UNIDAD 3 - Consultas

Consultas

Uniendo Dos consultas con SQL

Unión de consultas con la Cláusula UNION en SQL

La cláusula **UNION** en SQL se utiliza para combinar los resultados de dos o más consultas `SELECT` en un único conjunto de resultados, incluyendo todas las filas que pertenecen a todas las consultas combinadas. Es crucial para el uso correcto de `UNION` que cada consulta involucrada en la unión tenga el mismo número de columnas en sus resultados, con tipos de datos compatibles en sus columnas correspondientes.

Características de UNION

1. **Número de Columnas**: Todas las consultas combinadas deben tener el mismo número de columnas. Por ejemplo, si la primera consulta selecciona 3 columnas, entonces la segunda consulta también debe seleccionar 3 columnas.
2. **Tipos de Datos Compatibles**: Las columnas que se están combinando en diferentes consultas deben tener tipos de datos compatibles. Por ejemplo, no puedes combinar directamente una columna de texto con una de números sin realizar alguna forma de conversión o ajuste, pero aunque es posible realizarlo, siempre lo mas conveniente es tener el mismo tipo de datos en las columnas de cada una de las consultas, y por cuestiones de legibilidad, es conveniente que las mismas tengan el mismo nombre de columna. Con esto logramos simplificar su seguimiento.

Ejemplo de Uso de UNION

Supongamos que tenemos dos tablas de juegos, `game_level_1` y `game_level_2`, que contienen información de juegos en diferentes niveles pero con estructura similar:

-- Selección de juegos del nivel 1 SELECT id_game, name, description FROM game_level_1 WHERE condition = 'X';

UNION

-- Selección de juegos del nivel 2 SELECT id_game, name, description FROM game_level_2 WHERE condition = 'Y';

En este ejemplo, ambos `SELECT` tienen el mismo número de columnas (tres en este caso: `id_game`, `name`, y `description`) y se asume que las columnas correspondientes en ambas consultas son del mismo tipo de datos.

Importancia del Orden y la Sintaxis

- **Orden de las Columnas**: El orden en que se seleccionan las columnas en cada consulta debe ser idéntico para asegurar que los datos de cada columna se alineen correctamente en los resultados combinados.
- **Consideración de Casos de Uso**: `UNION` es útil para casos donde necesitas obtener una lista combinada de elementos de diferentes fuentes que comparten una estructura similar, como podría ser combinar listas de elementos de diferentes categorías, regiones, o períodos de tiempo que se encuentran almacenados en tablas separadas.

Al utilizar `UNION`, SQL por defecto elimina las filas duplicadas. Si deseas incluir todas las filas duplicadas en los resultados, puedes utilizar `UNION ALL`, que puede ser más rápido ya que no necesita verificar duplicados.

A la hora de presentar los resultados, si las columnas no tienen el mismo nombre en las distintas consultas, el motor va a adoptar el nombre que contienen las columnas en la primer consulta.

Luego, al realizar un `ORDER BY` se va a aplicar sobre la totalidad de los resultados.

En resumen, `UNION` es una herramienta poderosa en SQL que permite la unificación de resultados de múltiples consultas, siempre y cuando se cumplan las reglas de compatibilidad y estructura de columnas. Su correcta utilización puede simplificar y agilizar la recuperación de datos de múltiples fuentes dentro de una base de datos.

Tipos de Datos en SQL

Repaso de los Principales Tipos de Datos en SQL

Tipos Numéricos

- **INT**: Para valores enteros sin decimales, ideal para identificadores o cantidades.
- **DECIMAL y NUMERIC**: Usados para números con precisión decimal, como precios.
- **FLOAT y DOUBLE**: Para cálculos que requieren muchos decimales, aunque no son recomendados para datos financieros debido a posibles errores de redondeo.

Tipos de Texto

- **CHAR y VARCHAR**: CHAR para textos de longitud fija y VARCHAR para textos de longitud variable.
- **TEXT**: Adecuado para textos largos como descripciones o comentarios.

Tipos de Fecha y Hora

- **DATE**: Solo almacena fechas.
- **TIME**: Solo registra tiempos.
- **DATETIME y TIMESTAMP**: Almacenan tanto la fecha como la hora; TIMESTAMP ajusta automáticamente las zonas horarias.

Otros Tipos de Datos

- **BOOLEAN**: Para valores verdadero o falso.
- **BINARY y VARBINARY**: Para datos binarios, como imágenes o archivos.

Consideraciones para la Elección de Tipos de Datos

- **Espacio de Almacenamiento**: Elegir el tipo de dato correcto puede reducir significativamente el espacio necesario.
- **Rendimiento**: Las operaciones son generalmente más rápidas con los tipos de datos adecuados.
- **Integridad de los Datos**: Ayuda a mantener la precisión y coherencia de los datos, evitando errores de formato.
- **Dimensionamiento:** Hay tipos de datos que requiere que se le indique el dimensionamiento, por ejemplo, el CHAR o VARCHAR debe indicarse su longitud y en el caso de los decimales la cantidad de dígitos y cantidad de decimales.

La correcta selección de tipos de datos es crucial para el diseño eficiente de bases de datos y para asegurar un rendimiento óptimo en las operaciones de consulta y manipulación de datos.

Cada motor implementa tipos de datos distintos, antes de crear las tablas hay que consultar cuales son los disponibles para el motor a utilizar en la versión a utilizar, incluso dentro de las mismas ediciones (Enterprise, Standard) hay tipos de datos que pueden variar.

Subconsultas

Para practicar...

Archivo SQL

Instrucciones para usar el archivo en ejercicios prácticos:

1. **Preparación:**

    - Asegúrate de tener instalado un sistema de gestión de bases de datos como MySQL.
    - Crea una nueva base de datos o selecciona una existente donde importarás la tabla `empleados`.

2. **Importación de datos:**

    - Abre tu cliente MySQL (como MySQL Workbench o una consola de comandos).
    - Ejecuta el contenido del archivo descargado para crear la tabla y poblarla con los datos de ejemplo.

3. **Ejercicios prácticos:**

    - Ejecuta las consultas SQL del guion utilizando la tabla `empleados`.
    - Por ejemplo:
        - Identifica empleados con el salario más alto usando una subconsulta:

          SELECT nombre, salario FROM empleados

          WHERE salario = (SELECT MAX(salario) FROM empleados);

        - Encuentra empleados con salarios superiores al promedio de su departamento:

          SELECT a.nombre, a.salario

          FROM empleados a

          WHERE a.salario > (SELECT AVG(salario) FROM empleados b
          WHERE b.departamento = a.departamento);

        - Explora más ejemplos del guion para practicar conceptos avanzados como subconsultas correlacionadas y anidadas.

Profundizando en el Uso del Operador LIKE

Uso del Operador LIKE en SQL

El operador **LIKE** en SQL se utiliza para realizar búsquedas de patrones dentro de cadenas de texto, permitiendo encontrar coincidencias parciales o completas en los registros.

Comodines en LIKE

- **% (Porcentaje)**: Sustituye a cualquier número de caracteres y es útil para búsquedas generales en cualquier posición de la cadena.
- **_ (Guion Bajo)**: Representa un único carácter y es útil cuando se conoce la posición exacta del carácter en la cadena.

Ejemplos Prácticos

- Para nombres que comienzan con "Al":

```
SELECT * FROM clientes WHERE nombre LIKE 'Al%';
```

- Para descripciones que contienen "libro":

```
SELECT * FROM productos WHERE descripcion LIKE '%libro%';
```

- Para apellidos que terminan en "son":

```
SELECT * FROM empleados WHERE apellido LIKE '%son';
```

- Para emails con formato específico:

```
SELECT * FROM usuarios WHERE email LIKE '_a%o@dominio.com';
```

Consideraciones Importantes

- **Sensibilidad al Caso**: Depende del sistema gestor de la base de datos; algunos son sensibles a mayúsculas y minúsculas.
- **Rendimiento**: El uso de comodines, especialmente al inicio del patrón (`%algo`), puede afectar el rendimiento al impedir el uso eficiente de índices.

El operador **LIKE** es una herramienta esencial para filtrar y analizar datos basados en criterios de texto en SQL.

Sublenguaje DDL

Introducción a DDL en SQL

Lenguaje de Definición de Datos (DDL) en SQL

El **Lenguaje de Definición de Datos** (DDL) es fundamental en SQL para la gestión de estructuras de base de datos. Proporciona las sentencias necesarias para crear, modificar, y eliminar objetos dentro de una base de datos, como tablas, índices, vistas, funciones, procedimientos, triggers y otros tipos de estructuras.

Funciones Principales de DDL

- **CREATE**: Utilizada para crear nuevos objetos en la base de datos. Por ejemplo, `CREATE TABLE` se emplea para crear una nueva tabla especificando sus columnas y restricciones.
- **DROP**: Esta sentencia elimina objetos existentes en la base de datos. Por ejemplo, `DROP TABLE` eliminaría completamente una tabla especificada, incluyendo todos sus datos.
- **ALTER**: Permite modificar la estructura de objetos existentes. Se puede usar para añadir, eliminar, o modificar columnas en una tabla o para cambiar otras propiedades de la tabla.

Aplicaciones de DDL

1. **Creación de Tablas**: Usando `CREATE TABLE`, se especifican las columnas de la tabla, sus tipos de datos, y posibles restricciones como claves primarias o valores por defecto. CREATE TABLE Employees (ID INT PRIMARY KEY, Name VARCHAR(100), HireDate DATE);
2. **Modificación de Tablas**: Con `ALTER TABLE`, se pueden hacer cambios como añadir nuevas columnas, cambiar tipos de datos de columnas existentes, o eliminar columnas. ALTER TABLE Employees ADD Email VARCHAR(255);
3. **Eliminación de Objetos**: `DROP TABLE` se utiliza para eliminar tablas y `DROP DATABASE` para eliminar bases de datos completas. DROP TABLE Employees;

Consideraciones al Usar DDL

- **Impacto Permanente**: Las operaciones de DDL tienen un impacto permanente en la estructura de la base de datos, por lo que deben usarse con precaución.
- **Seguridad y Control de Acceso**: Es importante controlar quién tiene permisos para ejecutar sentencias DDL, ya que implican cambios significativos en la estructura de la base de datos.
- **Planificación y Pruebas**: Antes de aplicar cambios significativos en un entorno de producción, estos deben ser cuidadosamente planificados y probados en un entorno de desarrollo.

DDL es esencial para la administración de bases de datos, permitiendo a los administradores y desarrolladores estructurar y reestructurar la base de datos conforme a las necesidades cambiantes de la aplicación y de la organización.

Si bien esta presente en todas las bases de datos, cada motor implementa pequeñas variaciones y adecuaciones a la hora de generar la sintaxis, es por eso que hay que revisar la documentación actualizada de cada uno de los motores.

Material Complementario

- Subconsultas SQL | Píldoras informáticas

- Subconsultas en SQL SERVER | CodeStack