Clase 7: Material Complementario

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	Día:	Friday, 23 de January de 2026, 10:18
Libro:	Clase 7: Material Complementario		

Tabla de contenidos

1. Clausulas SQL

Temario:

- **Cláusulas Principales**
- **Operadores y Condicionales**
- **Funciones de Agregación**
- **Tipos de JOIN**
- **Cardinalidad**

1.1. Clausulas SQL

Cláusulas Principales

- SELECT: Esta cláusula se utiliza para seleccionar columnas específicas de una tabla y mostrar los resultados de una consulta. Es fundamental para extraer información en SQL.

- FROM: Especifica la tabla de la cual se obtendrán los datos, siendo esencial para indicar la fuente de datos en la consulta.

- WHERE: Permite aplicar condiciones para filtrar los datos de la tabla, restringiendo los resultados a filas que cumplan con ciertos criterios.

- DISTINCT: Devuelve valores únicos de una columna o combinación de columnas, eliminando los duplicados en los resultados.

- ORDER BY: Ordena los resultados en función de una o más columnas, en orden ascendente (ASC) o descendente (DESC). Es útil para organizar los datos de una manera específica en el resultado.

- LIMIT: Limita la cantidad de filas mostradas en los resultados de una consulta, permitiendo controlar la cantidad de datos visualizados.

- OFFSET: Se usa junto a LIMIT para omitir un número específico de filas antes de comenzar a devolver los resultados, útil en la paginación de resultados.

- GROUP BY: Agrupa filas con base en el valor de una columna específica, permitiendo aplicar funciones de agregación como SUM, AVG, COUNT, etc., sobre cada grupo.

- HAVING: Aplica condiciones a los resultados agrupados después de usar GROUP BY, permitiendo filtrar grupos según el resultado de las funciones de agregación. A diferencia de WHERE, esta cláusula actúa sobre grupos de datos, no sobre filas individuales.

Operadores y Condicionales

- LIKE: Este operador se utiliza para buscar patrones dentro de columnas de tipo texto. Combinado con el comodín %, permite encontrar coincidencias basadas en caracteres parciales (e.g., nombres que empiezan con "A" o terminan en "Z").

- BETWEEN: Permite especificar un rango en el cual deben estar los valores de una columna. Se utiliza tanto con valores numéricos como con texto o fechas, delimitando un rango de resultados.

- IN: Permite definir un conjunto de valores específicos en los cuales deben estar los resultados. Este operador es útil para filtrar una columna con múltiples opciones posibles.

- NOT IN: Similar a IN, pero excluye las filas cuyo valor esté dentro del conjunto de valores especificado, devolviendo resultados que no coincidan con ninguna de las opciones listadas.

- IS NULL / IS NOT NULL: Se utiliza para verificar si una columna tiene o no valores nulos, útil para identificar registros con datos faltantes o incompletos.

Funciones de Agregación

- COUNT: Cuenta el número de filas que cumplen con una condición específica o en una columna determinada, incluso considerando duplicados. Es común usarlo en conjunto con GROUP BY para contar elementos dentro de cada grupo.

- SUM: Calcula la suma total de los valores en una columna específica, aplicándose generalmente en columnas de tipo numérico para obtener totales.

- AVG: Calcula el promedio de los valores en una columna numérica, proporcionando el valor promedio de un conjunto de datos o dentro de un grupo.

- MIN: Devuelve el valor mínimo dentro de una columna numérica o de fecha, permitiendo obtener el menor valor registrado en un conjunto de datos.

- MAX: Devuelve el valor máximo en una columna numérica o de fecha, encontrando el mayor valor

registrado en un conjunto de datos.

Tipos de JOIN

Permiten combinar datos, de dos o más tablas.

Según la relación lógica entre ellas, generalmente basada en una clave.

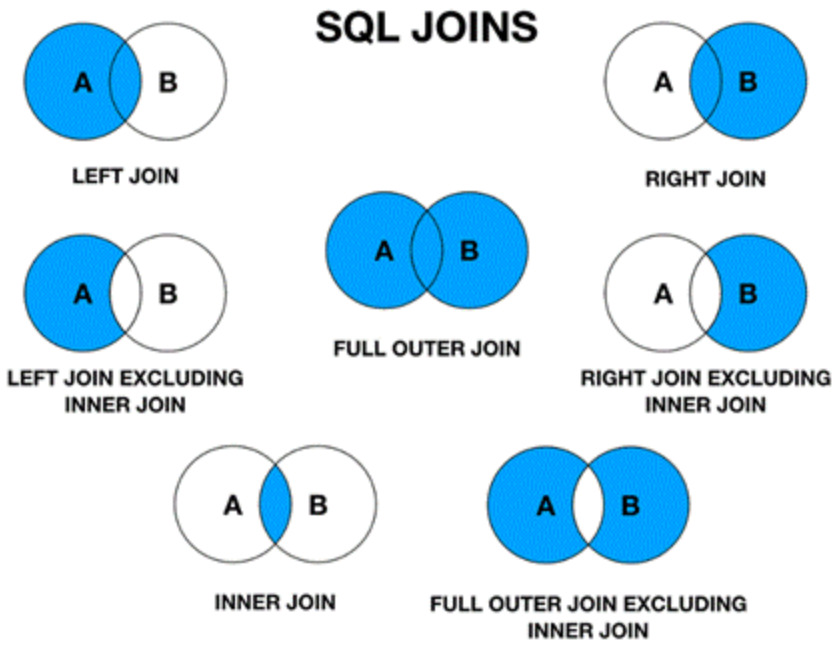
- INNER JOIN: Devuelve solo los registros que tienen coincidencias en ambas tablas. Es útil para encontrar datos relacionados que existen en ambas tablas.

- LEFT (OUTER) JOIN: Devuelve todos los registros de la tabla izquierda y los registros coincidentes de la tabla derecha. Si no hay coincidencias, se muestran valores NULL en las columnas de la tabla derecha.

- RIGHT (OUTER) JOIN: Similar al LEFT JOIN, pero devuelve todos los registros de la tabla derecha y los registros coincidentes de la tabla izquierda. Los valores NULL aparecen en las columnas de la tabla izquierda cuando no hay coincidencias.

- FULL (OUTER) JOIN: Devuelve todos los registros cuando hay una coincidencia en una de las tablas, incluyendo registros que no tienen coincidencias en ambas tablas. Llena los espacios vacíos con NULL donde no haya coincidencias.

- CROSS JOIN: Realiza un producto cartesiano entre las filas de dos tablas, combinando cada fila de la primera tabla con cada fila de la segunda. Este tipo de JOIN es útil para generar todas las posibles combinaciones entre dos conjuntos de datos.



Cardinalidad

Cardinalidad en bases de datos se refiere al tipo de relación que existe entre dos tablas, definiendo cómo los datos en una tabla se relacionan con los datos en otra. Existen tres tipos principales de cardinalidad:

1. Relación Uno a Uno (1:1): Cada registro en una tabla se asocia con un solo registro en otra tabla. Este tipo de relación se usa cuando se quiere dividir una tabla en dos para mejorar la organización de la información sin duplicarla.

> *Ejemplo: En la base de datos de una institución educativa, supongamos que tenemos una tabla alumnos y una tabla detalles_contacto. Si cada alumno tiene un solo contacto de emergencia, se puede establecer una relación 1:1 entre alumnos y detalles_contacto.*

2. Relación Uno a Muchos (1:N): Un registro en una tabla está asociado con múltiples registros en otra tabla. Este tipo de relación es común y se usa cuando un elemento de una tabla se relaciona con muchos elementos de otra.

> *Ejemplo: La relación entre alumnos y asignaciones en la base de datos de la institución educativa. Un alumno puede estar inscrito en varias materias, pero cada inscripción pertenece a un único alumno.*

3. Relación Muchos a Muchos (N:M): Varios registros en una tabla se asocian con varios registros en otra tabla. Este tipo de relación requiere una tabla intermedia (tabla de unión) para administrar la relación, ya que las bases de datos relacionales no permiten la implementación directa de una relación N:M.

> *Ejemplo: La relación entre alumnos y materias. Un alumno puede inscribirse en múltiples materias, y cada materia puede tener múltiples alumnos inscritos. Aquí, la tabla asignaciones actúa como una tabla de unión para representar esta relación N:M.*

En este caso:

- alumnos se relacionan con materias mediante la tabla intermedia asignaciones.

- asignaciones contiene id_alumno y id_materia como claves foráneas para representar la relación muchos a muchos.

Resumen de Ejemplos de Cardinalidad

Cardinalidad	Tablas Relacionadas	Ejemplo en Institución Educativa
1:1	`alumnos` - `detalles_contacto`	Un alumno tiene un contacto de emergencia único.
1	`alumnos` - `asignaciones`	Un alumno puede inscribirse en varias asignaciones.
N	`alumnos` - `materias`	Los alumnos se inscriben en múltiples materias.

Estos tipos de cardinalidad ayudan a organizar las bases de datos de forma eficiente, evitando la duplicación de datos y facilitando la gestión de la información en la institución educativa.

2. Categorías SQL

Temario

- DDL (Data Definition Language):
- DML (Data Manipulation Language):
- DCL (Data Control Language):

2.1. Categorías SQL

Consultas más comunes de cada categoría de bases sql

DDL (Data Definition Language):

-

Explicación:

DDL (Lenguaje de Definición de Datos) se utiliza para definir y modificar la estructura de objetos en una base de datos. Estos objetos pueden incluir bases de datos, tablas, vistas, índices, etc. Las consultas DDL se centran en la creación, modificación y eliminación de estos objetos, lo que define la estructura de la base de datos en sí misma.

-

Propósito:

Definir la estructura de la base de datos.

Crear, modificar y eliminar objetos de la base de datos.

-

1. Creación de Tablas:

```
CREATE TABLE Persona(
    nombre varchar,
    apellido varchar,
    ...
);
```

2. Modificación de Tablas (Agregar una Columna):

```
ALTER TABLE Persona
ADD fecha_nacimiento DATE;
```

3. Modificación de Tablas (Eliminar una Columna):

```
ALTER TABLE NombreTabla
DROP COLUMN ColumnaExistente;
```

4. Eliminación de Tablas:

```
DROP TABLE NombreTabla;
```

DML (Data Manipulation Language):

Explicación:

DML (Lenguaje de Manipulación de Datos) se utiliza para manipular los datos almacenados en la base de datos. Las consultas DML se centran en la inserción, actualización, eliminación y consulta de datos en las tablas.

Propósito:

Manipular los datos almacenados en la base de datos.

Insertar, actualizar, eliminar y consultar datos en las tablas.

1. Inserción de Datos:

```
INSERT INTO persona(nombre, apellido)
VALUES ('javier', 'lopez')
```

2. Actualización de Datos:

```
UPDATE NombreTabla
SET columna1 = valor1, columna2 = valor2, ...
WHERE condición;
```

3. Eliminación de Datos:

```
DELETE FROM NombreTabla
WHERE condición;
```

4. Consulta de Datos:

```
SELECT columna1, columna2, ...
FROM NombreTabla
WHERE condición;
```

DCL (Data Control Language):

Explicación:

DCL (Lenguaje de Control de Datos) se utiliza para gestionar los permisos y la seguridad en una base de datos. Las consultas DCL se centran en otorgar o revocar privilegios de acceso a los usuarios y en la gestión de roles de usuario.

Propósito:

Gestionar los permisos y la seguridad en la base de datos.

Conceder y revocar privilegios de acceso a los usuarios.

Gestionar roles de usuario.

1. Concesión de Privilegios:

```
GRANT tipo_privilegio ON objeto TO usuario;
```

2. Revocación de Privilegios:

```
REVOKE tipo_privilegio ON objeto FROM usuario;
```

3. Creación de Usuarios:

```
CREATE USER nombre_usuario IDENTIFIED BY 'contraseña';
```

4. Eliminación de Usuarios:

```
DROP USER nombre_usuario;
```