

TP FINAL BASE DE DATOS

2024



**Universidad
Nacional
de San Martín**

Integrantes:

- Barneche, Facundo
- Menini, Alejo
- Gibelli, Julian
- Nuñez, Emiliano
- Caceffo, Juan Ignacio

Parte 1: Análisis y Limpieza de Datos

Objetivo: Asegurar que los datos proporcionados en el archivo inicial estén organizados y normalizados para su inclusión en la base de datos.

1. Análisis de los Datos Existentes:

- Inspección de la Información: Examinar los datos proporcionados en el archivo adjunto, identificando datos clave y campos incompletos.
- Organización y Normalización de Datos:
 - Estandarizar el formato de nombres, roles y grupos que aparecen en el archivo adjunto y dividir campos según sea necesario (por ejemplo, separar nombre y apellido, validar información y estandarizar los nombres de las localidades y materias, eliminar duplicados, etc).
 - Documentación Inicial: Elaborar un informe que describa
 - Pasos de limpieza y organización realizados (data preprocessing).
 - Proponer soluciones para los datos incompletos o faltantes.

1) Los datos provienen de un archivo Excel (XLSX) que contiene distintos registros de estudiantes y docentes:

- Columnas iniciales: Columna sin nombre, Nombre y Apellido, Email , Grupo, Rol, Descripción

Pasos realizados de preprocesamiento:

- El encabezado de la primer columna no tiene nombre, la renombramos como atributo **DNI** numero documento con el fin de mejorar la descriptividad del campo
- Como puede haber alumnos con el mismo documento, este no es la PK, vamos a generar una columna **ID**. Esta columna fue definida como un identificador único y autoincremental para cada registro.
- La columna "**Nombre y Apellido**" fue dividida en dos columnas separadas: **[Nombre]** y **[Apellido]**. Esta transformación asegura que se cumpla con la **Primera Forma Normal (1FN)**, en la que los atributos deben ser atómicos. El apellido fue definido como obligatorio y el nombre como opcional, de acuerdo con los requisitos del modelo de datos.
- Si el estudiante tiene un grupo es obligatorio que tenga un rol.
- **EMAIL** y **APELLIDO** son campos obligatorios que no pueden contener nulos. La columna **EMAIL** debe ser un campo único no repetible.
- Definimos que para estar en el curso el registro **debe** tener, email, apellido y estar asignado a grupo.

- Se realizó una normalización en la entidad **[Localidad]**, generalizando aquellas localidades que pertenecen a **CABA**, con el fin de no perder aquellos registros que solo se especificó que pertenecen a CABA.
- Se **crean dos columnas** en la entidad **[Integrante]** que detallan si el integrante posee conocimiento o no de base de datos relaciones como no relacionales. Las mismas tienen como data type Bool. Utiliza de base en la tabla inicial la columna descripción.
- Se **crea la columna Recomendación**, utilizando los datos de la tabla base la columna Descripción (**VALIDAR**)
- Se tomó la decisión de **generalizar los campos** de la entidad **[Mascota]** debido a que un porcentaje significativo de los registros no cuentan con el nombre de la mascota.
- En la columna **Descripción**, que originalmente contenía diversos tipos de información (como nombre de materias, trabajos, hobbies, localidades y mascotas), se decidió aplicar un proceso de normalización para descomponer los datos en entidades más específicas. Como resultado se crearon **[MATERIA], [TRABAJO], [LOCALIDAD], [HOBBY], [MASCOTA]**. Este proceso de normalización sigue principalmente la **Tercera Forma Normal (3FN)**.
- En el modelo de datos, se identificaron dos relaciones de tipo muchos a muchos: entre **[HOBBY]** e **[INTEGRANTE]**, y entre **[INTEGRANTE]** y **[MASCOTA]**. Para garantizar la unicidad de las combinaciones y evitar duplicaciones, se decidió crear **tablas intermedias** que vinculan las entidades correspondientes.

Diccionario de tablas y columnas del modelo de base de datos que adoptamos:
[Diccionario de tablas](#)

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

1. Investigación y Selección de Motor Relacional:

- Ventajas del motor seleccionado: Describir las características específicas que hacen que el motor relacional sea adecuado para este proyecto.

Motor seleccionado MySQL:

Ventajas de MySQL

- **Gratuito y de open source**

Es gratis y open source lo cual hace a MySQL la primera opción para startups y developers. Proporciona casi todas las características deseadas en cualquier servidor de bases de datos, por lo que la coherencia y el rendimiento de las aplicaciones no se ven comprometidos.

- **Amplia comunidad de soporte**

Aunque MySQL es open source, no significa que te quedes solo. Una enorme comunidad en línea está siempre a su disposición para buscar soluciones o encontrar la mejor manera de hacer las cosas.

- **Alto rendimiento, escalabilidad, flexibilidad**

MySQL está considerada como una de las bases de datos más rápidas que existen. Además, también proporciona multi-threading para conseguir un rendimiento más optimizado. Con el soporte de aplicaciones embebidas, MySQL es adecuado para diversos casos de uso.

- **Independencia de plataforma**

MySQL es independiente de la plataforma. Por lo tanto, si la aplicación se ejecuta en Linux Server o Windows Server o cualquier otro sistema operativo (UNIX, OS/2), MySQL funciona..

- Cualidades en términos de rendimiento, organización y escalabilidad: Explicar cómo el motor relacional facilita la organización de datos estructurados y consultas complejas.

Factores que pueden afectar al rendimiento de la base de datos MySQL, entre ellos:

Recursos de hardware: La CPU, la memoria y el almacenamiento pueden afectar al rendimiento de la base de datos.

Configuración de MySQL: Ajustar la configuración de MySQL puede tener un impacto significativo en el rendimiento. Los ajustes clave incluyen el tamaño del buffer pool, table_open_cache y query_cache_size.

Indexación: Una indexación adecuada puede mejorar significativamente el rendimiento de las consultas al reducir la cantidad de datos que hay que leer.

Optimización de las consultas: Escribir consultas eficientes puede ayudar a reducir la carga de trabajo de la base de datos, lo que se traduce en un mejor rendimiento.

- Condicionamientos o limitaciones: Mencionar posibles limitaciones, como requerimientos de infraestructura o restricciones de licencia que pudieran condicionar en la elección del motor.

Desventajas de MySQL

- Problemas de estabilidad

MySQL sufre de problemas de estabilidad y tiende a corromperse bajo ciertos casos de uso. Aunque esta queja no se hace en general, se han hecho quejas generales sobre la corrupción de datos al auditar o hacer transacciones.

- Bajo rendimiento en cargas altas

Mientras que MySQL es lo mejor para muchos casos de uso, para las grandes empresas que tienen millones de registros y transacciones, MySQL no es adecuado. La razón es que para volúmenes tan elevados MySQL no proporciona un soporte adecuado para operaciones de lectura/escritura. Para tales escenarios, Oracle o SQL Server ganan su parte.

1. Evaluación Comparativa con una Base de Datos No Relacional:

- Descripción Alternativa con una Base de Datos No Relacional: Proponer y describir cómo funcionaría hipotéticamente la misma solución utilizando un motor de base de datos no relacional (por ejemplo, MongoDB).

En este caso se va a proponer una solución tanto teórica como con tintes técnicos de cómo implementarla utilizando Firebase | Firestore (simil MongoDB).

En este caso, Cloud Firestore es una base de datos NoSQL de tipo cloud orientada a los documentos. A diferencia de la solución propuesta en SQL acá no hay tablas ni filas. y en su lugar, almacenaríamos los datos en documentos, que se organizan en colecciones.


Cada documento contiene un conjunto de pares clave-valor. Todos los documentos se tienen que almacenar en colecciones, y pueden contener subcolecciones y objetos anidados. Además, ambos pueden incluir campos primitivos, como strings, o tipos de objetos complejos, como listas.

En primer punto habría que dirigirse al sitio de Firebase, para dentro de Firebase Console crear un nuevo proyecto, el cual nos va a brindar credenciales para poder ser vinculado desde manera local.

Luego según la tecnología en la que se esté desarrollando habría que implementar la dependencia necesaria para cargar el SDK localmente. De esta manera y gracias a las credenciales provistas desde el sitio podremos conectarnos a nuestro proyecto inicializado de manera cloud.

Documentos


Como se mencionó previamente, en Cloud Firestore, la unidad de almacenamiento es el documento. Un documento es un registro liviano que contiene campos con valores asignados. Cada documento se identifica con un nombre.


 julianGibelli
nombre:
pila: "Julian"
apellido : "Gibelli"
mail : juligibelli@gmail.com


Estos documentos son muy similares a JSON; de hecho, básicamente son JSON. Existen algunas diferencias sutiles como capacidad máxima pero pueden ser tratados como tal.

Colecciones:

Los documentos viven en colecciones, que simplemente son contenedores de documentos. Por ejemplo, podríamos tener una colección llamada **estudiantes** con los distintos estudiantes relevados, en la que haya un documento que represente a cada uno:

 estudiantes

 julianGibelli
nombre:
pila: "Julian"
apellido : "Gibelli"
mail : juligibelli@gmail.com
grupo: 2 - nullpointer
rol: organizador

 emilianoNuñez
nombre:
pila: "Emiliano"
apellido : "Nuñez"
mail : emi.nuñez@asd.com
grupo: 2 - nullpointer
dni: 3333333333

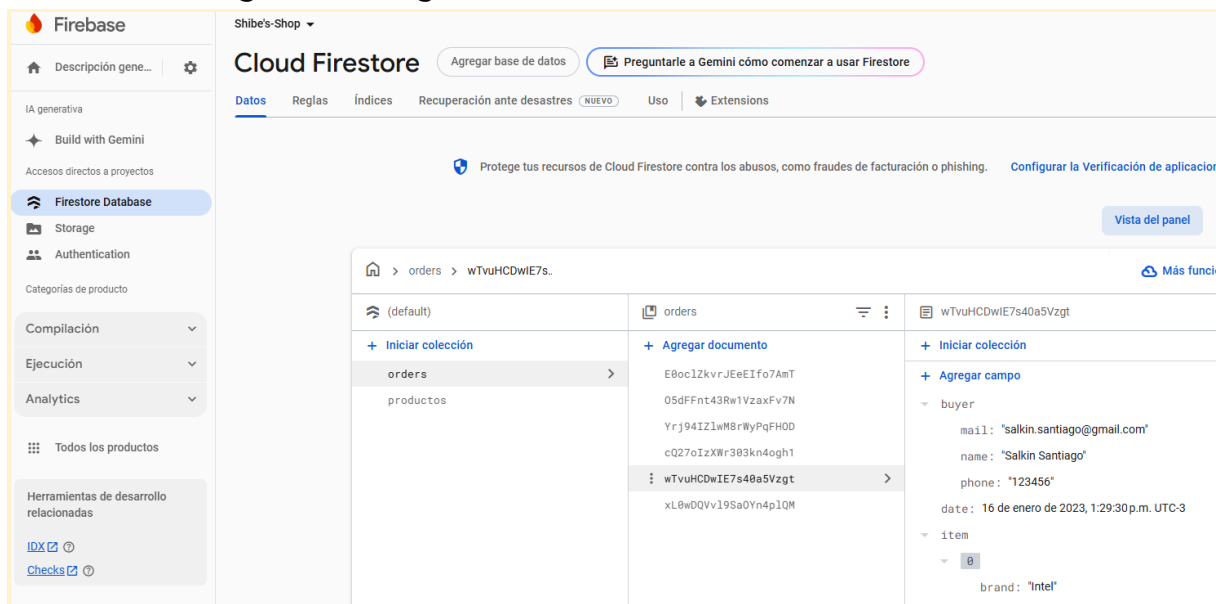
Cloud Firestore no usa esquemas como si lo hace SQL, por lo que hay libertad total sobre los campos que se ponen en cada documento y los tipos de datos a

guardar en esos campos. Los documentos dentro de una misma colección pueden contener campos diferentes o almacenar distintos tipos de datos en esos campos, lo cual es cómodo para trabajar con ciertos campos nulos o faltantes en el relevamiento/limpieza de los datos.

Los nombres de documentos dentro de una colección son únicos, por lo que en nuestro ejemplo, sería práctico usar los nombre y apellido de los estudiantes como identificador único de los documentos en la colección **estudiantes**.

Ahora bien, ¿cómo haríamos para popular nuestra nueva base de datos no relacional?

Podríamos hacerlo directamente desde el panel web de firebase como se muestra en la siguiente imagen:



(imagen ilustrativa para ver colecciones - documentos - campos)

o bien desde manera local trabajar con objetos de tipo clave-valor como mapas para luego vinculandonos mediante el SDK provisto previamente tanto crear la colección como insertar los documentos.

```
// Creamos un Map para guardar la info que queremos setear
Map<String, Object> docData = new HashMap<>();
docData.put("nombre", "Julian");
docData.put("apellido", "Gibelli");
docData.put("mail", "juligibelli@gmail.com");
docData.put("grupo", "2 - nullpointer");
// agregamos este nuevo documento (asynchronously) a la colección de
estudiantes
ApiFuture<WriteResult> future =
```

```
db.collection("estudiantes").document("julianGibelli").set(docData);  
// ...
```

- Justificación de la Alternativa No Relacional: Explicar los beneficios y limitaciones de una base de datos no relacional en este contexto, considerando aspectos como flexibilidad en el almacenamiento, escalabilidad, y tipos de consulta.

Uno de los beneficios significativos de NoSQL es que no es necesario definir un esquema por adelantado como si tuvimos que hacer en SQL. Esto facilita la adición de nuevas columnas sin lidiar con todos los problemas involucrados en la alteración de una tabla extensa con mucha información ya almacenada. También significa que, si las consultas que necesitemos hacer no requieren SQL, se puede evitar el trabajo de análisis y declaraciones SQL complejas, lo que te proporciona un gran rendimiento al tratar con grandes cantidades de datos. Sin embargo, NoSQL todavía es algo relativamente joven en comparación a SQL.

Posibles ventajas de NoSQL:

- Esquema flexible
- Utilizable en plataformas de infraestructura distribuida
- Infraestructura de bajo costo
- Alta disponibilidad y rendimiento

Posibles desventajas de NoSQL:

- Tecnología menos madura y difícil de gestionar
- Capacidades de consulta limitadas
- Inconsistencia de datos y bajo rendimiento en algunos escenarios complejos

En este contexto, como se mencionó previamente una limitante podría ser las capacidades de consultas. A diferencia de SQL no cuenta con la potencia y alcance como tal.

Por ejemplo, firebase presenta las siguientes limitaciones a la hora de hacer consultas:

- Cloud Firestore proporciona compatibilidad con las consultas OR lógicas a través de los operadores or, in y array-contains-any. Estas consultas se limitan a 30 disyunciones.

- En una consulta compuesta, las comparaciones de intervalo (<, <=, > y >=) y desigualdad (!= y not-in) deben aplicar filtros en el mismo campo.
- Puedes usar como máximo una cláusula array-contains por disyunción (grupo or).
- No puedes combinar array-contains con array-contains-any en la misma disyunción.
- No puedes combinar not-in con in, array-contains-any o or en la misma consulta.
- Solo se permite un not-in o != por consulta.
- not-in admite hasta 10 valores de comparación.
- La suma de filtros, órdenes y la ruta del documento principal (1 para una subcolección o 0 para una colección raíz) en una consulta no puede ser mayor que 100. Esto se calcula en función del formato disyuntivo normal de la consulta.
- Una consulta con un filtro de desigualdad en un campo implica ordenar según ese campo y filtra la existencia de ese campo.

Mas info:

<https://firebase.google.com/docs/firestore/query-data/get-data?hl=es-419>

A nivel escalabilidad es una ventaja ya que ofrecen escalabilidad horizontal, lo que significa que simplemente se necesitan agregar más servidores para aumentar la carga de datos. Esto quiere decir que las bases de datos NoSQL son mejores para las infraestructuras modernas basadas en la nube, que ofrecen recursos distribuidos.

■ **Análisis Comparativo:** Resumir las ventajas y desventajas de ambos enfoques (relacional vs. no relacional), proporcionando una visión general de cuándo y por qué podría ser preferible cada uno para esta solución.

Una base de datos relacional como SQL es una muy buena opción si estamos buscando construir una aplicación estructurada en torno a la relación entre tablas. SQL también es buena opción cuando queremos garantizar que los datos sean consistentes en todas las tablas. Sin embargo, las bases de datos relacionales no siempre son la mejor opción en términos de flexibilidad o escalabilidad ya que escalan de manera vertical.

Una base de datos NoSQL no relacional no utiliza tablas estructuradas, sino que utiliza esquemas flexibles para el almacenamiento de datos no estructurados. Esto nos brinda más capacidad para escalar proyectos según sea necesario. Sin embargo, también significa que tendríamos menos control sobre la consistencia y las relaciones de datos.

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

1. Diseño del Modelo Entidad-Relación:

- Crear un diagrama E-R que capture la información organizada en entidades como alumnos, materias, hobbies, ubicación y cualquier otra entidad que considere necesaria.

<https://drive.google.com/file/d/1FABqEWOxsPGhYw03NbrgarIXuxTWmPIW/view?usp=sharing>

- Documentar y justificar cada entidad y relación en el modelo, asegurándose de alcanzar la tercera forma normal (3NF), de ser posible.

Entidades:

- INTEGRANTE : Representa a cada persona ya sea docente, alumno u otro rol. El cual cuenta con atributos como el DNI, Nombre, Apellido, Email y Recomendación.
- DOCENTE : Específica a los integrantes que cumplen el rol de docente, está vinculada con "Materia" porque los docentes son responsables de impartir las materias.
- ESTUDIANTE : Representa a los integrantes que cumplen el rol de estudiantes. Incluye atributos como ID Relacional y DB Nacional, lo que indica si el mismo tiene conocimiento en base de datos relacionales y no relacionales.
- MATERIA: Representa a las asignaturas que forman parte del sistema de la Universidad.
- GRUPO: Representa el grupo al que pertenecen solamente los estudiantes.
- ROL : Representa qué rol poseen solamente los estudiantes dentro de cada grupo.
- MASCOTA: Representa a qué tipo de mascota puede tener un integrante.
- LOCALIDAD : Representa a qué tipo de localidad vive cada integrante.
- HOBBY: Representa intereses personales propios de los integrantes.
- TRABAJO : Asocia a los integrantes con ocupaciones laborales.

Cómo se cumple la 3FN en nuestro modelo:

- **División en tablas independientes (entidades):**
 - Cada entidad ahora tiene su propia tabla (INTEGRANTE, DOCENTE, ESTUDIANTE, MASCOTA, LOCALIDAD, etc.).
- **Separación de atributos que no dependen directamente de la clave primaria:**
 - Ejemplo: Los hobbies y localidades no dependen directamente de los datos del integrante (DNI, Nombre, etc.). Estos atributos se trasladan a tablas independientes (HOBBY, LOCALIDAD) y se relacionan con los integrantes a través de claves foráneas.
- **Eliminación de dependencias transitivas:**
 - Si un docente imparte materias, esa relación se modela con una tabla independiente (DOCENTE - MATERIA). No hay atributos de materias dentro de la tabla de DOCENTES o viceversa.
 - Lo mismo ocurre con la relación ESTUDIANTE - GRUPO - ROL: se separa en una tabla para manejar las combinaciones específicas de estudiantes, grupos y roles.

Relaciones:

INTEGRANTE - MASCOTA : Un integrante tiene mascota.

- Tipo de relación : Binaria
- Parcialidad : Hay parcialidad en integrante, debido a que el integrante puede no tener mascotas.
- Cardinalidad : Muchos a Muchos (N a N).
 1. Un integrante puede tener más de un tipo de mascota.
 2. Un tipo de mascota puede estar asociado con muchos integrantes.

INTEGRANTE - LOCALIDAD : Un integrante tiene localidad.

- Tipo de relación : Binaria
- Parcialidad : Hay parcialidad en integrante, debido a que el integrante puede no pertenecer a ninguna localidad.
- Cardinalidad: Uno a Muchos (1 a N).
 1. Un integrante solo puede estar asociado con una localidad.
 2. Una localidad puede estar asociada con varios integrantes.

INTEGRANTE - HOBBY : Un integrante tiene hobby.

- Tipo de relación : Binaria.
- Parcialidad : Hay parcialidad en integrante, debido a que un integrante puede no tener ningún hobby.
- Cardinalidad : Muchos a Muchos (N a N).
 1. Un integrante puede tener más de un hobby.
 2. Un hobby puede pertenecer a muchos integrantes.

INTEGRANTE - TRABAJO : Un integrante tiene trabajo.

- Tipo de relación : Binaria.
- Parcialidad: Hay parcialidad en integrante, debido a que un integrante puede no tener trabajo
- Cardinalidad : Uno a Muchos (1 a N).
 1. Un integrante solo puede tener un trabajo.
 2. Un trabajo puede estar asociado con varios integrantes.

DOCENTE - MATERIA : Un docente imparte la materia.

- Tipo de relación : Binaria.
- Parcialidad : Hay parcialidad en docente y materia, debido a que un docente puede no impartir materias y una materia puede ser impartida por ningún docente.
- Cardinalidad: Muchos a Muchos (N a N).
 1. Un docente puede impartir varias materias.
 2. Una materia puede ser impartida por varios docentes.

ALUMNO - MATERIA : Un alumno cursa materia

- Tipo de relación : Binaria.
- Parcialidad : Hay parcialidad en docente y materia, debido a que un alumno puede no cursar materias y además, una materia puede ser cursada por ningún alumno.
- Cardinalidad: Muchos a Muchos (N a N).
 1. Un alumno puede cursar varias materias.
 2. Una materia puede ser cursada por varios alumnos.

ESTUDIANTE - GRUPO - ROL : Un estudiante tiene un grupo y un rol

- Tipo de relación: Ternaria , Un estudiante pertenece a un grupo con un rol asociado. Cada instancia de la relación une un estudiante específico con un grupo y un rol concreto
- Parcialidad: No hay parcialidad en ninguna entidad:
 1. Todo estudiante debe tener un grupo y un rol.
 2. Todo grupo debe tener al menos un estudiante con un rol.
 3. Todo rol debe estar asignado a algún estudiante dentro de un grupo.
- Cardinalidad :
 - a. **Cada instancia del estudiante** está asociada con una única instancia de un grupo y un rol.
 - b. **Cada instancia del grupo** tiene exactamente un estudiante con un único rol.
 - c. **Cada instancia del rol** solo está asociada a un estudiante en un grupo.

INTEGRANTE - DOCENTE Y ESTUDIANTE

- Descripción: Un INTEGRANTE puede asumir el rol de DOCENTE, el rol de ESTUDIANTE, o ambos simultáneamente.
- Tipo de relación: **Superposición** (*overlapping*), ya que un integrante puede ser únicamente docente, únicamente estudiante o desempeñar ambos roles a la vez.

Parte 4: Carga y Manipulación de Datos

Objetivo: Cargar los datos en el esquema creado y desarrollar consultas avanzadas para extraer información relevante.

1. Carga Inicial de Datos:

- Importar los datos organizados en las tablas correspondientes.
- Verificar la integridad de los datos y la conformidad con las restricciones.

Consultas Básicas y Reportes:

En esta sección abordaremos la ejecución y explicación de las consultas y procedimientos llevados a cabo para obtener los siguientes resultados.

Inicialmente se necesitó un reporte el cual muestre el Hobby con mas Alumnos que lo realicen por Localidad.

Para lograr este reporte se atacó a la tabla HOBBY_INTEGRANTE, la cuál se utilizó como nexo para unir las tablas de INTEGRANTE, HOBBY y LOCALIDAD y lograr el resultado esperado.

Si bien contábamos con la Localidad y el nombre de los Hobbies, se necesitó realizar un conteo del Total de alumnos y con la ayuda de `ROW_NUMBER` generamos particiones para que por cada localidad se pueda obtener ordenado de manera Descendente (del más alto al más bajo) todos los alumnos que practican un respectivo Hobby (desempatan por nombre del hobby). Para finalizar filtramos por `rn = 1` que lo que hace es brindarnos el número de fila de cada partición más alto.

```
-- a) Crear un reporte que muestre, por cada localidad, el hobby
-- predominante y la cantidad de alumnos que lo practican.
SELECT Localidad, Hobby_mas_comun, Total_alumnos FROM (
  SELECT
    l.nombre AS Localidad,
    h.nombre AS Hobby_mas_comun,
    COUNT(hi.id_hobby) AS Total_alumnos,
    ROW_NUMBER() OVER (PARTITION BY l.nombre ORDER BY
COUNT(hi.id_hobby)DESC, h.nombre ) AS rn
  FROM HOBBY_INTEGRANTE hi
  JOIN INTEGRANTE i ON hi.id_integrante = i.id_integrante
  JOIN HOBBY h ON hi.id_hobby = h.id_hobby
  JOIN LOCALIDAD l ON i.id_localidad = l.id_localidad
  GROUP BY l.nombre, h.nombre
) sub WHERE rn = 1;
```

Se logra ver en el reporte las localidades ordenadas alfabéticamente, su total de alumnos que realizan el respectivo hobby y dicho hobby.

#	Localidad	Hobby_mas_comun	Total_alumnos
1	CABA	Deportes	6
2	Chilavert	Deportes	1
3	Ciudad Jardin	Deportes	1
4	El Palomar	Deportes	1
5	Escobar	Deportes	1
6	Florida	Deportes	1
7	Jose Leon Suarez	Lectura	1
8	Laferrere	Musica	1
9	Loma Hermosa	Deportes	1
10	Pacheco	Deportes	1
11	San Andres	Deportes	1
12	San Martin	Musica	3
13	Santos Lugares	Deportes	2
14	Villa Ballester	Deportes	3
15	Villa Bosch	Deportes	1

Este análisis se abordó directamente desde la tabla ESTUDIANTE_MATERIA y así poder unir las tablas INTEGRANTE y ESTUDIANTE, necesarias para obtener la información solicitada.

Se utilizó el apellido de los alumnos, se realizó un conteo de las materias y la experiencia en base de datos relacional y no relacional de ESTUDIANTE, agrupados por los últimos mencionados y ordenados por id_integrante Ascendente con un limit = 5 para obtener el TOP 5.

```
-- 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
i.apellido AS Alumno,
COUNT(em.id_materia) AS 'Materias_en_curso',
IF(e.db_no_relacional = 1, 'Sí', 'No') AS db_no_relacional,
IF(e.db_relacional = 1, 'Sí', 'No') AS db_relacional
FROM ESTUDIANTE_MATERIA em
JOIN INTEGRANTE i ON i.id_integrante = em.id_estudiante
JOIN ESTUDIANTE e ON e.id_integrante = em.id_estudiante
GROUP BY em.id_estudiante, i.apellido, e.db_no_relacional, e.db_relacional
ORDER BY i.id_integrante ASC
LIMIT 5;
```

Se logra ver que el reporte muestra la cantidad de materias cursadas por un alumno y su experiencia a lo que bases datos se refiere.

#	Alumno	Materias_en_curso	db_no_relacional	db_relacional
1	Calvetti	1	Sí	Sí
2	Kovinchich	2	No	Sí
3	Abelardo	2	No	No
4	Pavon Gomez	2	No	No
5	Dragonetti	2	No	No

En este punto se necesitó abordar el punto desde la tabla ESTUDIANTE_GRUPO_ROL y así lograr unir gracias a las interrelaciones a las tablas ESTUDIANTE, GRUPO, ESTUDIANTE_MATERIA y MATERIA.

Se realizó un conteo de alumnos y se obtuvo el porcentaje de grupo respecto a la materia (conteo de alumnos * 20, qué es lo mismo que el conteo de alumnos por 100 / 5, siendo 5 el total de alumnos por grupo). Por otro lado, nuevamente con ROW_NUMBER se obtiene en forma particionada todos los valores por materia, filtrando estas por la de más porcentaje con la sentencia rn = 1

```
-- 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.
SELECT nombre_materia, nombre_grupo, total_alumno, porcentaje_grupo
FROM (
    SELECT m.nombre AS nombre_materia, g.nombre AS nombre_grupo,
    COUNT(i.id_integrante) as total_alumno,
    CONCAT(COUNT(i.id_integrante) * 20, '%') as porcentaje_grupo,
    ROW_NUMBER() OVER (PARTITION BY g.nombre ORDER BY
COUNT(i.id_integrante) DESC, m.nombre) AS rn
    FROM ESTUDIANTE_GRUPO_ROL egr
    JOIN ESTUDIANTE i ON (egr.id_estudiante = i.id_integrante)
    JOIN GRUPO g ON (egr.id_grupo = g.id_grupo)
    JOIN ESTUDIANTE_MATERIA em ON (i.id_integrante = em.id_estudiante)
    JOIN MATERIA m ON (em.id_materia = m.id_materia)
    GROUP BY m.nombre, g.nombre
) sub WHERE rn = 1;
```

El reporte muestra todos los grupos con el total de alumnos con mayor porcentaje cursando la materia mas popular, este reporte tenia como particularidad que al cursar todos bases de datos siempre obtiene el 100%, al menos que empate con otra materia, como por ejemplo Algoritmos III, eligiendo esta última ya que desempata alfabéticamente por nombre de materia.

#	nombre_materia	nombre_grupo	total_alumno	porcentaje_grupo
1	Bases de Datos	datamasters	5	100%
2	Algoritmos III	DreamTeam	5	100%
3	Bases de Datos	DropTable	5	100%
4	Bases de Datos	Enrutados	5	100%
5	Bases de Datos	Mandarina	5	100%
6	Algoritmos III	MCTeam	5	100%
7	Bases de Datos	nullpointer	5	100%
8	Bases de Datos	okupas	5	100%
9	Algoritmos III	undefined	5	100%

El último reporte de consultas básicas se abordó desde la tabla INTEGRANTE, y uniendo las tablas ESTUDIANTE y TRABAJO.

Utilizando el apellido del alumno, la experiencia en bases de datos relacionales y no relacionales y su respectivo trabajo (Actividad) se realizó un TOP 5 de alumnos con experiencia tanto en relacional como no relacional o que trabaje en sectores como "Backend", "IT", "Data analytics", "Desarrollador aplicaciones". Esto se pensó de esta manera ya que el problema pide experiencia significativa en bases de

datos o en actividades tecnológicas pero también se podría pensar en reemplazar el **OR** por el **AND** para otro reporte de valor.

```
-- 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
SELECT i.apellido AS Alumno,
IF(e.db_no_relacional = 1, 'Sí', 'No') AS "Experiencia BD Relacional",
IF(e.db_relacional = 1, 'Sí', 'No') AS "Experiencia BD No Relacional",
t.nombre AS Actividad
FROM INTEGRANTE AS i
JOIN ESTUDIANTE AS e ON (e.id_integrante = i.id_integrante)
JOIN TRABAJO AS t ON (t.id_trabajo = i.id_trabajo)
WHERE e.db_no_relacional = 1
AND e.db_relacional = 1
OR t.nombre in ("Backend", "IT", "Data analytics", "Desarrollador aplicaciones")
LIMIT 5;
```

En el reporte se muestra un TOP 5 de alumnos con experiencia en base de datos relacional y no relacional o su actividad principal dentro del mundo de la tecnología.

#	Alumno	Experiencia BD Relacional	Experiencia BD No Relacional	Actividad
1	Gibelli	Sí	Sí	Backend
2	Villafañez S...	No	Sí	Backend
3	Borrelli	Sí	Sí	Backend
4	Toledo Con...	No	Sí	Backend
5	Signorello	No	Sí	Backend

2. Consultas Avanzadas y Optimización:

En esta sección de consultas avanzadas, se necesitó listar la cantidad de alumnos por localidad junto con el promedio de materias cursadas del grupo de alumnos de dicha localidad.

Se pensó realizar las uniones desde INTEGRANTE de las tablas ESTUDIANTE, ESTUDIANTE_MATERIA y LOCALIDAD.

Contando con esta información, se utilizó el nombre de la localidad, un promedio de materias (redondeando en 2 decimales la división del conteo de las materias con el conteo de todos los distintos alumnos) y por último simplemente el conteo del total de alumnos diferentes.

```
-- Desarrollar consultas avanzadas
-- a) Listar la cantidad de alumnos por localidad junto con el número de
-- materias promedio en las que están inscriptos, lo cual permitiría
-- analizar la carga académica en diferentes áreas geográficas.
SELECT
l.nombre AS Localidad,
ROUND(COUNT(em.id_materia) / COUNT(DISTINCT e.id_integrante), 2) AS
Promedio_materias,
COUNT(DISTINCT e.id_integrante) AS Total_alumnos
FROM INTEGRANTE AS i
JOIN ESTUDIANTE AS e ON i.id_integrante = e.id_integrante
JOIN ESTUDIANTE_MATERIA AS em ON e.id_integrante = em.id_estudiante
JOIN LOCALIDAD AS l ON i.id_localidad = l.id_localidad
GROUP BY l.nombre
ORDER BY Total_alumnos DESC;
```

Se muestra en el reporte todas las localidades con el total de alumnos cursando alguna materia y su promedio de materias cursadas.

Una pregunta que nos surgió luego de este reporte fue ¿No existen 15 localidades?. Exacto, pero la localidad faltante proviene de un Docente, por ende, es correcto que no se pueda visualizar.

#	Localidad	Promedio_materias	Total_alumnos
1	CABA	2.22	9
2	San Martin	2.14	7
3	Villa Ballester	2.50	6
4	Santos Lugares	2.33	3
5	Escobar	3.00	2
6	Loma Hermosa	2.50	2
7	Villa Bosch	2.00	2
8	Chilavert	3.00	1
9	Ciudad Jardin	3.00	1
10	El Palomar	2.00	1
11	Jose Leon Suarez	2.00	1
12	LaFerrere	1.00	1
13	Pacheco	1.00	1
14	San Andres	1.00	1

El último punto de consultas avanzadas se abordó desde la tabla HOBBY_INTEGRANTE, uniendo las tablas HOBBY_INTEGRANTE (que matcheen los alumnos sus ids de hobby pero no de integrante, esto quiere decir que no sea la misma persona), INTEGRANTE, HOBBY y ESTUDIANTE.

Por otro lado, se filtraron los resultados por los mentores que su experiencia sea mayor al alumno mentoreado en relacional o no relacional (el operador ">" funciona correctamente ya que el tipo boolean es reconocido como un tinybit en el motor trabajado (0 y 1)).

```
-- 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.
SELECT
  mentor_integrante.apellido AS Mentor_Apellido,
  mentee_integrante.apellido AS Mentee_Apellido,
  GROUP_CONCAT(hobby.nombre) AS Hobbies_Compartidos,
  estudiante_mentor.db_relacional AS Mentor_DB_Relacional,
  estudiante_mentor.db_no_relacional AS Mentor_DB_No_Relacional
FROM HOBBY_INTEGRANTE hi_mentor
INNER JOIN HOBBY_INTEGRANTE hi_mentee ON (hi_mentor.id_hobby =
hi_mentee.id_hobby)
AND (hi_mentor.id_integrante != hi_mentee.id_integrante)
INNER JOIN INTEGRANTE mentor_integrante ON (hi_mentor.id_integrante =
mentor_integrante.id_integrante)
INNER JOIN INTEGRANTE mentee_integrante ON (hi_mentee.id_integrante =
mentee_integrante.id_integrante)
INNER JOIN HOBBY hobby ON (hi_mentor.id_hobby = hobby.id_hobby)
INNER JOIN ESTUDIANTE estudiante_mentor ON
(mentor_integrante.id_integrante = estudiante_mentor.id_integrante)
INNER JOIN ESTUDIANTE estudiante_mentee ON
(mentee_integrante.id_integrante = estudiante_mentee.id_integrante)
WHERE estudiante_mentor.db_relacional > estudiante_mentee.db_relacional
OR estudiante_mentor.db_no_relacional > estudiante_mentee.db_no_relacional
GROUP BY mentor_integrante.id_integrante, mentee_integrante.id_integrante
ORDER BY Mentor_Apellido, Mentee_Apellido;
```

Se puede ver en la siguiente imagen los primeros 10 registro del reporte ordenado alfabéticamente por apellido de mentor.

#	Mentor_Apellido	Mentee_Apellido	Hobbies_Compartidos	Mentor_DB_Relacional	Mentor_DB_No_Relacional
1	Barneche	Bortolussi	Deportes	1	1
2	Barneche	Caballero	Musica	1	1
3	Barneche	Cuellar	Deportes,Musica	1	1
4	Barneche	Decuzzi	Deportes	1	1
5	Barneche	Dragonetti	Musica	1	1
6	Barneche	Ferretti	Deportes,Musica	1	1
7	Barneche	Fucceneco	Musica	1	1
8	Barneche	Geraghty	Deportes	1	1
9	Barneche	Hoj	Deportes	1	1
10	Barneche	Kovinchich	Musica	1	1

Evaluar/considerar el rendimiento de las consultas con grandes volúmenes de datos y proponer ajustes como por ejemplo: uso de índices de ser necesario, incluir justificación en cada caso.

Buenas Prácticas Implementadas

Esta sección tiene como fin principal abordar posibles mejoras y optimizaciones para el proyecto en curso. Entendemos que actualmente es una base de dato muy pequeña pero esta podría crecer a gran escala y verse probablemente afectada con problemas de recursos y optimización.

Como primer propuesta, el equipo ve con muy buenos ojos la posibilidad de agregar índices en columnas claves que se utilizan frecuentemente, como por ej `id_integrante`, `id_hobby` y `id_materia`.

Esta propuesta se puede afrontar de la siguiente manera.

```
CREATE INDEX idx_hobby_integrante ON HOBBY_INTEGRANTE(id_integrante, id_hobby);
CREATE INDEX idx_estudiante_materia ON ESTUDIANTE_MATERIA(id_estudiante, id_materia);
CREATE INDEX idx_integrante_localidad ON INTEGRANTE(id_localidad);
```

Otro punto que se propone es la creación de vistas para consultas muy complejas y/o de mucho recursos. Por ejemplo las `ROW_NUMBER` son muy costosas a lo que recursos se refiere, por ende, el equipo no ve con malos ojos usar vistas para almacenar resultados intermedios.

Se deja un ejemplo de como se podría abordar dicha optimización.

```
SQL ▾ Copy Caption ...

CREATE VIEW vw_estudiante_materia_grupo AS
SELECT
    m.nombre AS nombre_materia,
    g.nombre AS nombre_grupo,
    COUNT(i.id_integrante) AS total_alumno,
    CONCAT(COUNT(i.id_integrante) * 20, '%') AS porcentaje_grupo,
    ROW_NUMBER() OVER (PARTITION BY g.nombre ORDER BY COUNT(i.id_integrante) DESC,
FROM ESTUDIANTE_GRUPO_ROL_egr
JOIN ESTUDIANTE i ON egr.id_estudiante = i.id_integrante
JOIN GRUPO g ON egr.id_grupo = g.id_grupo
JOIN ESTUDIANTE_MATERIA em ON i.id_integrante = em.id_estudiante
JOIN MATERIA m ON em.id_materia = m.id_materia
GROUP BY m.nombre, g.nombre;
```

Una vez creada la vista, se puede obtener de manera más óptima el valor guardado en la misma de la siguiente manera.

```
SQL
SELECT nombre_materia, nombre_grupo, total_alumno, porcentaje_grupo
FROM vw_estudiante_materia_grupo
WHERE rn = 1;
```

Como último punto, si bien entendemos que se puede perder precisión en cuanto al resultado, se podría reemplazar los valores NULOS de las columnas con índices por algún valor por defecto para mejorar su optimización ya que los valores NULOS afectan negativamente la optimización de índices y filtros.

Parte 5: Procedimientos, Triggers y Automatización

1. Análisis Teórico de Procedimientos Almacenados:

- Investigación: Investigar el concepto de procedimientos almacenados en bases de datos, su propósito y beneficios en la automatización de tareas recurrentes. ○ Propuesta Teórica:

Análisis teórico Store Procedure

Un procedimiento almacenado o Store Procedure (SP) es un conjunto de instrucciones en SQL a las que se les asigna un nombre que es almacenado en el servidor, permitiendo encapsular tareas repetitivas.

Un procedimiento almacenado puede hacer referencia a tablas, vistas e incluso puede generar tablas.

Ventajas

- Centralizar la lógica de negocio en un solo lugar, ya que puede ser llamado por distintas aplicaciones evitando duplicidad o diferencias de ejecución no deseadas.
- Reducción del tráfico de red, disminuye el número de solicitudes al servidor, ya que se ejecuta de una sola llamada.
- Protección de la lógica, los usuarios podrán ejecutar la llamada sin necesidad de acceder directamente a las tablas de base de datos, limitando accesos.

Desventajas

- Sintaxis compleja puede ser difícil de depurar.
 - Curva de aprendizaje, requiere un conocimiento avanzado de SQL escribir y optimizar
- Proponer un procedimiento teórico para registrar automáticamente la inscripción de un nuevo alumno en una materia.

Propuesta Teórica:

En base al requerimiento para registrar la inscripción de los alumnos en la base de datos un procedimiento almacenado es un método útil para cargar en las tablas

Modelo de solución:

Debemos utilizar la tabla materia_estudiante donde vamos a insertar las inscripciones

Creamos un Store Procedure 'sp_inscripcion_materia' que llevará las instrucciones necesarias para insertar en la tabla ESTUDIANTE_MATERIA al estudiante y la materia.

```
CREATE PROCEDURE sp_inscripcion_materia (  
    IN p_estudiante INT,  
    IN p_materia INT  
)  
BEGIN  
    DECLARE materia_existente INT;  
    START TRANSACTION;  
    SELECT COUNT(*) INTO materia_existente  
    FROM materia WHERE id_materia = p_materia;  
    IF materia_existente = 1 THEN  
        INSERT INTO inscripcion_materia (  
            ID_ESTUDIANTE, ID_MATERIA  
        )  
        VALUES ( p_estudiante, p_materia);  
        COMMIT;  
    ELSE  
        ROLLBACK;  
    END IF;  
END  
CALL sp_inscripcion_materia(1, 14);
```

- Explicar cómo este procedimiento podría facilitar la gestión de datos y reducir la carga manual.

De esta manera al ejecutar el procedimiento automatizará operación, lograría evitar problemas de inconsistencia de datos ya que la lógica del negocio estaría en este SP como por ejemplo validaciones de que exista la materia.

1. Análisis Teórico de Triggers:

- Investigación: Estudiar el concepto de triggers en bases de datos, cuándo se usan y por qué son útiles para mantener la integridad de los datos.

Análisis Teórico de Triggers:

Un "trigger" (disparador o desencadenador) es un bloque de algoritmo que se ejecuta cuando se intenta realizar una modificación sobre una tabla, al producirse ese evento desencadenador el trigger se dispara ejecutando una acción como por ejemplo insert,update,delete de forma automática

Son útiles a la hora de mantener la integridad de datos porque previenen inconsistencias, actualiza datos relacionados, logra trazabilidad de los registros ya que puede generar tabla de auditoría.

Ventajas

- Automatización de Procesos
- Mantenimiento de la Integridad de los Datos
- Registro Auditoría

Desventajas

- Mantenimiento, por su lógica pueden ser difícil de entender
- El rendimiento, al ejecutarse sin intervención puede ralentizar las operaciones del usuario.

- Proponer un trigger que registre un log automáticamente cuando se modifique la información personal de un alumno, explicando su utilidad en auditorías de datos.

Modelo de solución:

Debemos crear la tabla ALUMNO_LOG donde vamos a dejar registro de los cambios que realice el alumno.

```
CREATE TABLE alumno_log (  
  id INT IDENTITY(1,1) PRIMARY KEY,  
  fecha DATETIME DEFAULT GETDATE(),  
  columna_modificada VARCHAR(255),  
  valor_anterior VARCHAR(255),  
  valor_nuevo VARCHAR(255),  
  usuario VARCHAR(50)  
);
```

Aquí vamos a guardar la auditoría de los movimientos del alumno.
Generamos el trigger:

```
CREATE TRIGGER trg_alumno_modificacion  
ON INTEGRANTE AFTER UPDATE AS BEGIN  
  SET NOCOUNT ON;  
  -- Insertar registro en el log para cada columna modificada  
  DECLARE @columna VARCHAR(255)  
  DECLARE @valor_anterior VARCHAR(255)  
  DECLARE @valor_nuevo VARCHAR(255)  
  DECLARE cursor_modificaciones CURSOR FOR  
  SELECT COL_NAME(PARTITION_SCHEMA, PARTITION_COLUMN_ID) AS  
columna,  
    CAST(old.[INTEGRANTE] AS VARCHAR(255)) AS valor_anterior,  
    CAST(new.[INTEGRANTE] AS VARCHAR(255)) AS valor_nuevo  
  FROM inserted  
  CROSS APPLY (SELECT * FROM deleted) d  
  WHERE d.INTEGRANTE <> new.INTEGRANTE  
  OPEN cursor_modificaciones  
  FETCH NEXT FROM cursor_modificaciones INTO @columna,  
@valor_anterior, @valor_nuevo  
  WHILE @@FETCH_STATUS = 0  
  BEGIN  
    INSERT INTO alumno_log (columna_modificada, valor_anterior,  
valor_nuevo, usuario)  
    VALUES (@columna, @valor_anterior, @valor_nuevo, SYSTEM_USER)  
    FETCH NEXT FROM cursor_modificaciones INTO @columna,  
@valor_anterior, @valor_nuevo  
  END  
  CLOSE cursor_modificaciones  
  DEALLOCATE cursor_modificaciones  
END
```


- Documentación: Describir el trigger en detalle, incluyendo cuándo se activaría, qué acciones ejecutaría y por qué mejoraría la gestión de la base de datos.

De esta manera cada vez que se realice una modificación se ejecutará esta lógica, insertando la columna, el valor anterior y el valor actual de lo modificado, lo que proporciona un historial claro de los cambios realizados.

1. Propuesta de Estrategias de Automatización:

- Investigación de Automatización en Bases de Datos: Investigar técnicas básicas de automatización en bases de datos, como los índices de optimización, el uso de vistas, y la programación de tareas (backups, actualizaciones, etc.).

El uso de vistas proporciona los siguientes beneficios:

- Seguridad: Ya que las vistas proporcionan acceso “virtual” para los usuarios a tablas que no son directamente accesibles. Lo que permite al DBA darle a los usuarios la información que únicamente ellos requieren o necesitan sin comprometer info de la misma tabla.
- Simplicidad: las vistas nos permiten ocultar y hacer reuso de queries complejas.
- Claridad: por medio de vistas también puedes dar alias a columnas para hacerlas más entendibles y claras.
- Optimización: ya que podrías armar vistas para alguna query compleja y particular. Y luego aplicar una query sobre esta vista creada, lo cual simplifica y optimiza el trabajo.

Índices de Optimización

Los índices son estructuras de datos asociadas a tablas que mejoran la velocidad de las operaciones de búsqueda y consulta.

Teniendo un índice en la tabla, hará más rápido el proceso de recuperación de información, proveyendo al cliente la información requerida en un corto tiempo. La programación de tareas consiste en la ejecución automática de comandos o scripts en intervalos de tiempo predefinidos.

Aplicaciones Comunes

- Backups: Las copias de seguridad programadas aseguran la integridad de datos, permitiendo su recuperación en caso de fallos del sistema o ciberataques.
- Actualizaciones: Automatizar la instalación de parches y actualizaciones mejora la seguridad y funcionalidad del sistema.

- Monitoreo y Mantenimiento: Scripts programados pueden supervisar recursos del sistema, generar reportes o ejecutar limpiezas periódicas de archivos temporales.
- Propuesta de Estrategia Teórica: Proponer al menos una estrategia de automatización para la base de datos de alumnos, por ejemplo: una rutina de backup o cómo implementar índices y vistas para mejorar el rendimiento.

Tareas de backup:

Aquí vamos a generar el script para realizar automatización

```
#!/bin/bash

# Database credentials
DB_USER="your_username"
DB_PASSWORD="your_password"
DB_NAME="your_database_name"

# Backup directory
BACKUP_DIR="/path/to/backup/directory"

# Timestamp (to create unique backup filenames)
TIMESTAMP=$(date +"%Y%m%d%H%M%S")

# Create backup directory if it doesn't exist
mkdir -p $BACKUP_DIR

# Backup the MySQL database
mysqldump -u$DB_USER -p$DB_PASSWORD $DB_NAME >
$BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql

# Compress the backup
gzip $BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql

# Optionally, you can remove backups older than a certain period
# find $BACKUP_DIR -type f -name "*.gz" -mtime +7 -exec rm {} \;

echo "Backup completed: $BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql.gz"
```

Ejecutar el script completando las credenciales correspondientes, el script comprueba si el directorio de copia de seguridad existe, si no existe, lo crea. El mysqldump comando exporta la base de datos MySQL especificada a un archivo SQL.

Luego, el archivo de respaldo se comprime gzip para un almacenamiento eficiente.

Automatización y programación

Para automatizar este script, puedes aprovechar herramientas como las *cron* que se usan en sistemas tipo Unix. Por ejemplo, para programar una copia de seguridad diaria, agrega la siguiente línea a tu crontab:

```
0 0 *** /ruta/to/your/script.sh
```

Esta línea programa el script para que se ejecute todos los días a medianoche. Con este script bash, ha establecido una solución sólida para automatizar las copias de seguridad de bases de datos MySQL. Realizar copias de seguridad periódicas de su base de datos es un aspecto crucial de la gestión de datos, lo que garantiza que pueda recuperarse rápidamente de eventos imprevistos.

Parte 6: Buenas Prácticas, Documentación y Presentación del TP

Objetivo: Documentar el proceso completo y presentar el proyecto en un formato profesional y orientado al cliente.

1. Buenas Prácticas y Documentación Completa:

- Documentar el proceso completo de desarrollo de este trabajo práctico, incluyendo:
 - Organización inicial de datos, diseño del modelo, elección del motor y justificación.
 - Ejecución y explicación de consultas y procedimientos implementados.
 - Las queries utilizadas y capturas de la consola con los resultados obtenidos.
 - Explicar las buenas prácticas aplicadas (ej. uso de índices, normalización y backups)

2. Presentación final del documento:

- Preparar una presentación clara, concisa y profesional que incluya:
 - Descripción del modelo y el esquema de la base de datos.
 - Justificación del motor seleccionado y sus beneficios de la solución.
 - Ejemplos de reportes o análisis que pueden extraerse de la base de datos.



6. Para abordar este punto se crea tanto un documento técnico como una presentación para el cliente.

Documentación Técnica: [Documentación Técnica](#)

Presentación Cliente: [Presentación Cliente](#)