Universidad Autónoma de Querétaro

Facultad de Contaduría y Administración

Proyecto Final

"SQL HandBook"

Como parte de la evaluación de la materia de temas selectos de estadística dentro de la carrera en:

Licenciatura en Actuaría

Presenta:

Ríos González Olaf Farid

Santiago de Querétaro, Qro, Noviembre 2024

Índice

Introducción

Bases de Datos

Una base de datos es una estructura organizada destinada a almacenar y gestionar información de manera eficiente. Se compone de dos elementos clave:

- Datos: La información descriptiva almacenada (conocida como *raw data*).
- Metadatos: Información que describe la estructura de la base de datos, como la definición de tablas (por ejemplo, la tabla de clientes).

Las bases de datos son gestionadas por un Sistema de Gestión de Bases de Datos (*DBMS*, por sus siglas en inglés), que facilita su administración.

Base de Datos vs Conjunto de Datos

Un conjunto de datos (*dataset*) es una colección de datos relacionados, generalmente presentados en forma tabular. A continuación, se comparan las principales diferencias entre una base de datos y un conjunto de datos:

Característica	Conjunto de Datos	Base de Datos
Propósito	Análisis e informes	Almacenamiento, recuperación y manipulación
Estructura	Tabular	Tablas, índices, vistas y procedimientos
Formatos de Almacenamiento	CSV, XLSX, JSON	MySQL, PostgreSQL, Oracle

---

Base de Datos vs Modelo de Base de Datos

Un modelo de base de datos es un esquema que define cómo se organizan, almacenan y manipulan los datos dentro de una base de datos. Mientras que una base de datos es la

colección real de datos gestionada por un DBMS, el modelo de base de datos es el diseño abstracto que describe cómo se estructuran y se relacionan esos datos.

Tipos de Modelos de Base de Datos

1. **Modelo Jerárquico (1950):**
    - Los datos están organizados en una estructura de árbol.
    - Las tablas tienen una relación de tipo padre-hijo.
    - Cada tabla hija tiene un único padre, mientras que cada padre puede tener múltiples hijos.
2. **Modelo de Red (1960):**
    - Los datos se organizan en una estructura de grafo, con nodos y enlaces.
    - Permite que las tablas hijas tengan múltiples padres.
3. **Modelo Relacional (1970):**
    - Los datos se organizan en tablas con filas (registros) y columnas (campos).
    - Supera las limitaciones de la estructura jerárquica.
    - Las tablas pueden accederse de manera directa sin necesidad de recorrer todos los objetos padres.
4. **Modelo Orientado a Objetos (1990):**
    - Los datos se representan como objetos, similar a la programación orientada a objetos.
    - Utiliza una estructura tridimensional.
    - Eficiente para la localización de elementos individuales, pero con un rendimiento menor cuando se necesitan varios elementos a la vez.

SQL

El SQL (Structured Query Language o Lenguaje de Consulta Estructurado) es un lenguaje diseñado para que tanto usuarios técnicos como no técnicos puedan consultar, manipular y transformar datos en bases de datos relacionales. Gracias a su simplicidad, SQL se ha convertido en el estándar para el almacenamiento seguro y escalable de datos en sitios web y aplicaciones móviles.

Conceptos Clave

- **Registro:** Fila que contiene los datos de una observación o entidad.
- **Campo:** Columna que almacena un dato específico para todos los registros en la tabla.
- **Tabla:** Estructura bidimensional formada por campos (columnas) y registros (filas). Los campos se definen al crear la tabla y no existe un límite para la cantidad de registros.
- **Base de Datos Relacional:** Conjunto de tablas relacionadas que almacenan datos de manera estructurada, con relaciones definidas entre ellas.
- **Identificadores Únicos:** Valores únicos, generalmente enteros, usados para identificar registros dentro de una tabla (también conocidas como Claves Primarias).
- **Clave Foránea:** Campo (o conjunto de campos) en una tabla que identifica de manera única una fila en otra tabla, estableciendo una relación entre ambas.
- **Consulta:** Petición de datos o información de una base de datos, escrita en SQL. Las consultas pueden recuperar, insertar, actualizar o eliminar datos.

Normas de Nombres en SQL

Nombres de Tablas

- Usar minúsculas.
- Evitar espacios; en su lugar, usar guion bajo (_).

- Los nombres deben hacer referencia a un grupo colectivo o ser plurales.
- Deben diferir de los nombres de los campos y de otras tablas.

Nombres de Campos

- Siempre en singular.
- Evitar el uso de "id" como identificador principal de la tabla.
- No añadir una columna con el mismo nombre que la tabla.
- Usar minúsculas, salvo cuando tenga sentido lo contrario (por ejemplo, para nombres propios).

Tipos de Datos

Los tipos de datos en SQL definen el tipo de información que puede almacenarse en un campo. Es importante que cada campo tenga un único tipo de dato.

Numéricos

- **INT:** Almacena números enteros (TINYINT, SMALLINT, MEDIUMINT, BIGINT son variantes con diferentes rangos).
- **DECIMAL(p, s):** Almacena números de punto fijo con una precisión de p dígitos y s dígitos después del punto decimal.
- **FLOAT y DOUBLE:** Almacenan números de punto flotante aproximados, con FLOAT ocupando menos espacio, pero ofreciendo menor precisión que DOUBLE.

Caracteres

- **CHAR(n):** Almacena cadenas de longitud fija. Si los datos son más cortos que n caracteres, se rellenan con espacios.
- **VARCHAR(n):** Almacena cadenas de longitud variable. Utiliza solo el espacio necesario para los datos, más un pequeño sobrecosto para la longitud.

- **TEXT:** Almacena grandes cantidades de texto. Las variantes incluyen TINYTEXT, TEXT, MEDIUMTEXT y LONGTEXT.

Fecha y Hora

- **DATE:** Almacena fechas en formato YYYY-MM-DD.
- **TIME:** Almacena horas en formato HH:MM:SS.
- **DATETIME:** Almacena fecha y hora en el formato YYYY-MM-DD HH:MM:SS.
- **TIMESTAMP:** Similar a DATETIME, pero generalmente se usa para rastrear cambios en los registros, almacenando la fecha y hora en formato UTC.
- **YEAR:** Almacena un valor de año, generalmente en formato de dos o cuatro dígitos.

Comandos Básicos de SQL

Crear una tabla

Este comando crea una nueva tabla en la base de datos, especificando el nombre de la tabla y los campos con sus tipos de datos correspondientes.

CREATE TABLE table_name (

field_1 datatype,

field_2 datatype,

...

field_n datatype);

Nota: Si el nombre de la tabla incluye espacios (lo cual no es recomendable), se debe utilizar comillas dobles ("") alrededor del nombre.

Insertar datos

El comando INSERT INTO se utiliza para agregar nuevos registros a una tabla.

INSERT INTO table_name (field_1, field_2, ..., field_n)

VALUES

(value_1, value_2, ..., value_n),

(value_1, value_2, ..., value_n);

Se pueden agregar tantos registros como sea necesario, separándolos por comas.

Seleccionar datos

Para recuperar datos de una base de datos, se utiliza el comando SELECT.

SELECT field_1, field_2, ..., field_n

FROM table_name;

Si deseas recuperar todos los campos de una tabla, puedes utilizar el asterisco (*):

SELECT *

FROM table_name;

Bases de datos relacionales

Antes de aprender la sintaxis SQL, es importante comprender cómo funcionan las bases de datos relacionales. Estas representan una colección de tablas bidimensionales, cada una similar a una hoja de cálculo de Excel, con un número fijo de columnas (atributos) y un número variable de filas (registros).

Consultas SELECT

Las consultas **SELECT** son el principal medio para recuperar datos de una base de datos SQL. Una consulta especifica qué datos se buscan, dónde encontrarlos y cómo transformarlos antes de devolverlos. Una tabla en SQL puede verse como un tipo de entidad (por ejemplo, "perros"), y cada fila como una instancia de esa entidad (por ejemplo, un pug, un beagle, etc.).

La consulta más básica selecciona columnas específicas de una tabla:

SELECT column, another_column, ...

FROM mytable;

Si deseas recuperar todas las columnas, puedes utilizar el siguiente comando:

SELECT *

FROM mytable;

Consultas con restricciones

Para evitar que una consulta devuelva demasiados resultados (por ejemplo, en una tabla con millones de filas), puedes agregar condiciones con la cláusula **WHERE**. Esta cláusula filtra los resultados según los valores de las columnas.

SELECT column, another_column, ...

FROM mytable

WHERE condition

AND/OR another_condition;

Algunos operadores útiles para condiciones numéricas incluyen:

=, !=, <, <=, >, >=:	Operadores estándar.
BETWEEN ... AND ...	Rango de valores inclusivos.
IN (...):	El valor está en una lista
NOT IN (...):	El valor no está en una lista

Operadores para datos de texto

Cuando trabajas con columnas de texto, SQL ofrece operadores como:

=	Comparación exacta (sensible a mayúsculas y minúsculas).
LIKE	Comparación sin distinción de mayúsculas y minúsculas.
NOT LIKE	Desigualdad de cadenas sin distinción de mayúsculas y minúsculas.
%	Comodín que representa cero o más caracteres.
_	Comodín para un solo carácter.

Filtrado y ordenación de resultados

Si deseas evitar resultados duplicados, utiliza la palabra clave **DISTINCT**:

SELECT DISTINCT column, another_column, ...

FROM mytable;

Para ordenar los resultados de una consulta, usa **ORDER BY**, seguido del nombre de la columna y la dirección de orden (**ASC** para ascendente o **DESC** para descendente):

SELECT column, another_column, ...

FROM mytable

WHERE condition

ORDER BY column ASC/DESC;

Además, si solo necesitas un subconjunto de los resultados, puedes usar **LIMIT** y **OFFSET**:

SELECT column, another_column, ...

FROM mytable

WHERE condition

ORDER BY column ASC/DESC

LIMIT num_limit OFFSET num_offset;

Consultas simples

Una consulta básica selecciona columnas, las filtra con una cláusula WHERE, las ordena con ORDER BY, y limita el número de resultados con LIMIT y OFFSET.

SELECT column, another_column, ...

FROM mytable

WHERE condition

ORDER BY column ASC/DESC

LIMIT num_limit OFFSET num_offset;

Consultas multitabla con JOIN

En bases de datos normalizadas, los datos se distribuyen en varias tablas relacionadas. Para obtener información combinada de estas tablas, se utiliza la cláusula **JOIN**. El tipo más común es el **INNER JOIN**, que devuelve solo las filas que tienen una coincidencia en ambas tablas.

SELECT column, another_table_column, ...

FROM mytable

INNER JOIN another_table

ON mytable.id = another_table.id

WHERE condition

ORDER BY column ASC/DESC

LIMIT num_limit OFFSET num_offset;

Uniones externas

Cuando necesitas incluir datos de ambas tablas, incluso si no hay una coincidencia, se utilizan **LEFT JOIN**, **RIGHT JOIN**, o **FULL JOIN**:

SELECT column, another_column, ...

FROM mytable

LEFT/RIGHT/FULL JOIN another_table

ON mytable.id = another_table.matching_id

WHERE condition

ORDER BY column ASC/DESC

LIMIT num_limit OFFSET num_offset;

LEFT JOIN	Incluye filas de la primera tabla incluso sin coincidencias en la segunda tabla.
RIGHT JOIN	Incluye filas de la segunda tabla, aunque no haya coincidencias en la primera.
FULL JOIN	Mantiene todas las filas de ambas tablas, con valores NULL donde no hay coincidencia.

Valores NULL

En SQL, un valor NULL indica que un campo no tiene valor o está vacío. Las consultas deben tener en cuenta estos valores, especialmente cuando se utilizan en condiciones. Para filtrar filas con valores NULL, se usan las condiciones IS NULL o IS NOT NULL:

SELECT column, another_column, ...

FROM mytable

WHERE column IS NULL;

Consultas con expresiones

En SQL, además de consultar datos directos de las columnas, es posible emplear expresiones para incorporar lógica más compleja dentro de una consulta. Estas expresiones pueden combinar funciones matemáticas, de cadenas y operaciones aritméticas para transformar los valores en el momento de ejecutar la consulta, tal como se muestra en el siguiente ejemplo:

SELECT particle_speed / 2.0 AS half_particle_speed

FROM physics_data

WHERE ABS(particle_position) * 10.0 > 500;

Cada base de datos ofrece su propio conjunto de funciones matemáticas, de cadenas y de fecha, que pueden consultarse en la documentación oficial de la base de datos correspondiente.

El uso de expresiones puede simplificar la consulta y evitar pasos adicionales de procesamiento de los datos, pero puede hacer que la consulta sea más difícil de leer. Por ello, se recomienda asignar un alias descriptivo a las expresiones utilizando la palabra clave AS.

Consulta con alias para expresiones:

SELECT col_expression AS expr_description, ...

FROM mytable;

Además de las expresiones, se pueden asignar alias a columnas y tablas para facilitar la referencia a los datos en la salida y simplificar consultas complejas.

Ejemplo de consulta con alias en columnas y tablas:

SELECT column AS better_column_name, ...

FROM a_long_widgets_table_name AS mywidgets

INNER JOIN widget_sales

ON mywidgets.id = widget_sales.widget_id;

Consultas con agregados

Además de las expresiones básicas, SQL también permite utilizar funciones agregadas, que resumen información sobre un conjunto de filas. Estas funciones son útiles para responder preguntas como: *"¿Cuántas películas ha producido A24?"* o *"¿Cuál es la película más taquillera de A24 cada año?"*

Consulta de selección con funciones agregadas sobre todas las filas:

SELECT group_by_column, AGG_FUNC(column_expression) AS aggregate_result_alias, ...

FROM mytable

WHERE condition

GROUP BY column

HAVING group_condition;

La cláusula **HAVING** permite filtrar las filas después de la agrupación, lo que es útil cuando se necesita aplicar restricciones sobre los grupos resultantes.

Orden de ejecución de una consulta

Entender el orden de ejecución de las partes de una consulta es fundamental para saber qué datos estarán disponibles en cada etapa. El orden de ejecución es el siguiente:

1. FROM y JOINs

- El primer paso es identificar las tablas involucradas en la consulta.
- Aquí se procesan las uniones entre tablas.

2. WHERE

- Filtra registros según condiciones específicas.
- Sólo los registros que cumplen las condiciones pasan al siguiente paso.

3. GROUP BY

- Agrupa registros que tienen los mismos valores en campos específicos en registros de resumen.
- A estos grupos se les aplican funciones de agregación .

4. HAVING

- Filtra grupos según condiciones.
- Similar a WHERE, pero opera en grupos creados por GROUP BY.

5. SELECT

- Selecciona los campos que se incluirán en el resultado final.
- Puede incluir funciones y expresiones agregadas.

6. DISTINCT

- Eliminar los registros duplicados.
- Trabaja en los campos marcados.

7. ORDER BY

- Ordena el conjunto de resultados según los campos especificados.
- Puede ordenar en orden ascendente (ASC) o descendente (DESC).

8. LIMIT/OFFSET

- Limita el número de registros devueltos por la consulta.
- OFFSET especifica el número de registros que se deben omitir antes de comenzar a devolver registros.

Es importante entender cómo estas partes interactúan para escribir consultas eficientes y correctas.

Inserción de filas

En esta lección, comenzamos a explorar cómo agregar datos a una base de datos mediante la instrucción **INSERT**. Al insertar nuevos datos, debes especificar las columnas de la tabla y los valores correspondientes.

Ejemplo de inserción de valores en todas las columnas:

INSERT INTO mytable

VALUES (value_or_expr, another_value_or_expr, ...),

(value_or_expr_2, another_value_or_expr_2, ...),

...;

Es posible insertar varias filas al mismo tiempo, enumerándolas secuencialmente. Si solo tienes valores parciales, puedes especificar explícitamente las columnas a insertar.

Ejemplo de inserción con columnas específicas:

INSERT INTO mytable

(column, another_column, ...)

VALUES (value_or_expr, another_value_or_expr, ...),

(value_or_expr_2, another_value_or_expr_2, ...),

...;

Además, es posible usar expresiones para calcular valores durante la inserción, lo que garantiza que los datos se ajusten a un formato determinado.

Actualización de filas

Cuando se necesita modificar datos existentes, se utiliza la instrucción **UPDATE**, especificando qué tabla, columnas y condiciones deben cumplirse para realizar la actualización.

Ejemplo de declaración de actualización con valores:

UPDATE mytable

SET column = value_or_expr,

other_column = another_value_or_expr,

...

WHERE condition;

Es esencial verificar las restricciones en la cláusula WHERE antes de realizar la actualización para evitar errores.

Eliminación de filas

La instrucción DELETE se utiliza para eliminar datos de una tabla. Es muy importante aplicar correctamente la cláusula WHERE para evitar la eliminación accidental de todas las filas.

Ejemplo de declaración de eliminación:

DELETE FROM mytable

WHERE condition;

Al igual que con la instrucción UPDATE, se recomienda probar la restricción WHERE con una consulta SELECT antes de ejecutar el DELETE.

Creación de tablas

Cuando es necesario agregar nuevas entidades a la base de datos, se puede utilizar la instrucción **CREATE TABLE** para definir la estructura de una nueva tabla.

Ejemplo de creación de tabla con restricciones:

CREATE TABLE IF NOT EXISTS mytable (

column DataType TableConstraint DEFAULT default_value,

another_column DataType TableConstraint DEFAULT default_value,

...

);

La instrucción CREATE TABLE permite definir las columnas, los tipos de datos, las restricciones y los valores predeterminados para una tabla. También es posible omitir la creación de una tabla si ya existe utilizando la cláusula **IF NOT EXISTS**.

Modificación de tablas

Con el tiempo, las tablas pueden necesitar ser modificadas. Para ello, se utiliza la instrucción **ALTER TABLE**, que permite agregar, eliminar o modificar columnas y restricciones en una tabla existente.

Ejemplo de adición de columna:

ALTER TABLE mytable

ADD column DataType OptionalTableConstraint DEFAULT default_value;

Eliminación de tablas

Si es necesario eliminar una tabla completa, incluyendo sus datos y estructura, se puede usar la instrucción **DROP TABLE**.

Ejemplo de eliminación de tabla:

DROP TABLE IF EXISTS mytable;

La cláusula **IF EXISTS** asegura que no se genere un error si la tabla no existe.

Normalización

Definición 1

La normalización de bases de datos es el proceso de organizar los atributos de la base de datos para reducir o eliminar la redundancia de datos (tener los mismos datos, pero en diferentes lugares). La redundancia de datos aumenta innecesariamente el tamaño de la base de datos, ya que los mismos datos se repiten en muchos lugares. Los problemas de inconsistencia también surgen durante las operaciones de *inserción* , *eliminación* y *actualización* .

Definición 2

La normalización de bases de datos es un principio de diseño de bases de datos para organizar datos de forma organizada y consistente.

Le ayuda a evitar la redundancia y a mantener la integridad de la base de datos. También le ayuda a eliminar características no deseadas asociadas con *la inserción* , *la eliminación* y *la actualización* .

Tipos de Claves

Para este tema es importante recordar los diferentes tipos de claves en una base de datos:

- **La clave principal** es un *campo* que identifica de forma única los *registros* de datos en esa tabla.
- **La clave externa** es un campo que se relaciona con la *clave principal* en otra tabla.
- **Una clave compuesta** es como una clave principal, pero en lugar de tener un campo, tiene varios campos.

Formas normales

Las formas normales son los pasos para lograr la normalización. Cada forma depende de la anterior y decimos que una base de datos está en forma normal si todas sus tablas están en esa forma formal.

Primera Forma (1NF)

Eliminar grupos repetidos

Una tabla está en **1NF** si cada atributo en esa relación es un atributo de valor único:

- Hay una **clave principal** para la identificación.
- No hay campos ni registros **duplicados .**
- Cada campo solo tiene un valor para cada registro.

¿Cómo hacerlo?

1. Eliminar grupos repetidos en tablas individuales.
2. Cree una tabla separada para cada conjunto de datos relacionados.
3. Identifique cada conjunto de datos relacionados con una clave principal.

Segunda Forma (2NF)

Eliminar redundancia

Una tabla está en **2NF** si:

- Está en **1NF** .

- Cada atributo que no es de clave principal depende funcionalmente y en su totalidad de la clave principal.

¿Cómo hacerlo?

1. Cree tablas separadas para conjuntos de valores que se apliquen a múltiples registros.
2. Relacione estas tablas con una clave externa.

Tercera Forma (3FN)

Eliminar la dependencia parcial transitiva

Una tabla está en **3NF** si:

- Está en **2NF** .
- No tienen dependencia parcial transitiva.

¿Cómo hacerlo?

1. Eliminar campos que no dependen de la clave.

Más formas de normalización

Existen más formas de normalización:

- **Cuarta Forma de Normalización** o Forma Normal de Boyce-Codd (**BCNF**).
- **Quinta Forma de Normalización** Estas formas evitan las posibles excepciones que pueden ocurrir en las primeras 3 formas.