue un esquema rígido, lo cual es perfecto para proyectos donde los cambios de los datos son más frecuentes.

La **escalabilidad** en MongoDB está diseñada para escalar horizontalmente, lo que significa que puede distribuir los datos entre varios servidores, lo que es ideal para manejar grandes volúmenes de datos y cargas altas sin comprometer el rendimiento.

El último beneficio que vamos a mencionar es la **agilidad en consultas simples y CRUD** este motor facilita las operaciones y es ideal para cuando se hacen consultas sencillas sobre grandes volúmenes de datos.

Si bien todo parece ser beneficioso también tenemos que hablar de las limitaciones de se basan principalmente en:

En **términos de relaciones complejas Mongo** no está diseñado para manejar eficientemente relaciones entre datos, como las que involucran claves foráneas. Esto puede complicar y hacer menos eficientes las consultas que requieren combinar múltiples entidades.

La **Integridad de Datos** no proporciona el mismo control sobre la integridad referencial que las bases de datos relacionales, lo que puede generar inconsistencias en los datos si no se manejan correctamente las actualizaciones y eliminaciones.

Las **Consultas Complejas** aunque el motor es rápido para consultas simples, no es tan eficiente para realizar operaciones más complejas que involucren filtros, agrupaciones o uniones de datos.

- **Análisis Comparativo:**

A continuación analizamos en un cuadro sinóptico la comparativa definitiva entre los 2 modelos tanto el relacional como el no relacional,

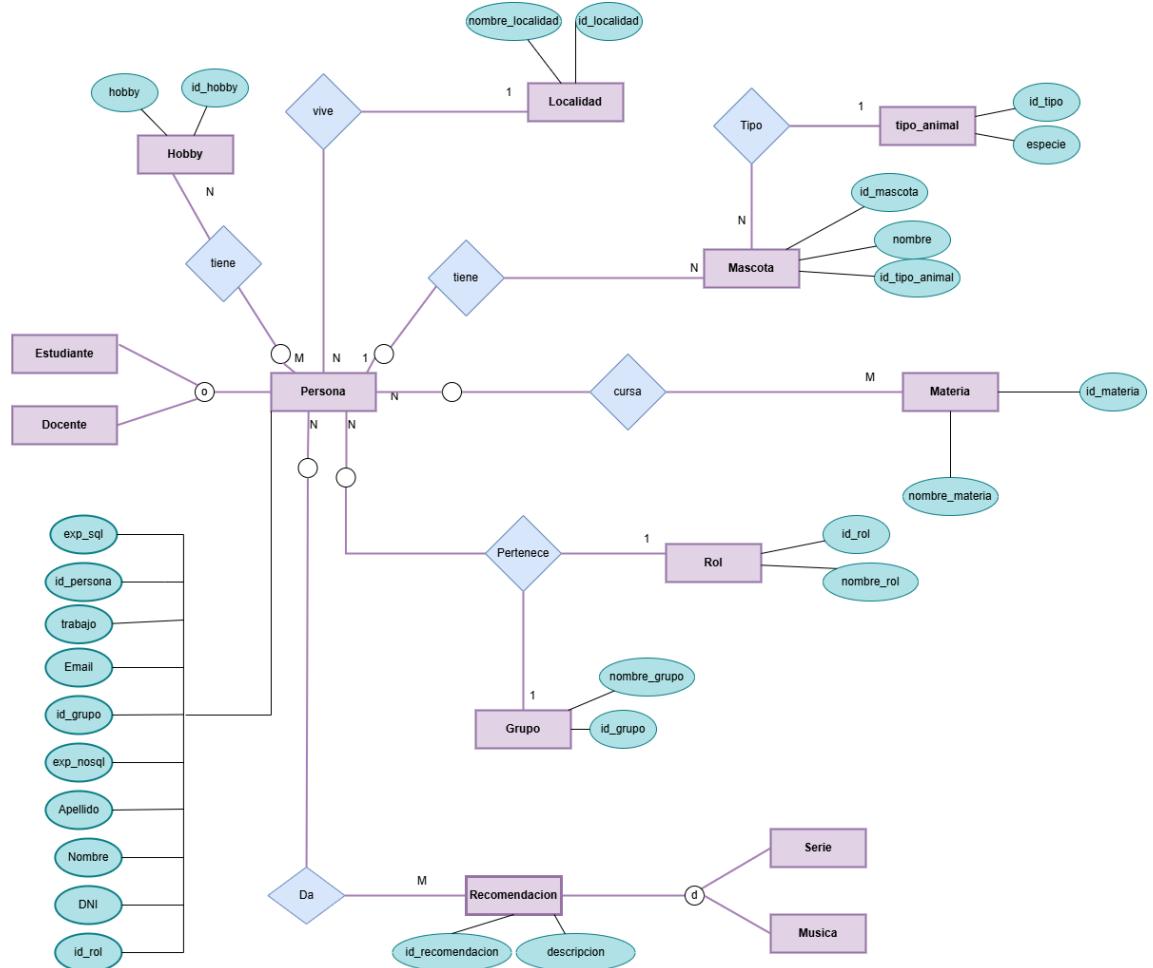
Característica	MySQL (Relacional)	MongoDB (No Relacional)
Estructura de Datos	Tablas con filas y columnas, esquema fijo.	Documentos JSON/BSON con estructura flexible.
Relaciones	Manejo de relaciones complejas entre tablas mediante claves foráneas.	No es ideal para relaciones complejas entre documentos.
Escalabilidad	Escalabilidad vertical, ideal para bases de datos de tamaño moderado.	Escalabilidad horizontal, ideal para grandes volúmenes de datos.
Flexibilidad	Requiere un esquema definido y cambios más complicados en la estructura.	Alta flexibilidad, ideal para datos que cambian con frecuencia.
Rendimiento	Buen rendimiento en datos estructurados, pero puede disminuir con volúmenes grandes.	Alto rendimiento en la gestión de grandes volúmenes de datos y cargas altas.
Consultas Complejas	Eficaz para consultas complejas que involucran múltiples tablas y relaciones.	Menos eficiente para consultas complejas que involucren relaciones entre documentos.
Integridad de Datos	Alta integridad referencial y control transaccional.	Menor integridad, especialmente en relaciones y transacciones

Ahora bien.. ¿Cuál elegimos dependiendo del enfoque?

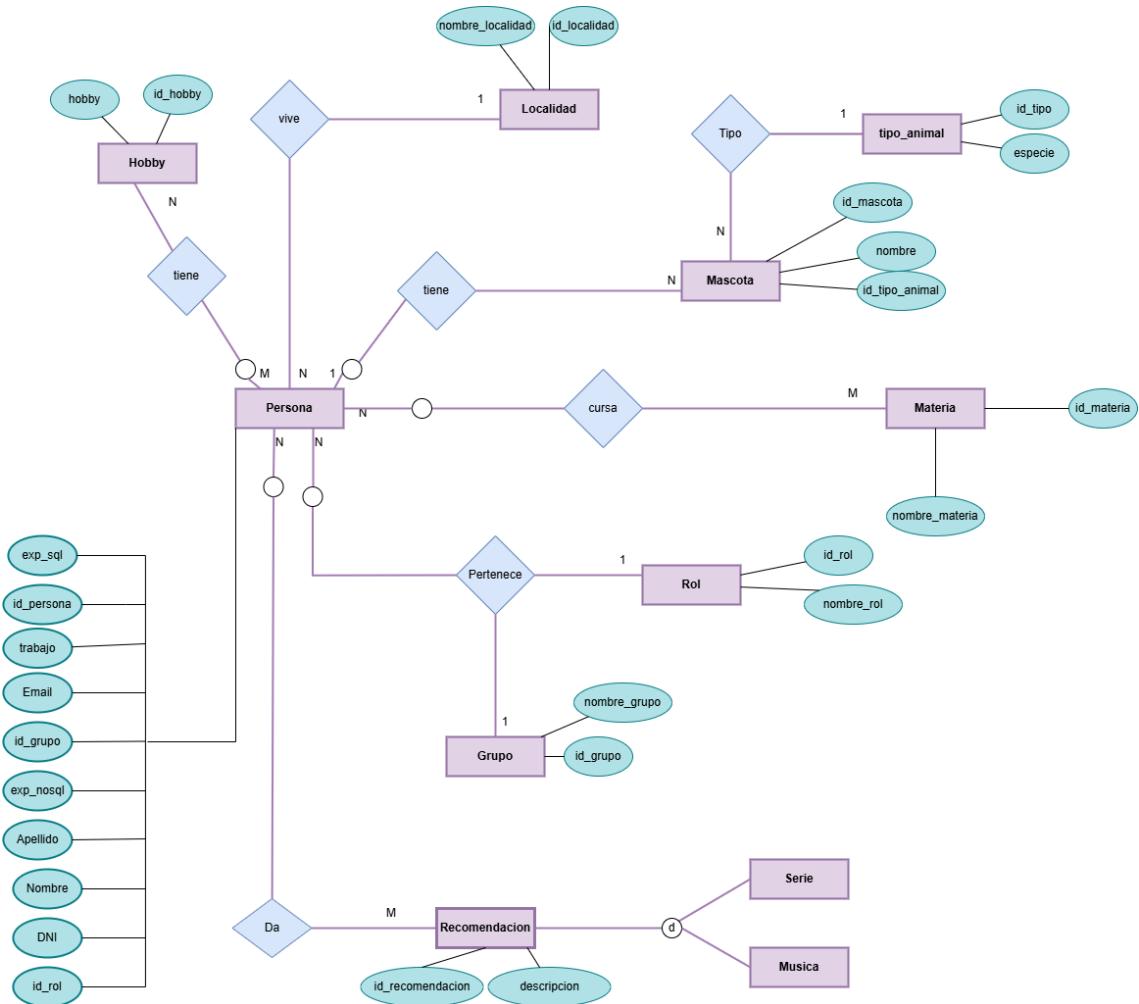
La elección del motor va a depender pura y exclusivamente de las necesidades de nuestro proyecto como vemos en nuestro caso nosotros optamos por MySQL ya que **MySQL** es más preferible tanto para proyectos de sistemas académicos como para aplicaciones que requieran datos altamente estructurados. Por otro lado, **MongoDB** se adapta mejor a aplicaciones donde la flexibilidad y la capacidad de escalar rápidamente son más importantes que la estructura rígida, como redes sociales o aplicaciones con datos que cambian constantemente.

Parte 3: Diseño de Modelo de Datos y Creación del Esquema

3.1 Diseño del Modelo Entidad-Relación:



Pensamos que esta solución del DER es la más optima, usando un overlapping para diferenciar la persona entre Docente y Estudiante. Se agrega una parcialidad en la relación entre Grupo y Rol ya que con el overlapping es suficiente para darnos cuenta que una persona es Docente. Sin embargo, decidimos armar el esquema de la base de datos teniendo en cuenta algunas diferencias con este DER, ya que requería usar técnicas para crear las tablas con overlapping que no entraba en el scope del proyecto.



Por eso decidimos hacer unos cambios para adaptarlo a nuestras necesidades y decidimos sacar el overlapping, pero mantener la parcialidad con las relaciones con Grupo y Rol para diferenciar aquellas personas que no sean docentes.

link al drawio

https://app.diagrams.net/#G1uGSfpKhFEGEyQFZ_ZsLdydbpZYRv1y4Z#%7B%22pageId%22%3A%22y4MMuLv9vLaCz474wJZ8%22%7D

Documentación por cada Entidad

- **Persona**: Son las personas tanto estudiantes como docentes, que se encuentran en nuestro sistema.

Atributos: [id_Persona](#), DNI, nombre, apellido, trabajo, email, rol, foreign key de [id_grupo](#).

- **Localidad**: Representa la ubicación de las personas, vincula a cada persona con una localidad específica.

Una persona vive en una localidad y en una localidad viven varias personas..

Atributos: nombre de la localidad, id_localidad

- **Grupo:** Son los grupos que están integrados por personas, en estos grupos las personas trabajarán durante toda la materia.

Una persona pertenece a un único grupo, pero un grupo tiene varias personas.

Atributos: id_grupo, nombre del grupo

- **Rol:** A cada persona se le asigna un rol específico dentro del grupo.

Una persona tiene un rol en un grupo determinado. Un rol tiene varias personas.

Atributos: id_rol , nombre del rol

- **Materia:** Representa las asignaturas que las personas están cursando.

Una materia puede ser cursada por varias personas. Una persona puede cursar varias materias.

Atributos: id_materia, nombre de la materia.

- **Mascota:** Son las mascotas de las personas, con atributos para su nombre y tipo.

Una persona puede tener varias mascotas. Una mascota es de una persona.

Atributos: id_mascota, nombre de la mascota, foreign key de tipo de mascota.

- **Tipo de Mascota:** Categoriza el tipo de mascota, permitiendo clasificar especies. Por ejemplo, gato, perro etc.

Una mascota es de un tipo de especie. Un tipo de mascota tiene una mascota.

Atributos: id_tipo , especie

- **Hobbie:** Representa lo que les gusta hacer en su tiempo libre a las personas.. Por ejemplo , deportes, videojuegos etc.

Una persona puede tener muchos hobbies. Un hobby lo pueden tener varias personas.

Atributos: id_hobby, hobby

- **Recomendación:** En esta entidad la persona puede dar una recomendación tanto de alguna serie o de algún grupo musical.

Una persona puede dar varias recomendaciones. Una recomendación puede ser dada por varias personas.

Atributos: [id_recomendación](#), descripción.

Justificación de las Relaciones

- Relación **Persona** con **Localidad** (Ubicación) es para saber dónde reside cada persona.
- Relación **Persona** con **Grupo** la necesitamos saber en qué grupo está cada persona.
- Relación **Persona** con **Rol** se necesita saber qué rol tiene la persona en un grupo determinado.
- Relación **Persona** con **Materia** se necesita saber las materias que está cursando el alumno.
- Relación **Persona** con **Mascota** son parte de los detalles personales de las personas, se detallan las mascotas de cada persona.
- Relación **Mascota** con **Tipo de mascota** para saber que tipo de mascota tiene la persona.
- Relación **Persona** con **Hobby**, también es un detalle personal de las personas.
- Relación **Persona** con **Recomendación**, si la persona quiere dar una recomendación ya sea de series o de música.

3. 2 Implementación del Esquema en SQL:

- Crear las tablas en SQL según el diseño E-R

Se crea la tabla Grupo la cual tiene atributos y restricciones:

PRIMARY KEY (id_grupo): Garantiza que cada grupo tenga un identificador único.

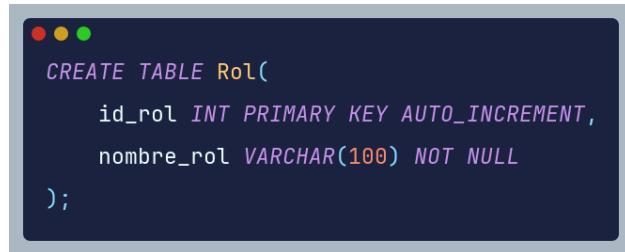
NOT NULL (nombre_grupo): El nombre del grupo es obligatorio, ya que cada grupo debe tener una identidad.

```
CREATE TABLE Grupo(
    id_grupo INT PRIMARY KEY AUTO_INCREMENT,
    nombre_grupo VARCHAR(100) NOT NULL
);
```

Se crea la tabla Rol la cual tiene atributos y restricciones:

PRIMARY KEY (id_rol): Asegura un identificador único para cada rol.

NOT NULL (nombre_rol): El nombre del rol es obligatorio, ya que define el tipo de función de la persona en el grupo.



```
CREATE TABLE Rol(
    id_rol INT PRIMARY KEY AUTO_INCREMENT,
    nombre_rol VARCHAR(100) NOT NULL
);
```

Se crea la tabla Persona la cual tiene atributos y restricciones:

PRIMARY KEY (id_persona): Garantiza que cada persona tenga un identificador único en el sistema.

UNIQUE (DNI,email): Cada persona tiene un Documento Nacional de Identidad único. Esto asegura que no haya duplicados de DNI. Y los correos electrónicos también son únicos para cada persona, lo que permite una comunicación individualizada sin duplicados.

NOT NULL (DNI, nombre, apellido): Estos campos son obligatorios, ya que representan la identidad básica de la persona.

BOOLEAN (trabajo, exp_sql, exp_nosql): Asegura que estos valores solo puedan ser TRUE o FALSE, indicando si la persona trabaja o tiene experiencia en SQL o NoSQL.

FOREIGN KEY (id_grupo) REFERENCES Grupo(id_grupo): Asegura que cada persona pertenezca a un grupo existente.

```
CREATE TABLE Persona(
    id_persona INT PRIMARY KEY AUTO_INCREMENT,
    dni INT UNIQUE DEFAULT NULL,
    nombre VARCHAR(100) DEFAULT NULL,
    apellido VARCHAR(100) NOT NULL ,
    email VARCHAR(200) NOT NULL ,
    es_docente BOOLEAN,
    fk_grupo INT DEFAULT NULL,
    FOREIGN KEY (fk_grupo) REFERENCES Grupo (id_grupo),
    fk_rol INT DEFAULT NULL,
    FOREIGN KEY (fk_rol) REFERENCES Rol (id_rol),
    fk_localidad INT DEFAULT 16,
    FOREIGN KEY (fk_localidad) REFERENCES Localidad (id_localidad),
    exp_sql BOOLEAN,
    exp_nosql BOOLEAN,
    trabajo BOOLEAN
);
```

Se crea la tabla Localidad con sus atributos y restricciones:

PRIMARY KEY (id_localidad): Cada localidad tiene un identificador único.

NOT NULL (nombre_localidad): El nombre de la localidad es obligatorio para identificar el lugar.

```
CREATE TABLE Localidad(
    id_localidad INT PRIMARY KEY AUTO_INCREMENT,
    nombre_localidad VARCHAR(100) DEFAULT 'Bs As'
);
```

Se crea la tabla Materia con sus atributos y restricciones:

PRIMARY KEY (id_materia): Cada materia tiene un identificador único.

NOT NULL (nombre_materia): El nombre de la materia es obligatorio, ya que es el nombre con el que se identifica la asignatura.

```
CREATE TABLE Materia(
    id_materia INT PRIMARY KEY AUTO_INCREMENT,
    nombre_materia VARCHAR(100) NOT NULL
);
```

Se crea la tabla TipoMascota con sus atributos y restricciones:

PRIMARY KEY (id_tipo): Cada tipo de animal tiene un identificador único.

NOT NULL (especie): La especie es obligatoria para clasificar el tipo de mascota.

```
CREATE TABLE TipoMascota(
    id_tipo INT PRIMARY KEY AUTO_INCREMENT NOT NULL ,
    especie VARCHAR(50) NOT NULL
);
```

Se crea la tabla Mascota con sus atributos y restricciones:

PRIMARY KEY (id_mascota): Asegura un identificador único para cada mascota.

NOT NULL (nombre): El nombre de la mascota es obligatorio para poder identificarla.

FOREIGN KEY (id_tipo animal) REFERENCES Tipo(id_tipo): Vincula a cada mascota con un tipo específico de animal, manteniendo la integridad de los datos y categorizando a las mascotas según su especie.

FOREIGN KEY (id_persona) REFERENCES Persona(id_persona): Establece que cada mascota pertenece a una persona específica en el sistema. Esto mantiene la integridad referencial y garantiza que cada mascota esté asociada con una persona registrada en la base de datos.

```
CREATE TABLE Mascota(
    id_mascota INT PRIMARY KEY AUTO_INCREMENT,
    fk_tipo INT NOT NULL ,
    nombre_mascota VARCHAR(100) DEFAULT 'Sin Nombre',
    FOREIGN KEY (fk_tipo) REFERENCES TipoMascota(id_tipo) ,
    id_persona INT NOT NULL ,
    FOREIGN KEY (id_persona) REFERENCES Persona(id_persona)
);
```

Se crea la tabla Hobbie con sus atributos y restricciones:

PRIMARY KEY (id_hobby): Asegura que cada hobby tenga un identificador único.

NOT NULL (hobby): El nombre del hobby es obligatorio para describir el interés de la persona.

```
CREATE TABLE Hobbie(  
    id_hobbie INT PRIMARY KEY AUTO_INCREMENT,  
    nombre_hobbie VARCHAR(50) NOT NULL  
);
```

Se crea la tabla Recomendación con sus atributos y restricciones:

PRIMARY KEY (id_reco): Asegura un identificador único para cada recomendación.

NOT NULL (tipo): El tipo de recomendación que realizó la persona.

NOT NULL (descripción): El nombre de la recomendación que realizó la persona.

```
CREATE TABLE Recomendacion(  
    id_reco INT PRIMARY KEY AUTO_INCREMENT,  
    tipo ENUM('Serie', 'Musica') NOT NULL,  
    descripcion VARCHAR(100) NOT NULL  
);
```

Se crea la tabla intermedia PersonaHobbie con sus atributos y restricciones:

PRIMARY KEY (id_persona, id_hobbie): Asegura un identificador único para cada relación.

FOREIGN KEY (id_persona) REFERENCES Persona(id_persona): Establece que cada relación persona-hobbie pertenece a una persona específica en el sistema. Esto mantiene la integridad referencial y garantiza que cada relación persona-hobbie esté asociada con una persona registrada en la base de datos.

FOREIGN KEY (id_hobbie) REFERENCES Hobbie(id_hobbie): Establece que cada relación persona-hobbie pertenece a un hobby específico en el sistema. Esto mantiene la integridad referencial y garantiza que cada relación persona-hobbie esté asociada con un hobby registrado en la base de datos.

```
CREATE TABLE PersonaHobbie(
    id_persona INT NOT NULL ,
    id_hobbie INT NOT NULL ,
    PRIMARY KEY (id_persona, id_hobbie),
    FOREIGN KEY (id_persona) REFERENCES Persona(id_persona),
    FOREIGN KEY (id_hobbie) REFERENCES Hobbie(id_hobbie)
);
```

Se crea la tabla intermedia PersonaMateria con sus atributos y restricciones:

PRIMARY KEY (id_persona, id_materia): Asegura un identificador único para cada relación.

FOREIGN KEY (id_persona) REFERENCES Persona(id_persona): Establece que cada relación persona-materia pertenece a una persona específica en el sistema. Esto mantiene la integridad referencial y garantiza que cada relación persona-materia esté asociada con una persona registrada en la base de datos.

FOREIGN KEY (id_materia) REFERENCES Materia(id_materia): Establece que cada relación persona-materia pertenece a una materia específica en el sistema. Esto mantiene la integridad referencial y garantiza que cada relación persona-materia esté asociada con una materia registrada en la base de datos.

```
CREATE TABLE PersonaMateria (
    id_persona INT NOT NULL ,
    id_materia INT NOT NULL ,
    PRIMARY KEY (id_persona, id_materia),
    FOREIGN KEY (id_persona) REFERENCES Persona(id_persona),
    FOREIGN KEY (id_materia) REFERENCES Materia(id_materia)
);
```

Se crea la tabla intermedia PersonaRecomendacion con sus atributos y restricciones:

PRIMARY KEY (id_persona, id_recomendacion): Asegura un identificador único para cada relación.

FOREIGN KEY (id_persona) REFERENCES Persona(id_persona): Establece que cada relación persona-recomendacion pertenece a una persona específica en el sistema. Esto mantiene la integridad referencial y garantiza que cada relación persona-recomendación esté asociada con una persona registrada en la base de datos.

FOREIGN KEY (id_recomendacion) REFERENCES Recomendacion(id_recomendacion): Establece que cada relación persona-recomendación pertenece a una recomendación específica en el

sistema. Esto mantiene la integridad referencial y garantiza que cada relación persona-recomendación esté asociada con una recomendación registrada en la base de datos.

```
CREATE TABLE PersonaRecomendacion (
    id_persona INT NOT NULL ,
    id_reco INT NOT NULL,
    PRIMARY KEY (id_persona, id_reco),
    FOREIGN KEY (id_persona) REFERENCES Persona(id_persona),
    FOREIGN KEY (id_reco) REFERENCES Recomendacion(id_reco)
);
```

Parte 4: Carga y Manipulación de Datos

4.1 Carga Inicial de Datos

4.1.1 Importar los datos organizados en las tablas correspondientes.

```
INSERT INTO Rol (nombre_rol)
VALUES ( nombre_rol 'Team Lider'),( nombre_rol 'Organizador'),( nombre_rol 'Representante'),
        ( nombre_rol 'Supervisor') ,( nombre_rol 'Lider Tecnico');
```

```
INSERT INTO Grupo (nombre_grupo)
VALUES ( nombre_grupo 'DataMasters'),( nombre_grupo 'nullpointer'),( nombre_grupo 'Enrutados'),
        ( nombre_grupo 'Mandarina'),( nombre_grupo 'MCTeam'),( nombre_grupo 'okupas'),
        ( nombre_grupo 'undefined'),( nombre_grupo 'DropTable'),( nombre_grupo 'DreamTeam');
```

```
INSERT INTO Localidad (nombre_localidad)
VALUES ( nombre_localidad 'San Martin'),( nombre_localidad 'CABA'),( nombre_localidad 'Villa Ballester'),
        ( nombre_localidad 'J. L. Suarez'),( nombre_localidad 'Gral. Pacheco'),( nombre_localidad 'Escobar'),
        ( nombre_localidad 'Santos Lugares'), ( nombre_localidad 'San Andres'), ( nombre_localidad 'Villa Bosch'),
        ( nombre_localidad 'El Palomar'), ( nombre_localidad 'Ciudad Jardin'), ( nombre_localidad 'Laferriere'),
        ( nombre_localidad 'Loma Hermosa'), ( nombre_localidad 'Chilavert'), ( nombre_localidad 'Florida'),
        ( nombre_localidad DEFAULT);
```

```
INSERT INTO Materia (nombre_materia)
VALUES ( nombre_materia 'Algoritmos 3'), ( nombre_materia 'CASO'),
        ( nombre_materia 'Seminario de Prog. Concurrente'), ( nombre_materia 'Bases de Datos');
```

```
INSERT INTO Hobbie (nombre_hobbie)
VALUES ( nombre_hobbie 'Tocar Guitarra'), ( nombre_hobbie 'Viajar'), ( nombre_hobbie 'Videojuegos'),
        ( nombre_hobbie 'Manga/Anime'), ( nombre_hobbie 'Jardineria'), ( nombre_hobbie 'Reparar PC'),
        ( nombre_hobbie 'Deportes'), ( nombre_hobbie 'Leer'), ( nombre_hobbie 'Cantar'),
        ( nombre_hobbie 'Correr'), ( nombre_hobbie 'Gimnasio'), ( nombre_hobbie 'Dibujar'),
        ( nombre_hobbie 'Peliculas'), ( nombre_hobbie 'Musica'), ( nombre_hobbie 'Ajedrez'),
        ( nombre_hobbie 'Series'), ( nombre_hobbie 'Twitter'), ( nombre_hobbie 'Juntadas'),
        ( nombre_hobbie 'Bicicleta'), ( nombre_hobbie 'Fotografia'), ( nombre_hobbie 'Skate'),
        ( nombre_hobbie 'Carpinteria');
```

```
INSERT INTO Recomendacion (tipo, descripcion)
VALUES ( tipo 'Serie', descripcion 'Supernatural'), ( tipo 'Serie', descripcion 'The Boys'),
        ( tipo 'Serie', descripcion 'The Bear'),( tipo 'Serie', descripcion 'Betty la Fea'),
        ( tipo 'Serie', descripcion 'Dragon Ball'), ( tipo 'Serie', descripcion 'Jojo's'),
        ( tipo 'Serie', descripcion 'The Office'), ( tipo 'Serie', descripcion 'House of the Dragons'),
        ( tipo 'Serie', descripcion 'Arcane'),( tipo 'Serie', descripcion 'Dark'),
        ( tipo 'Serie', descripcion 'Black Mirror'), ( tipo 'Serie', descripcion 'Better Call Saul'),
        ( tipo 'Musica', descripcion 'Rock'), ( tipo 'Musica', descripcion 'Bandas Sonoras'),
        ( tipo 'Musica', descripcion 'Tropical'),( tipo 'Musica', descripcion 'Variado'),
        ( tipo 'Musica', descripcion 'Cuarteto'), ( tipo 'Musica', descripcion 'Metal'),
        ( tipo 'Musica', descripcion 'Electronica'),( tipo 'Musica', descripcion 'Trap'),
        ( tipo 'Musica', descripcion 'Heavy Metal'),( tipo 'Musica', descripcion 'Pop'),
        ( tipo 'Musica', descripcion 'Rap'),( tipo 'Musica', descripcion 'Kpop'),
        ( tipo 'Musica', descripcion 'Pelea de Gallos'), ( tipo 'Musica', descripcion 'Reggae'),
        ( tipo 'Musica', descripcion 'Soul'),( tipo 'Musica', descripcion 'Alternativa');
```

```

INSERT INTO Persona (dni, nombre, apellido, email, es_docente, fk_grupo,
fk_rol, exp_sql, exp_nosql, trabajo, fk_localidad)

VALUES  ( dni 37688075,  nombre DEFAULT,  apellido 'Jotallan Calvetti',  email 'gaabicarp@gmail.com',  es_docente false,
fk_grupo 1,  fk_rol 2,  exp_sql true,  exp_nosql true,  trabajo true,  fk_localidad 1),
( dni 41472398,  nombre 'Mariel Nadine',  apellido 'Kovinchich',  email 'marielkov1998@gmail.com',  es_docente false,
fk_grupo 1,  fk_rol 3,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad 4),
( dni 38532025,  nombre 'Gaston Ezequiel',  apellido 'Abelardo',  email 'gabelardo@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 1,  fk_rol 1,  exp_sql false,  exp_nosql false,  trabajo true,  fk_localidad 1),
( dni 40663686,  nombre 'Rodrigo Nicolas',  apellido 'Pavon Gomez',  email 'rodrigopavongomez@gmail.com',  es_docente false,
fk_grupo 1,  fk_rol 5,  exp_sql false,  exp_nosql false,  trabajo true,  fk_localidad DEFAULT),
( dni 38681662,  nombre DEFAULT,  apellido 'Dragonetti',  email 'mdragonetti@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 1,  fk_rol 4,  exp_sql false,  exp_nosql false,  trabajo true,  fk_localidad 1),
( dni 44547586,  nombre 'Juan Ignacio',  apellido 'Caceffo',  email 'jicaceffo@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 2,  fk_rol 5,  exp_sql true,  exp_nosql true,  trabajo true,  fk_localidad 5),
( dni 37984582,  nombre DEFAULT,  apellido 'Gibelli',  email 'juligibelli@gmail.com',  es_docente false,
fk_grupo 2,  fk_rol 2,  exp_sql true,  exp_nosql true,  trabajo true,  fk_localidad 2),
( dni 35972489,  nombre 'Emiliano Javier',  apellido 'Nuñez',  email 'ejnunez@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 2,  fk_rol 3,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad DEFAULT),
( dni 43036494,  nombre 'Alejo',  apellido 'Menini',  email 'amenini@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 2,  fk_rol 4,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad 8),
( dni 35726283,  nombre 'Facundo',  apellido 'Barneche',  email 'fh.barneche@gmail.com',  es_docente false,
fk_grupo 2,  fk_rol 1,  exp_sql true,  exp_nosql true,  trabajo true,  fk_localidad 7),
( dni 43245286,  nombre 'Valentino',  apellido 'Bortolussi',  email 'valentino.bortolussi.lembo@gmail.com',  es_docente false,
fk_grupo 3,  fk_rol 4,  exp_sql false,  exp_nosql false,  trabajo false,  fk_localidad 6),
( dni 42647332,  nombre 'Santiago',  apellido 'Bouza',  email 'sboza@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 3,  fk_rol 5,  exp_sql false,  exp_nosql false,  trabajo false,  fk_localidad 6),
( dni 38783368,  nombre 'Cristian',  apellido 'Lomas',  email 'cristian.lomas.a@gmail.com',  es_docente false,
fk_grupo 3,  fk_rol 3,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad DEFAULT),
( dni 42472348,  nombre 'Tomás',  apellido 'Neiro',  email 'tomasneiro@hotmail.com',  es_docente false,
fk_grupo 3,  fk_rol 2,  exp_sql false,  exp_nosql false,  trabajo true,  fk_localidad 2),
( dni 37626822,  nombre DEFAULT,  apellido 'Villafañez Sobrecasa',  email 'ccvillafanezsobrecasa@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 3,  fk_rol 1,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad 2),
( dni 41067566,  nombre 'Maximiliano',  apellido 'Borrelli',  email 'maxifborrelli@gmail.com',  es_docente false,
fk_grupo 4,  fk_rol 5,  exp_sql true,  exp_nosql true,  trabajo true,  fk_localidad 3),
( dni 44792981,  nombre 'Theo',  apellido 'Harmontas Bocci',  email 'harmontastheo@gmail.com',  es_docente false,
fk_grupo 4,  fk_rol 2,  exp_sql false,  exp_nosql false,  trabajo false,  fk_localidad DEFAULT),
( dni 95822280,  nombre 'Paola',  apellido 'Toledo Contreras',  email 'ptoledocontreras@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 4,  fk_rol 3,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad 2),
( dni 35993466,  nombre 'Matias Hernan',  apellido 'Diaz',  email 'diaz.matiash@gmail.com',  es_docente false,
fk_grupo 4,  fk_rol 4,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad 7),
( dni 42321002,  nombre 'Annabella',  apellido 'Pagano',  email 'apagano@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 4,  fk_rol 1,  exp_sql true,  exp_nosql false,  trabajo true,  fk_localidad 1),
( dni 40007189,  nombre 'Gabriel',  apellido 'Tarquini',  email 'gabi.tarquini@gmail.com',  es_docente false,
fk_grupo 5,  fk_rol 5,  exp_sql false,  exp_nosql false,  trabajo true,  fk_localidad 9),
( dni 39800551,  nombre 'Valentin Pedro',  apellido 'Fucceneco',  email 'vfucceneco@estudiantes.unsam.edu.ar',  es_docente false,
fk_grupo 5,  fk_rol 2,  exp_sql false,  exp_nosql false,  trabajo true,  fk_localidad 1),
( dni 41106994,  nombre 'Agustina',  apellido 'Rey Brienza',  email 'a.reybrienza@gmail.com',  es_docente false,
fk_grupo 5,  fk_rol 3,  exp_sql false,  exp_nosql false,  trabajo true,  fk_localidad 10),

```

```

( dni 43781315, nombre 'Emiliano', apellido 'Decuzzi', email 'eadeuzzi@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 5, fk_rol 4, exp_sql true, exp_nosql false, trabajo true, fk_localidad 11),
( dni 44160355, nombre 'Tatiana', apellido 'Sabbatini', email 'tsabbatini@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 5, fk_rol 1, exp_sql false, exp_nosql false, trabajo true, fk_localidad 2),
( dni 40395042, nombre DEFAULT, apellido 'Ruina', email 'mjruina@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 6, fk_rol 5, exp_sql true, exp_nosql true, trabajo false, fk_localidad DEFAULT),
( dni 41684308, nombre 'Lautaro', apellido 'Quellar', email 'lautacuellar69@hotmail.com', es_docente false,
fk_grupo 6, fk_rol 2, exp_sql true, exp_nosql true, trabajo false, fk_localidad 13),
( dni 43017353, nombre 'Federico', apellido 'Virgilio', email 'fedevirgili00@gmail.com', es_docente false,
fk_grupo 6, fk_rol 3, exp_sql true, exp_nosql false, trabajo true, fk_localidad 1),
( dni 42291365, nombre 'Alan', apellido 'Exarchos', email 'alanexarchos@gmail.com', es_docente false,
fk_grupo 6, fk_rol 4, exp_sql true, exp_nosql false, trabajo false, fk_localidad 9),
( dni 45105469, nombre DEFAULT, apellido 'Rossi', email 'flrossi@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 6, fk_rol 1, exp_sql true, exp_nosql false, trabajo true, fk_localidad 2),
( dni DEFAULT, nombre 'Ernesto', apellido 'Davogustto', email 'ernesto.davogustto@gmail.com', es_docente false,
fk_grupo 7, fk_rol 5, exp_sql false, exp_nosql false, trabajo true, fk_localidad 2),
( dni 45174406, nombre 'Lucas Gonzalo', apellido 'Rodriguez', email 'lgrodriguez@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 7, fk_rol 3, exp_sql false, exp_nosql false, trabajo false, fk_localidad 3),
( dni 39916775, nombre 'Alan', apellido 'Guarino', email 'aguarino@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 7, fk_rol 4, exp_sql false, exp_nosql false, trabajo true, fk_localidad 3),
( dni 35766192, nombre 'Tamara Eleonor', apellido 'Mecozzi', email 'mecozzite@gmail.com', es_docente false,
fk_grupo 7, fk_rol 1, exp_sql false, exp_nosql false, trabajo true, fk_localidad 1),
( dni DEFAULT, nombre 'Agustin', apellido 'Hoj', email 'ahoj@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 7, fk_rol 2, exp_sql false, exp_nosql false, trabajo true, fk_localidad 3),
( dni 35093145, nombre 'Matias', apellido 'Caballero', email 'msebacaballero@gmail.com', es_docente false,
fk_grupo 8, fk_rol 5, exp_sql false, exp_nosql false, trabajo true, fk_localidad 12),
( dni 36594617, nombre 'David', apellido 'Pazos', email 'davidgpazos@gmail.com', es_docente false,
fk_grupo 8, fk_rol 2, exp_sql false, exp_nosql false, trabajo true, fk_localidad 2),
( dni 43441575, nombre 'Fabrizio', apellido 'Signorello', email 'fsignorello@estudiantes.undam.edu.ar', es_docente false,
fk_grupo 8, fk_rol 3, exp_sql true, exp_nosql false, trabajo false, fk_localidad 3),
( dni 42997600, nombre 'Andres Elias', apellido 'Simonini', email 'aesimonini@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 8, fk_rol 4, exp_sql false, exp_nosql false, trabajo false, fk_localidad 13),
( dni 35205248, nombre 'Emiliano', apellido 'Ferretti', email 'emieferretti@gmail.com', es_docente false,
fk_grupo 8, fk_rol 1, exp_sql false, exp_nosql false, trabajo true, fk_localidad 7),
( dni 41555134, nombre DEFAULT, apellido 'Perez', email 'amperez@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 9, fk_rol 5, exp_sql false, exp_nosql false, trabajo true, fk_localidad 3),
( dni 41548103, nombre 'Delfina', apellido 'Borrelli', email 'dborrelli@estudiantes.unsam.edu.ar', es_docente false,
fk_grupo 9, fk_rol 2, exp_sql false, exp_nosql false, trabajo true, fk_localidad DEFAULT),
( dni 42181048, nombre 'Pedro', apellido 'Geraghty', email 'pedrogeraghty02@gmail.com', es_docente false,
fk_grupo 9, fk_rol 3, exp_sql false, exp_nosql false, trabajo true, fk_localidad 2),
( dni 36791436, nombre 'Diego', apellido 'Lentz', email 'digoolentz@gmail.com', es_docente false,
fk_grupo 9, fk_rol 4, exp_sql false, exp_nosql false, trabajo true, fk_localidad 14),
( dni 44553142, nombre DEFAULT, apellido 'Pugliese', email 'valentinpugliesew@outlook.com', es_docente false,
fk_grupo 9, fk_rol 1, exp_sql true, exp_nosql false, trabajo true, fk_localidad DEFAULT),
( dni DEFAULT, nombre 'Pablo', apellido 'Nuñez Monzon', email 'pnunezmonzon@unsam.edu.ar', es_docente true,
fk_grupo DEFAULT, fk_rol DEFAULT, exp_sql true, exp_nosql true, trabajo true, fk_localidad 2),
( dni 32438510, nombre 'Denise', apellido 'Martin', email 'dmartin@unsam.edu.ar', es_docente true,
fk_grupo DEFAULT, fk_rol DEFAULT, exp_sql true, exp_nosql true, trabajo true, fk_localidad 2),
( dni 44514481, nombre 'Diego', apellido 'Mirarchi', email 'dhmirarchi@estudiantes.unsam.edu.ar', es_docente true,
fk_grupo DEFAULT, fk_rol DEFAULT, exp_sql true, exp_nosql true, trabajo true, fk_localidad 15);

```

4.1.2 Verificar la integridad de los datos y la conformidad con las restricciones.

- ❖ Comprobamos que no hay más de un DNI igual (**UNIQUE**). Esta consulta devuelve una tabla vacía que confirma la inexistencia de 2 o más DNI iguales.

```

SELECT dni, COUNT(*)
FROM Persona
WHERE dni IS NOT NULL
GROUP BY dni
HAVING COUNT(*) > 1;

```

- ❖ Comprobamos las claves foráneas en distintas tablas. Estas consultas devuelven tablas vacías y se usa para confirmar la relaciones de 1 a N entre las tablas de Persona-Grupo, Persona-Materia, Mascota-TipoMascota y Persona-Mascota.

```
SELECT p.id_persona
FROM Persona p
    LEFT JOIN Grupo g  1..n<->0..1: ON p.fk_grupo = g.id_grupo
WHERE p.fk_grupo IS NOT NULL AND g.id_grupo IS NULL;
```

```
SELECT pm.id_persona, pm.id_materia
FROM PersonaMateria pm
    LEFT JOIN Persona p  1..n<->1: ON pm.id_persona = p.id_persona
    LEFT JOIN Materia m  1..n<->1: ON pm.id_materia = m.id_materia
WHERE p.id_persona IS NULL OR m.id_materia IS NULL;
```

```
SELECT m.id_mascota
FROM Mascota m
    LEFT JOIN TipoMascota t  1..n<->1: ON m.fk_tipo = t.id_tipo
WHERE m.fk_tipo IS NOT NULL AND t.id_tipo IS NULL;
```

```
SELECT m.id_mascota
FROM Mascota m
    LEFT JOIN Persona p  1..n<->1: ON p.id_persona = m.id_persona
WHERE m.id_persona IS NULL AND p.id_persona IS NOT NULL;
```

- ❖ Comprobamos algunas de las columnas que no pueden ser nulas (**NOT NULL**) o tienen valores predeterminados. Ninguna de las tablas devuelve datos lo que confirma que no hay nulos y se usan los valores predeterminados correctamente.

```
SELECT *
FROM Persona
WHERE id_persona IS NULL
    OR apellido IS NULL
    OR email IS NULL
    OR fk_localidad IS NULL;
```

```
SELECT *
FROM Recomendacion
WHERE tipo IS NULL
    OR descripcion IS NULL;
```

```
SELECT *
FROM Mascota
WHERE id_persona IS NULL
    OR nombre_mascota IS NULL
    OR fk_tipo IS NULL;
```

```
SELECT *
FROM TiposMascota
WHERE especie IS NULL;
```

```
SELECT *
FROM Hobbie
WHERE nombre_hobbie IS NULL;
```

- ❖ Comprobamos la integridad de los datos como si se es docente tenga experiencia tanto en SQL como NoSQL. No se obtiene respuesta por lo que la integridad se mantiene.

```
SELECT id_persona
FROM Persona
WHERE es_docente = TRUE AND (exp_nosql = FALSE OR exp_sql = FALSE);
```

- ❖ Comprobamos las tablas intermedias para evitar combinaciones repetidas. Ej: que no haya una persona que esté inscripta dos o más veces en una misma materia. No se obtiene ningun datos por lo que no hay combinaciones repetidas

```
SELECT id_persona, id_hobbie, COUNT(*)
FROM PersonaHobbie
GROUP BY id_persona, id_hobbie
HAVING COUNT(*) > 1;
```

```
SELECT id_persona, id_materia, COUNT(*)
FROM PersonaMateria
GROUP BY id_persona, id_materia
HAVING COUNT(*) > 1;
```

```
SELECT id_persona, id_reco, COUNT(*)
FROM PersonaRecomendacion
GROUP BY id_persona, id_reco
HAVING COUNT(*) > 1;
```

4.2 Consultas Básicas y Reportes:

- A. Crear un reporte que muestre, por cada localidad, el hobbie predominante y la cantidad de alumnos que lo practican. Para esta consulta se aplicaron tres variantes:

```
-- A) Consulta Basica:
SELECT
    l.nombre_localidad,
    h.nombre_hobbie,
    COUNT(ph.id_persona) AS num_personas
FROM
    Persona p
        INNER JOIN Localidad l  1..n->>1: ON p.fk_localidad = l.id_localidad
        INNER JOIN PersonaHobbie ph  1->-1..n: ON p.id_persona = ph.id_persona
        INNER JOIN Hobbie h  1..n->>1: ON ph.id_hobbie = h.id_hobbie
GROUP BY
    l.nombre_localidad, h.nombre_hobbie
ORDER BY num_personas DESC
LIMIT 5;
```

Se toman los nombres de localidad y hobbie y se crea una columna que cuenta las personas. Se hace la intersección entre las 4 tablas (Persona, Localidad, PersonaHobbie y Hobbie), luego se agrupa por los nombres seleccionados y se ordena de mayor a menor cantidad de alumnos limitado solo a 5 entradas.

Resultado:

	nombre_localidad	nombre_hobbie	num_personas
1	CABA	Leer	4
2	Bs As	Gimnasio	2
3	San Martin	Viajar	2
4	CABA	Series	2
5	CABA	Peliculas	2

```

-- A) Consulta con Vista y repetición de localidades:
CREATE VIEW vw_hobbies_por_localidad AS
SELECT
    l.nombre_localidad,
    h.nombre_hobbie,
    COUNT(ph.id_persona) AS num_personas
FROM
    Persona p
        INNER JOIN Localidad l  1..n<->1: ON p.fk_localidad = l.id_localidad
        INNER JOIN PersonaHobbie ph  1<->1..n: ON p.id_persona = ph.id_persona
        INNER JOIN Hobbie h  1..n<->1: ON ph.id_hobbie = h.id_hobbie
GROUP BY
    l.nombre_localidad, h.nombre_hobbie;

SELECT * FROM vw_hobbies_por_localidad ORDER BY num_personas DESC
LIMIT 5;

```

Lo mismo que lo anterior pero se aplica la vista para agilizar el proceso, por ultimo se hace el ordenamiento de mayor a menor y se limita a 5 resultados.

Resultado:

	nombre_localidad	nombre_hobbie	num_personas
1	CABA	Leer	4
2	San Martin	Viajar	2
3	CABA	Videojuegos	2
4	Villa Ballester	Deportes	2
5	CABA	Peliculas	2

```

-- A) Consulta con Vista y sin repetición de localidades:
CREATE VIEW HobbieMasComunXLocalidad AS
SELECT
    l.id_localidad,
    h.id_hobbie AS hobbie_mas_comun,
    COUNT(ph.id_persona) AS total_alumnos,
    ROW_NUMBER() OVER (
        PARTITION BY l.id_localidad
        ORDER BY COUNT(ph.id_persona) DESC, h.nombre_hobbie
    ) AS fila
FROM Persona p
    JOIN Localidad l  1..n->1: ON p.fk_localidad = l.id_localidad
    JOIN PersonaHobbie ph  1<->1..n: ON p.id_persona = ph.id_persona
    JOIN Hobbie h  1..n->1: ON ph.id_hobbie = h.id_hobbie
GROUP BY l.id_localidad, h.id_hobbie;

SELECT
    l.nombre_localidad AS Localidad,
    h.nombre_hobbie AS Hobbie_Mas_Comun,
    hv.total_alumnos AS Total_Alumnos
FROM HobbieMasComunXLocalidad hv
    JOIN Localidad l ON hv.id_localidad = l.id_localidad
    JOIN Hobbie h ON hv.hobbie_mas_comun = h.id_hobbie
WHERE hv.fila = 1
ORDER BY hv.total_alumnos DESC;

```

Se usa una vista encapsulando la selección de id_localidad, id_hobbie, el contador de alumnos y un ROW_NUMBER() OVER que genera un número de fila para cada combinación de id_localidad y id_hobbie, reiniciando el contador para cada localidad (PARTITION BY l.id_localidad), luego se ordena por la cantidad de alumnos y nombre del hobby. Todo se engloba en fila (Número de fila en el ranking de hobbies por localidad)

Se hace la unión y a la hora de la consulta solo se toma el primero de los resultados de cada hobby por localidad ordenado por la cantidad de alumnos .

Resultado:

	Localidad	Hobbie_Mas_Comun	Total_Alumnos
1	CABA	Leer	4
2	San Martin	Viajar	2
3	Villa Ballester	Bicicleta	2
4	Santos Lugares	Musica	2
5	Bs As	Deportes	2

- B. Generar un análisis que incluya la cantidad de materias en curso por los alumnos y que detalle su experiencia previa en bases de datos (dividida entre relacional y no relacional). Top 5

```

SELECT
    p.nombre,
    p.apellido,
    COUNT(pm.id_materia) AS materias_en_curso,
    p.exp_sql,
    p.exp_nosql
FROM
    Persona p
        INNER JOIN PersonaMateria pm  1<->1..n: ON p.id_persona = pm.id_persona
GROUP BY
    p.id_persona, p.nombre, p.apellido, p.exp_sql, p.exp_nosql
ORDER BY
    materias_en_curso DESC
LIMIT 5;

```

Se toman el nombre, apellido y la experiencia SQL y NoSQL de la persona y un contador de materias , se hace la interacción Persona-PersonaMateria y se agrupa por id, nombre, apellido y la experiencia, por último se ordena por la cantidad de materias en curso en orden descendiente y limitado a 5 resultados.

Resultado:

	nombre	apellido	materias_en_curso	exp_sql	exp_nosql
1	Valentino	Bortolussi	3	0	0
2	Cristian	Lomas	3	1	0
3	<null>	Pugliese	3	1	0
4	Tomás	Neiro	3	0	0
5	Santiago	Bouza	3	0	0

C. Identificar la materia más popular en cada grupo de estudio y el porcentaje de alumnos de cada grupo que está inscrito en esas materias . Para esta consulta se plantearon tres formas:

```
-- C) Sin repetir grupos (osea, solo sale una de las materias):
SELECT nombre_grupo, nombre_materia, cantidad_alumnos_inscriptos, porcentaje_inscriptos
FROM (
    SELECT
        g.nombre_grupo,
        m.nombre_materia,
        COUNT(*) AS cantidad_alumnos_inscriptos,
        (COUNT(*) * 100.0 / (SELECT COUNT(*) FROM Persona p WHERE p.fk_grupo = g.id_grupo)) AS porcentaje_inscriptos,
        ROW_NUMBER() OVER (PARTITION BY g.nombre_grupo ORDER BY (COUNT(*) * 100.0 / (SELECT COUNT(*)
FROM Persona p
WHERE p.fk_grupo = g.id_grupo))
) DESC) as rn
FROM Persona p
INNER JOIN Grupo g 1..n->1: ON p.fk_grupo = g.id_grupo
INNER JOIN PersonaMateria pm 1<->1..n: ON p.id_persona = pm.id_persona
INNER JOIN Materia m 1..n->1: ON pm.id_materia = m.id_materia
GROUP BY g.nombre_grupo, m.nombre_materia, g.id_grupo
) ranked
WHERE rn = 1
ORDER BY nombre_grupo;
```

Se usan 2 selects, en el primero se selecciona el nombre de grupo y materia , cantidad de alumnos inscriptos y el porcentaje de inscripción

En el segundo, se usa también el grupo, materia, cantidad de alumnos inscriptos y el cálculo del porcentaje:

$$\text{Porcentaje} = \frac{\text{Total de Alumnos del Grupo}}{\text{Alumnos Inscriptos en la Materia}} \times 100$$

Se intersecciona Persona-Grupo-PersonaMateria-Materia y se agrupa por nombre de grupo y de materia y el id de grupo

Todo el segundo select se engloba en el subquery de ranked y al ejecutar solo se toma el primer resultado de cada grupo y ordenado por nombre de grupo.

Resultado:

	nombre_grupo	nombre_materia	cantidad_alumnos_inscriptos	porcentaje_inscriptos
1	DataMasters	Bases de Datos	5	100.00000
2	DreamTeam	Algoritmos 3	5	100.00000
3	DropTable	Bases de Datos	5	100.00000
4	Enrutados	Bases de Datos	5	100.00000
5	Mandarina	Bases de Datos	5	100.00000
6	MCTeam	Algoritmos 3	5	100.00000
7	nullpointer	Bases de Datos	5	100.00000
8	okupas	Seminario de Prog. Concurrente	5	100.00000
9	undefined	Algoritmos 3	5	100.00000

```

-- C) Repitiendo grupos (cada grupo tiene mas de una materia popular):
SELECT
    g.nombre_grupo AS Grupo,
    m.nombre_materia AS Materia,
    COUNT(pm.id_persona) AS Alumnos_Inscritos,
    CONCAT(ROUND((COUNT(pm.id_persona) * 100.0 / (SELECT COUNT(*)  

        FROM Persona p  

        WHERE p.fk_grupo = g.id_grupo)), 2), '%') AS Porcentaje_Grupo
FROM
    Grupo g
    JOIN
    Persona p 1<->1.n: ON g.id_grupo = p.fk_grupo
    JOIN
    PersonaMateria pm 1<->1.n: ON p.id_persona = pm.id_persona
    JOIN
    Materia m 1..n<->1: ON pm.id_materia = m.id_materia
GROUP BY
    g.id_grupo, m.id_materia
HAVING
    COUNT(pm.id_persona) = (
        SELECT MAX(total)
        FROM (
            SELECT COUNT(pm2.id_persona) AS total
            FROM Persona p2
            JOIN PersonaMateria pm2 1<->1.n: ON p2.id_persona = pm2.id_persona
            WHERE p2.fk_grupo = g.id_grupo
            GROUP BY pm2.id_materia
        ) subquery
    )
ORDER BY
    g.id_grupo;

```

Se repite el segundo select del punto anterior con la agregación del símbolo % y se hace intersección Persona-Grupo-PersonaMateria-Materia agrupando por id de grupo y materia y filtrando para seleccionar solo las materias que tienen el número máximo de alumnos inscritos dentro del grupo con la subconsulta que obtiene la cantidad de alumnos inscritos en cada materia dentro de un grupo específico y al final se ordena por el número del grupo.

Resultado:

	Grupo	Materia	Alumnos_Inscritos	Porcentaje_Grupo
1	DataMasters	Bases de Datos	5	100.00%
2	nullpointer	Bases de Datos	5	100.00%
3	Enrutados	Bases de Datos	5	100.00%
4	Mandarina	Bases de Datos	5	100.00%
5	MCTeam	Algoritmos 3	5	100.00%
6	MCTeam	Bases de Datos	5	100.00%
7	okupas	Seminario de Prog. Concurrente	5	100.00%
8	okupas	Bases de Datos	5	100.00%
9	undefined	Algoritmos 3	5	100.00%
10	undefined	Bases de Datos	5	100.00%
11	DropTable	Bases de Datos	5	100.00%
12	DreamTeam	Algoritmos 3	5	100.00%
13	DreamTeam	Bases de Datos	5	100.00%

```

-- C) Aplicando una Vista:
CREATE VIEW MateriaMasPopularPorGrupo AS
SELECT
    g.id_grupo,
    g.nombre_grupo AS Grupo,
    m.id_materia,
    m.nombre_materia AS Materia,
    COUNT(pm.id_persona) AS Cantidad_Inscriptos,
    ROUND((COUNT(pm.id_persona) /
    (SELECT COUNT(*)
     FROM Persona
     WHERE fk_grupo = g.id_grupo AND es_docente = FALSE)) * 100, 2) AS Porcentaje_Inscriptos,
    ROW_NUMBER() OVER (
        PARTITION BY g.id_grupo
        ORDER BY COUNT(pm.id_persona) DESC, m.nombre_materia ASC
    ) AS Fila
FROM Grupo g
    INNER JOIN Persona p 1<->1..n: ON g.id_grupo = p.fk_grupo
    INNER JOIN PersonaMateria pm 1<->1..n: ON p.id_persona = pm.id_persona
    INNER JOIN Materia m 1..n->1: ON pm.id_materia = m.id_materia
WHERE p.es_docente = FALSE
GROUP BY g.id_grupo, m.id_materia, g.nombre_grupo, m.nombre_materia;

SELECT
    Grupo,
    Materia,
    Cantidad_Inscriptos,
    Porcentaje_Inscriptos
FROM MateriaMasPopularPorGrupo
WHERE Fila = 1
ORDER BY Grupo;

```

Se aplica una vista con la consulta de id y nombre de grupo y materia, la cantidad de inscriptos y el cálculo de porcentaje visto anteriormente (ahora con la condición de que quita a los docentes)

Se intersecciona entre Grupo-Persona-PersonaMateria-Materia y se vuelve a aplicar la condición de que solo tome estudiantes y se agrupa en numero de grupo, id de materia y sus nombres.

Luego, en la consulta real se toma el Grupo, Materia, Cantidad_Inscriptos y Porcentaje_Inscriptos de la vista y solo se toma el primer resultado de cada grupo ordenandolo por este último

Resultado:

	Grupo	Materia	Cantidad_Inscritos	Porcentaje_Inscritos
1	DataMasters	Bases de Datos	5	100.00
2	DreamTeam	Algoritmos 3	5	100.00
3	DropTable	Bases de Datos	5	100.00
4	Enrutados	Bases de Datos	5	100.00
5	Mandarina	Bases de Datos	5	100.00
6	MCTeam	Algoritmos 3	5	100.00
7	nullpointer	Bases de Datos	5	100.00
8	okupas	Bases de Datos	5	100.00
9	undefined	Algoritmos 3	5	100.00

- D. Crear un reporte que identifique alumnos con experiencia significativa en temas de bases de datos o actividades tecnológicas, clasificándolos como posibles mentores para el resto de los compañeros. TOP 5. Para esta consulta se plantearon dos formas:

```
-- D) Repitiendo alumnos:
SELECT
    p.nombre,
    p.apellido,
    p.email,
    p.exp_sql,
    p.exp_nosql,
    m.nombre_materia
FROM
    Persona p
    JOIN
        PersonaMateria pm  1<->1.n: ON p.id_persona = pm.id_persona
    JOIN
        Materia m  1..n:<->1: ON pm.id_materia = m.id_materia
WHERE
    (p.exp_sql = TRUE OR p.exp_nosql = TRUE AND m.nombre_materia = 'Bases de Datos')
    AND p.es_docente = FALSE
ORDER BY
    (p.exp_sql + p.exp_nosql) DESC
LIMIT 5;
```

Se toma la información necesaria del alumno y el nombre de la materia , se hace la union de Persona-PersonaMateria-Materia con la condición de que tengan experiencia en alguna de los dos tipos de bases de datos, el nombre de la materia sea bases de datos y no sea un docente, ordenado por la suma de ambas experiencia para ubicar a los que tiene ambas en la cima y limitado a 5 entradas.

Resultado:

	nombre	apellido	email	exp_sql	exp_nosql	nombre_materia
1	<null>	Jotallan Calvetti	gaabicarp@gmail.com	1	1	Bases de Datos
2	Juan Ignacio	Caceffo	jicaceffo@estudiantes.unsam.edu.ar	1	1	Bases de Datos
3	<null>	Gibelli	juligibelli@gmail.com	1	1	Seminario de Prog. Concurrente
4	<null>	Gibelli	juligibelli@gmail.com	1	1	Bases de Datos
5	Facundo	Barneche	fh.barneche@gmail.com	1	1	Seminario de Prog. Concurrente

```

-- D) Sin repetir alumnos:
SELECT DISTINCT
    p.nombre,
    p.apellido,
    p.email,
    p.exp_sql,
    p.exp_nosql,
    m.nombre_materia
FROM
    Persona p
        JOIN
    PersonaMateria pm  1<->1..n: ON p.id_persona = pm.id_persona
        JOIN
    Materia m  1..n<->1: ON pm.id_materia = m.id_materia
WHERE
    m.nombre_materia = 'Bases de Datos'
    AND p.es_docente = FALSE
ORDER BY
    (p.exp_sql + p.exp_nosql) DESC
LIMIT 5;

```

Empieza tomando las mismas columnas que antes y haciendo la unión mencionada, lo único que cambia es la condición que ahora no toma la experiencia y se centra en filtrar por la materia y que no sea docente, luego se ordena y limita igual que lo anterior.

Resultado:

	nombre	apellido	email	exp_sql	exp_nosql	nombre_materia
1	<null>	Jotallan Calvetti	gaabicarp@gmail.com	1		1 Bases de Datos
2	<null>	Ruina	mjruiua@estudiantes.unsam.edu.ar	1		1 Bases de Datos
3	Maximiliano	Borrelli	maxifborrelli@gmail.com	1		1 Bases de Datos
4	Facundo	Barneche	fh.barneche@gmail.com	1		1 Bases de Datos
5	Lautaro	Cuellar	lautacuellar69@hotmail.com	1		1 Bases de Datos

4.3 Consultas Avanzadas y Optimización:

- A. Listar la cantidad de alumnos por localidad junto con el número de materias promedio en las que están inscritos, lo cual permitiría analizar la carga académica en diferentes áreas geográficas. Para esta consulta se plantearon dos formas:

```
-- 3. A) Consulta con Subquery
SELECT
    l.nombre_localidad,
    COUNT(DISTINCT p.id_persona) AS total_alumnos,
    AVG(subquery.total_materias) AS promedio_materias
FROM
    Persona p
    JOIN
        Localidad l  1..n->1: ON p.fk_localidad = l.id_localidad
    JOIN
        (
            SELECT
                pm.id_persona,
                COUNT(pm.id_materia) AS total_materias
            FROM
                PersonaMateria pm
            GROUP BY
                pm.id_persona
        ) AS subquery ON p.id_persona = subquery.id_persona
WHERE
    p.es_docente = FALSE
GROUP BY
    l.nombre_localidad
ORDER BY
    total_alumnos DESC;
```

Consultamos por nombre de localidad, conteo de alumnos y el promedio de total de materias a partir de la subconsulta, se unen Persona y Localidad y la subconsulta se toma el id de la persona y se cuentan el total de materias de la tabla intermedia PersonaMateria agrupado por el id de la persona y esta subconsulta devuelve donde los id de persona son iguales

Como condición solo se pide que no sea un docente, se agrupa por nombre de localidad y se ordena de mayor a menor el total de alumnos.

Resultado:

	nombre_localidad	total_alumnos	promedio_materias
1	CABA	9	2.2222
2	Bs As	7	2.7143
3	San Martin	7	2.0000
4	Villa Ballester	6	2.5000
5	Santos Lugares	3	2.3333
6	Escobar	2	3.0000
7	Loma Hermosa	2	2.5000
8	Villa Bosch	2	2.0000
9	Chilavert	1	2.0000
10	Ciudad Jardin	1	3.0000
11	El Palomar	1	2.0000
12	Gral. Pacheco	1	1.0000
13	J. L. Suarez	1	2.0000
14	Laferriere	1	2.0000
15	San Andres	1	1.0000

```
-- 3. A) Con Vista
CREATE VIEW MATERIASPORESTUDIANTE AS
SELECT id_persona, COUNT(id_materia) AS total_materias
FROM PersonaMateria
GROUP BY id_persona;

SELECT
    l.nombre_localidad,
    COUNT(DISTINCT p.id_persona) AS total_alumnos,
    AVG(m.total_materias) AS promedio_materias
FROM Persona p
    JOIN Localidad l  1..n<->1: ON p.fk_localidad = l.id_localidad
    JOIN MATERIASPORESTUDIANTE m ON p.id_persona = m.id_persona
WHERE p.es_docente = FALSE
GROUP BY l.nombre_localidad
ORDER BY total_alumnos DESC;
```

Se usa una vista de la consulta del id de la persona y un conteo de id de materias proveniente de PersonaMateria agrupados por el id_persona.

En la consulta básica, se usa el nombre de la localidad, el total de alumnos y el promedio de materias de Persona haciendo la unión con Localidad y la vista, teniendo en cuenta que la persona no sea docente. Esto se agrupa por el nombre de la localidad y se ordena decrecientemente con la cantidad de los alumnos.

Resultado:

	nombre_localidad	total_alumnos	promedio_materias
1	CABA	9	2.2222
2	Bs As	7	2.7143
3	San Martin	7	2.0000
4	Villa Ballester	6	2.5000
5	Santos Lugares	3	2.3333
6	Escobar	2	3.0000
7	Loma Hermosa	2	2.5000
8	Villa Bosch	2	2.0000
9	Chilavert	1	2.0000
10	Ciudad Jardin	1	3.0000
11	El Palomar	1	2.0000
12	Gral. Pacheco	1	1.0000
13	J. L. Suarez	1	2.0000
14	Laferriere	1	2.0000
15	San Andres	1	1.0000

- B. Generar un reporte que muestre qué alumnos pueden actuar como mentores de otros (match), basándose en afinidades de intereses y experiencia en temas específicos (como bases de datos). Se muestran alumnos junto a posibles mentores con experiencia en áreas similares. Para esta consulta se plantearon dos formas:

```
-- 3. B) Consulta Normal:  
SELECT DISTINCT  
    p1.apellido AS Apellido_Alumno,  
    p2.apellido AS Apellido_Mentor,  
    h.nombre_hobbie AS Interes_Comun,  
    p2.exp_sql AS Experiencia_Mentor_SQL,  
    p2.exp_nosql AS Experiencia_Mentor_NoSQL  
  
FROM  
    Persona p1  
    JOIN  
    PersonaHobbie ph1  1<->1..n: ON p1.id_persona = ph1.id_persona  
    JOIN  
    Hobbie h  1..n<->1: ON ph1.id_hobbie = h.id_hobbie  
    JOIN  
    PersonaHobbie ph2 ON h.id_hobbie = ph2.id_hobbie  
    JOIN  
    Persona p2 ON ph2.id_persona = p2.id_persona  
  
WHERE  
    p1.id_persona != p2.id_persona -- Evitar que alguien sea su propio mentor  
    AND (p2.exp_sql = TRUE OR p2.exp_nosql = TRUE) -- El mentor debe tener experiencia  
    AND p2.es_docente = FALSE  
LIMIT 100;
```

Se usa la tabla Persona para tomar los apellidos de dos tipos de alumnos, los normales y los mentores, que de estos últimos también se toma la experiencia en bases de datos relacionales y no relacionales, y los hobbies en común de ambos estudiantes(aplicando el distinct para que no se repitan los alumnos comunes).

Se unen las tablas para ambos tipos de alumnos, Persona como p1 para alumnos comunes y p2 como alumnos mentores, la tabla intermedia conectando con ambos tipos de alumnos con su hobbie y la tabla principal de Hobbie tomando el interés en común con ambos.

Las condiciones acá son que un alumno no puede ser su propio mentor, que los mentores deben tener experiencia en ambos tipos de bases de datos y que no debe ser docente, todo limitado a mostrar las primeras 100 filas para agilizar la consulta.

Resultado (TOP 5):

	Apellido_Alumno	Apellido_Mentor	Interes_Comun	Experiencia_Mentor_SQL	Experiencia_Mentor_NoSQL
1	Rey Brienza	Jotallan Calvetti	Tocar Guitarra	1	1
2	Caballero	Jotallan Calvetti	Tocar Guitarra	1	1
3	Mecoazzi	Jotallan Calvetti	Viajar	1	1
4	Narmontas Bocci	Jotallan Calvetti	Videojuegos	1	1
5	Decuzzi	Jotallan Calvetti	Videojuegos	1	1

```
-- 3. B) Con Vista
CREATE VIEW Mentores AS
SELECT id_persona, apellido, exp_sql, exp_noSQL
FROM Persona
WHERE (exp_sql = TRUE OR exp_noSQL = TRUE) AND es_docente = FALSE;

SELECT DISTINCT
    p1.apellido AS Apellido_Alumno,
    p2.apellido AS Apellido_Mentor,
    h.nombre_hobbie AS Interes_Comun,
    p2.exp_sql AS Experiencia_Mentor_SQL,
    p2.exp_noSQL AS Experiencia_Mentor_NoSQL
FROM Persona p1
    JOIN PersonaHobbie ph1  1<->1..n: ON p1.id_persona = ph1.id_persona
    JOIN Hobbie h  1..n<->1: ON ph1.id_hobbie = h.id_hobbie
    JOIN PersonaHobbie ph2 ON h.id_hobbie = ph2.id_hobbie
    JOIN Mentores p2 ON ph2.id_persona = p2.id_persona
WHERE p1.id_persona != p2.id_persona
LIMIT 100;
```

Se crea la vista de Mentores que guardará los ids, apellidos y experiencia de las Personas consideradas para el puesto con la condición de que tenga experiencia en bases de datos relational y no relational omitiendo a los docentes.

Luego, al igual que el anterior punto, se toman datos de alumnos comunes(sin repeticiones) y mentores y sus hobbies y se hace la unión igual que antes pero en lugar de tomar a los mentores como Persona, se llama a la vista mencionada que ya los trae filtrados.

Se vuelve a pedir que los alumnos mentores no pueden ser su mismo mentor y se limita a 100 resultados

Resultado (TOP 5):

	Apellido_Alumno	Apellido_Mentor	Interes_Comun	Experiencia_Mentor_SQL	Experiencia_Mentor_NoSQL
1	Rey Brienza	Jotallan Calvetti	Tocar Guitarra	1	1
2	Caballero	Jotallan Calvetti	Tocar Guitarra	1	1
3	Mecozzi	Jotallan Calvetti	Viajar	1	1
4	Narmontas Bocci	Jotallan Calvetti	Videojuegos	1	1
5	Decuzzi	Jotallan Calvetti	Videojuegos	1	1

- Las consultas hechas en este TP usan LIMIT para establecer un tope de datos a mostrar, esto agiliza las consultas obteniendo los datos con mayor velocidad.
- En el caso de índices, se utilizaron los siguientes:

➤ Índices Compuestos:

1. Este índice acelera la evaluación de las condiciones del WHERE al permitir buscar eficientemente personas con experiencia en SQL/NoSQL y que no sean docentes.

```
CREATE INDEX idx_persona_experiencia ON Persona(exp_sql, exp_nosql, es_docente);
```

2. Este índice mejora el rendimiento de las consultas que filtran o agrupan resultados según las columnas: fk_grupo, es_docente.

```
CREATE INDEX idx_persona_es_docente ON Persona(fk_grupo, es_docente);
```

➤ Indices Unicos :

1. Dado que el GROUP BY requiere agrupar por id_persona, un índice en esta columna reduce significativamente el tiempo de ejecución en grandes volúmenes de datos

```
CREATE INDEX idx_personamateria_idpersona ON PersonaMateria(id_persona);
```

2. Optimiza consultas que filtran o agrupan por la localidad asociada a la persona

```
CREATE INDEX idx_persona_fk_localidad ON Persona(fk_localidad);
```

3. Este índice facilita las consultas que buscan todos los hobbies de una persona específica en la tabla de relación PersonaHobbie.

```
CREATE INDEX idx_persona_hobbie_id_persona ON PersonaHobbie(id_persona);
```

4. Mejora las consultas que buscan todas las personas asociadas con un hobbie específico en la tabla `PersonaHobbie`.

```
CREATE INDEX idx_persona_hobbie_id_hobbie ON PersonaHobbie(id_hobbie);
```

5. Acelera las búsquedas que filtran por el nombre del hobbie.

```
CREATE INDEX idx_hobbie_nombre_hobbie ON Hobbie(nombre_hobbie);
```

6. Mejora la búsqueda por el identificador único de una localidad. Esta columna es la clave primaria y una clave foránea en `Persona`.

```
CREATE INDEX idx_localidad_id_localidad ON Localidad(id_localidad);
```

7. Agiliza las consultas que filtran o agrupan por el grupo al que pertenece una persona

```
CREATE INDEX idx_persona_fk_grupo ON Persona(fk_grupo);
```

8. Optimiza las consultas que buscan personas por su identificación (`id_persona`), que normalmente es único por registro.

```
CREATE INDEX idx_persona_id ON Persona(id_persona);
```

9. Mejora el rendimiento de consultas que filtran por `id_persona` en la tabla de relación `PersonaMateria`, que conecta personas con las materias que cursan.

```
CREATE INDEX idx_persona_materia_id_persona ON PersonaMateria(id_persona);
```

10. Optimiza las consultas que filtran o buscan todas las personas asociadas a una materia específica.

```
CREATE INDEX idx_persona_materia_id_materia ON PersonaMateria(id_materia);
```

11. Optimiza búsquedas por el identificador único de una materia (`id_materia`), que probablemente es la clave primaria de la tabla.

```
CREATE INDEX idx_materia_id_materia ON Materia(id_materia);
```

- También se utilizan vistas (`VIEW`) para crear tablas virtuales y simplificar las consultas más largas y complejas.

Parte 5: Procedimientos, Triggers y Estrategias de Automatización

5.1 Análisis Teórico de Procedimientos Almacenados:

5.1.1 Investigación

Un **proceso almacenado** (Stored Procedure) es un conjunto de declaraciones SQL que se guarda directamente en la base de datos y se puede reutilizar. El mismo acepta parámetros de entrada/salida y ejecuta las declaraciones SQL en el procedimiento, pudiendo devolver conjuntos de resultados. Además, poseen algunas ventajas importantes por ejemplo:

- Generan performance en ciertos casos.
- Otorgan un nivel de seguridad extra al limitar el acceso a cosas que puede consultar/actualizar.
- Facilitan el mantenimiento, al centralizar la lógica este mismo se podría el comportamiento de alguna operación y al modificar al mismo los cambios se aplican automáticamente.

5.1.2 Propuesta Teórica de Procedimiento

Procedimiento para registrar la inscripción de un nuevo alumno en una materia.

Pensemos un sistema de base de datos donde cada vez que un alumno se inscriba en una o más materias, se debe actualizar una tabla que almacene esta relación entre alumnos y materias. Este procedimiento almacenado podría recibir como parámetros el ID del alumno, el ID de la materia y registrar automáticamente la inscripción.

1. Parámetros:

- `id_alumno`: Identificador único del alumno.
- `id_materia`: Identificador único de la materia.

2. Acciones:

- Validar que el `id_alumno` y el `id_materia` existan en las tablas correspondientes ([Alumnos](#) y [Materias](#)).
- Registrar la inscripción en la tabla [Inscripciones](#).

3. **Beneficios:** Automatiza el proceso de inscripción, reduce errores manuales y garantiza que la fecha de inscripción se capture sin intervención humana.

Ejemplo de implementación en SQL:

```

CREATE PROCEDURE RegistrarInscripcion(
    IN id_alumno INT,
    IN id_materia INT
)
BEGIN
    -- Validar existencia del alumno y la materia
    IF EXISTS (SELECT 1 FROM Alumnos WHERE id = id_alumno) AND
        EXISTS (SELECT 1 FROM Materias WHERE id = id_materia) THEN

        -- Registrar inscripción
        INSERT INTO Inscripciones (id_alumno, id_materia, fecha_incripcion)
        VALUES (id_alumno, id_materia, fecha_incripcion NOW());

        -- Actualizar conteo de inscripciones en la materia
        UPDATE Materias
        SET numero_inscriptos = numero_inscriptos + 1
        WHERE id = id_materia;

    ELSE
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Alumno o materia no existe';
    END IF;
END;

```

5.1.3 Documentación de procesos almacenados

- **Descripción del Funcionamiento:**

Este procedimiento almacenado (Stored Procedure) , `RegistrarInscripcion`, permite registrar automáticamente la inscripción de un alumno en una o varias materias. Para ejecutar el procedimiento, se pasan dos parámetros: `id_alumno` e `id_materia`. El procedimiento realiza las siguientes acciones:

1. **Validar la Existencia:**

Verifica si el alumno y la materia existen en sus respectivas tablas (`Alumnos` y `Materias`).

2. **Registro de Inscripción:**

Si ambos registros existen, el procedimiento inserta una nueva fila en la tabla `Inscripciones`, registrando el ID del alumno, el ID de la materia y la fecha actual como `fecha_incripcion`.

- **Justificación en MySQL:**

Este motor de SQL es adecuado para implementar en este tipo de procedimientos debido a su eficacia y performance para ejecutar las tareas repetitivas y automatizar procesos, como por ejemplo las actualizaciones en campos dependientes

5.2 Análisis Teórico de Triggers

5.2.1 Investigación

Un *trigger* es un mecanismo que ejecuta una acción cuando cierto evento ocurre (*INSERT*, *UPDATE*, *DELETE*). La acción puede ser: una sentencia SQL, un bloque anónimo de código SPL (*Stored Procedures Languages*), un procedimiento almacenado o una función. Los *triggers* son útiles para mantener la integridad de los datos, para ejecutar logs de auditorías y autorizaciones de seguridad.

5.2.2 Propuesta de Trigger

Se propone un trigger llamado *LogActualizacionAlumno* que se ejecuta automáticamente cuando se **modifique la información personal de un alumno** (por ejemplo, nombre, dirección, etc.). Este trigger guardará un registro en una tabla de logs para auditorías futuras.

Condición de Activación:

- *AFTER UPDATE* en la tabla *Alumnos*.

Acciones:

- Insertar un registro en la tabla *Logs*, registrando la fecha y hora de la modificación, el ID del alumno modificado, y los valores anteriores de los campos modificados (nombre, dirección, etc.).

Ejemplo de implementación en MySQL:

```
CREATE TRIGGER LogActualizacionAlumno
AFTER UPDATE ON Alumnos
FOR EACH ROW
BEGIN
    INSERT INTO LogCambios (alumno_id, campo_modificado, valor_anterior, valor_nuevo, fecha_modificacion)
    VALUES (
        OLD.alumno_id,
        campo_modificado 'información personal',
        valor_anterior OLD.nombre, -- Puedes añadir más campos según necesidad
        valor_nuevo NEW.nombre,
        fecha_modificacion NOW()
    );
END;
```

5.3.3 Documentación de Trigger

Descripción y Funcionamiento del Trigger:

El trigger *LogActualizacionAlumno* se activa automáticamente cada vez que se actualice la información personal de un alumno en la tabla *Alumnos*. El propósito del trigger es registrar en la tabla *LogCambios* los detalles de cada cambio, incluyendo el ID del alumno, el campo modificado, el valor anterior, el nuevo valor y la fecha de modificación.

- **Cuándo se Activa:** El trigger se activa **después de una actualización** en la tabla *Alumnos*.

- **Acciones que Ejecuta:** Inserta en *LogCambios* un registro detallado de la modificación.
- **Motivación para la Gestión de la Base de Datos:** Este trigger mejora la gestión de datos al mantener un historial de cambios en la información personal de los alumnos, lo cual es útil para auditorías, trazabilidad de datos y cumplimiento de políticas de integridad. Al registrar automáticamente los cambios, se reduce la carga manual y el margen de error en auditorías.

El trigger crearía un registro en el *LOG* lo cual facilita el seguimiento y revisión de los cambios.

5.3 Propuesta de Estrategias de Automatización

5.3.1 Investigación

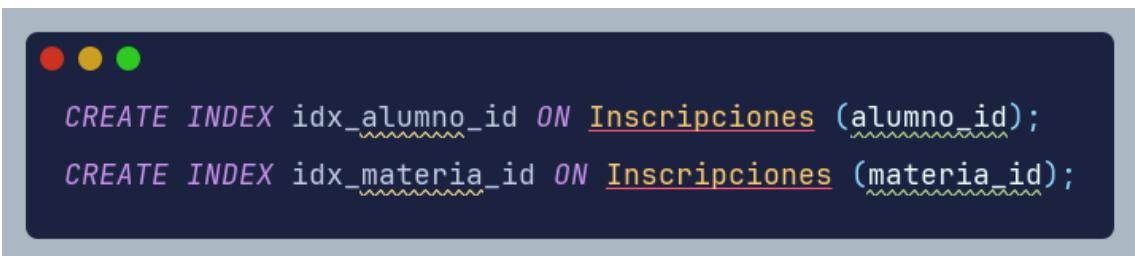
La automatización en bases de datos incluye la creación de índices, vistas y programación de tareas como backups o actualizaciones. Estas prácticas ayudan a optimizar el rendimiento de consultas y la disponibilidad de datos.

5.3.2 Propuesta de Estrategia de Automatización.

Para mejorar la eficiencia, se propone:

- **Crear índices** en los campos *alumno_id* y *materia_id* de las tablas más consultadas, como *Inscripciones* y *Alumnos*. Esto mejora significativamente la velocidad de las consultas y operaciones relacionadas para dicha tabla

Ejemplo de aplicación con MySQL:



```

CREATE INDEX idx_alumno_id ON Inscripciones (alumno_id);
CREATE INDEX idx_materia_id ON Inscripciones (materia_id);

```

- **Implementar una rutina de backups diarios** para proteger la información ante fallos o errores. Usar discos locales para asegurar una copia accesible y rápida ante errores o pérdidas de datos. Este método es ideal cuando el servidor tiene espacio suficiente y se encuentra en un entorno controlado.

Cosas a tener en cuenta:

- La ubicación del **backup** para este caso estaremos utilizando en medios locales del sistema operativo para generar el respaldo de la base de datos. Por ejemplo en Linux usar */var/backup* o windows c:/respaldo
- Algunas Recomendaciones:

1- Designar una carpeta protegida asegurando que esta misma tenga permisos.

2- Monitorear el espacio libre en el disco, ya que los backups pueden ocupar espacio con el tiempo, por lo cual se configura un sistema para borrar respaldo viejos.

3- Copias secundarias aunque se use el disco local, es aconsejable realizar copias manuales o automáticas en medios externos periódicamente. Por ejemplo usar Cron

4- Medidas de seguridad:

a) Configurar permisos de administrador para leer o escribir

b) Cifrar el archivo para proteger los datos, en caso si hay acceso para terceros en el disco

c) Verificar regularmente si los backups se está creando correctamente y que se pueda restaurar sin errores

Ejemplo de backup diario en MySQL y pasos a seguir:

1) Crear el script para el backup del disco

```
CREATE EVENT BackupDiario
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_TIMESTAMP
DO
BEGIN
    -- Mensaje de inicio
    SELECT 'Iniciando el respaldo diario...';

    -- Definir la ruta de respaldo en el disco local
    SET @backup_path = 'C:/Respaldos/nombre_base_datos.bak'; -- Cambia esta ruta según corresponda

    -- Validar si la base de datos existe
    IF NOT EXISTS (SELECT 1 FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'nombre_base_datos') THEN
        SIGNAL SQLSTATE '45000';
        SET MESSAGE_TEXT = 'Error: La base de datos no existe.';
    END IF;

    -- Realizar el respaldo (con compresión)
    BACKUP DATABASE nombre_base_datos
    TO DISK = @backup_path
    WITH COMPRESSION;

    -- Mensaje de éxito
    SELECT 'Respaldo completado exitosamente. Archivo almacenado en ', @backup_path;
END;
```

2) Configurar **cron** para ejecutar el script diario

- Abrir el crontab del usuario o equipo que tenga el backup

```
crontab -e
```

- Agregar la linea de codigo para ejecutar el script de mySQL:

```
0 2 * * * /usr/bin/mysql -u usuario -p'contraseña' -e "CREATE EVENT IF  
NOT EXISTS BackupDiario ON SCHEDULE EVERY 1 DAY STARTS  
CURRENT_TIMESTAMP DO BEGIN SELECT 'Backup iniciado'; BACKUP  
DATABASE nombre_base_datos TO  
DISK='/ruta/externa/respaldo/nombre_base_datos.bak' WITH  
COMPRESSION; END;"
```

este script se ejecutara todos los días a las 2 am

3) Medidas de seguridad:

- Restringir los permisos de la carpeta de respaldo para que solo los administradores puedan acceder a ella. por ejemplo en estos SO:

en linux : *chmod 700 /home/usuario/backups/*

en windows : darle permisos desde la propiedad de la carpeta

- Establecer procedimientos periódicos para verificar que los respaldos pueden ser restaurados correctamente. Esto puede incluir la restauración manual en un entorno de pruebas.
- Considerar usar algunas herramientas adicionales para cifrar los archivos de respaldo, especialmente si contienen información sensible. Por ejemplo usar *openssl*

- **Vistas:** Crear una vista *ResumenAlumnos* que combine información clave de los estudiantes permitiendo generar reportes eficientes y centralizados

Ejemplo de aplicación MySQL:

```
CREATE VIEW Resumen_Alumnos AS  
SELECT a.alumno_id, a.nombre, COUNT(i.materia_id) AS total_materias  
FROM Alumnos a  
JOIN Inscripciones i ON a.alumno_id = i.alumno_id  
GROUP BY a.alumno_id, a.nombre;
```

5.3.3 Documentación de las estrategias de automatización

- **Índices:** Los índices creados en campos clave como `alumno_id` y `materia_id` optimizan el rendimiento de las consultas en tablas grandes. Esto permite que las búsquedas y actualizaciones en estas tablas sean más rápidas, especialmente cuando se realizan con frecuencia.
- **Vistas:** La vista `ResumenAlumnos` facilita la consulta de datos de alumnos y materias inscritos de forma centralizada, reduciendo la carga computacional en consultas repetitivas y mejorando la performance en los reportes.
- **Backups Programados:** La automatización de backups incrementa la seguridad y disponibilidad de los datos. En un entorno real, esta medida previene pérdidas de información por errores humanos o fallos del sistema, asegurando que siempre exista una copia reciente de la base de datos.

5.4.4 Conclusión de estrategias de automatización

En conclusión, estas estrategias contribuyen a un sistema de base de datos más performance y seguro, permitiendo un acceso rápido a la información crítica y protegiendo la integridad de los datos en un entorno real. La implementación de índices y vistas reduce el tiempo de respuesta en consultas intensivas, mientras que los backups automáticos garantizan la seguridad y continuidad de la operación.

6. Buenas prácticas aplicadas.

- El uso de **Índices** tanto simples como compuestos nos ayudaron a agilizar más rápido acceder a los datos de una tabla para las consultas en sí (ver el punto 4).
- Con el uso de **Normalización** aplicamos la 1FN nos separamos en 10 columnas y estandarizamos datos y eliminamos los duplicados en las filas. Aplicamos la Segunda Forma Normal (2FN) en el trabajo práctico cuando decidimos dividir la información en tablas separadas, como Hobbie, Materias, etc. Esto se hizo para asegurarnos de que todos los atributos que no son clave dependan directamente de la clave primaria (PK) de su tabla correspondiente, eliminando dependencias parciales y mejorando la organización de los datos.
- El uso de Backup a nivel local usando disco (ver el punto 5.3.2)

7. Conclusion

En conclusión, el diseño e implementación de esta base de datos resalta la importancia de aplicar buenas prácticas, como la normalización, el uso de índices y la automatización, para garantizar un sistema eficiente, escalable y confiable.

MySQL 8 se destacó como una herramienta robusta y flexible, ideal para proyectos académicos, gracias a su capacidad para manejar relaciones complejas y optimizar el rendimiento de las consultas.

Finalmente, la implementación de reportes y análisis no sólo permitió organizar y presentar la información de manera clara, sino que también apoyó la toma de decisiones de manera más

ágil y precisa. Este proyecto dejó en evidencia cómo una base de datos bien diseñada puede transformar la forma en que se gestiona y utiliza la información.