*En esta sección, profundizaremos en las consultas SQL más complejas, que involucran múltiples tablas y subconsultas, para que los estudiantes estén mejor preparados para manejar conjuntos de datos más complejos y extraer información más significativa de ellos.*

*3.1 Uniendo tablas con JOIN*

*Las consultas que involucran múltiples tablas a menudo requieren la combinación de datos de dos o más tablas. La sentencia JOIN en SQL se utiliza para combinar dos o más tablas en una consulta. Es una operación fundamental en SQL y permite trabajar con conjuntos de datos complejos. Los cuatro tipos más comunes de JOIN son INNER JOIN, LEFT JOIN, RIGHT JOIN y FULL OUTER JOIN.*

* *INNER JOIN: combina las filas de dos tablas si hay una coincidencia en la clave primaria y devuelve sólo las filas que cumplen con la condición. Su sintaxis es la siguiente:*

| *SELECT columnas FROM tabla1 INNER JOIN tabla2 ON tabla1.columna = tabla2.columna* |
| --- |

* *LEFT JOIN: Combina todas las filas de la tabla de la izquierda (primera tabla mencionada) con las filas coincidentes de la tabla de la derecha (segunda tabla mencionada). Si no hay coincidencias, devuelve NULL. La sintaxis es la siguiente:*

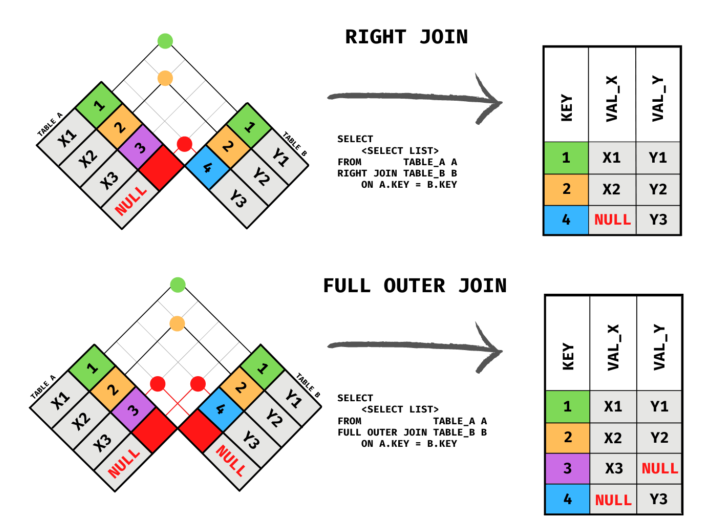
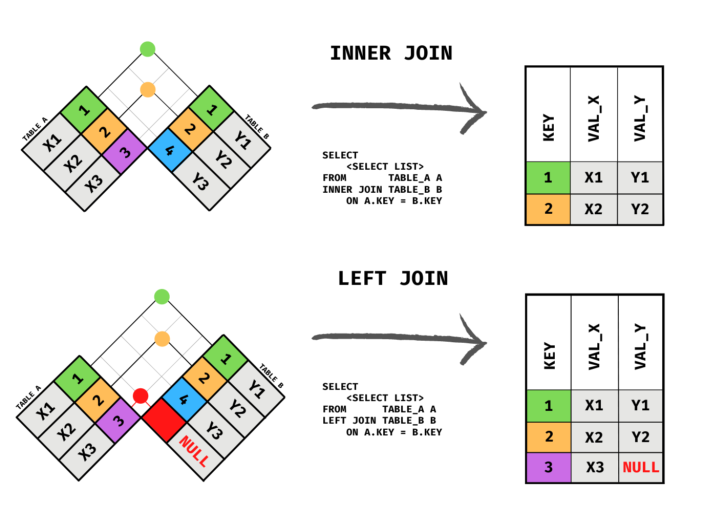
| *SELECT columnas FROM tabla1 LEFT JOIN tabla2 ON tabla1.columna = tabla2.columna* |
| --- |

* *RIGHT JOIN: Combina todas las filas de la tabla de la derecha (segunda tabla mencionada) con las filas coincidentes de la tabla de la izquierda (primera tabla mencionada). Si no hay coincidencias, devuelve NULL. La sintaxis es la siguiente:*

| *SELECT columnas FROM tabla1 RIGHT JOIN tabla2 ON tabla1.columna = tabla2.columna* |
| --- |

* *FULL OUTER JOIN: Combina todas las filas de ambas tablas, devolviendo NULL en cualquier columna donde no hay coincidencias. La sintaxis es la siguiente:*

| *SELECT columnas FROM tabla1 FULL OUTER JOIN tabla2 ON tabla1.columna = tabla2.columna* |
| --- |

*Para ejemplificar su funcionamiento, veamos los siguientes diagramas, en los cuales se describen los cuatro tipos de JOIN utilizados:*  


*El INNER JOIN nos trae una tabla con los registros 1 y 2, que se repiten en ambas tablas. El LEFT JOIN nos regresa una tabla con los registros 1, 2 y 3, que se encuentran en la tabla A, y dado que 3 no se encuentra en B, asigna un NULL en la columna VAL\_Y. Similarmente, el RIGHT JOIN nos regresa los registros de la tabla B 1, 2 y 4, poniendo en la columna VAL\_X un NULL. Por último, al hacer un FULL OUTER JOIN nos regresa una tabla con los registros de A (1, 2, 3) y de B (1, 2, 4) y en los registros no comunes (3 y 4) asigna el NULL correspondiente.*

*3.2 Subconsultas*

*Las subconsultas (también conocidas como subqueries) son consultas que se utilizan dentro de una consulta principal para obtener datos de una o más tablas. Se utilizan principalmente para filtrar datos en función de una condición en otra tabla o para calcular un valor que se utilizará en la consulta principal.*

*La sintaxis básica de una subconsulta es la siguiente:*

| *SELECT column1, column2, ... FROM table1 WHERE column\_name operator (  SELECT column\_name   FROM table\_name  WHERE condition);* |
| --- |

*En esta sintaxis, la subconsulta está encerrada entre paréntesis y se utiliza en lugar de un valor o lista de valores en la cláusula WHERE.*

*Las subconsultas se pueden utilizar en diferentes cláusulas de una consulta, como la cláusula WHERE, HAVING, SELECT y FROM. Algunas de las aplicaciones más comunes de las subconsultas son:*

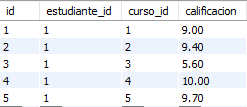
* *Filtrado de datos: se utilizan para filtrar los datos de una tabla en función de una condición en otra tabla.*
* *Cálculo de valores: se utilizan para calcular un valor que se utilizará en la consulta principal.*
* *Comprobación de la existencia: se utilizan para comprobar si una fila existe en otra tabla.*
* *Subconsultas correlacionadas: se utilizan para hacer referencia a valores de la tabla principal en la subconsulta.*

*A continuación, se presentan algunos ejemplos de subconsultas:*

* *Filtrado de datos: Supongamos que queremos obtener las calificaciones de un alumno “Juan Pérez”, pero no conocemos su Id, entonces utilizamos una subconsulta para filtrar la tabla de estudiantes y obtener su ID, y con ello filtrar la tabla de calificaciones.*

| *SELECT \* FROM calificaciones WHERE estudiante\_id in (SELECT id FROM estudiantes   WHERE nombre = 'Juan' AND apellido = 'Pérez');* |
| --- |

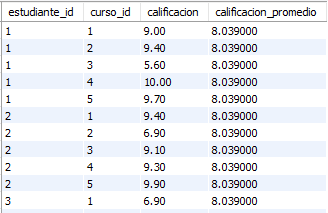
*Obtendremos la tabla entonces con las calificaciones del alumno:*

**

* *Cálculo de valores: En este ejemplo, la subconsulta se utiliza para calcular la calificación promedio de todos los alumnos (sin distinguir entre cursos) en la tabla de calificaciones y luego se utiliza en la consulta principal para mostrar el id del estudiante, el id del curso, la calificación correspondiente y la calificación promedio de todos los alumnos.*

| *SELECT estudiante\_id, curso\_id, calificacion,  (SELECT AVG(calificacion)   FROM calificaciones) AS calificacion\_promedio FROM calificaciones;* |
| --- |

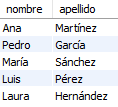
*La tabla generada será como la siguiente:*

**

* *Comprobación de la existencia: En este ejemplo, la subconsulta se utiliza para comprobar si cada maestro tiene al menos un curso asignado. Si existe un curso, el nombre del maestro se incluirá en los resultados de la consulta principal.*

| *SELECT nombre, apellido FROM maestros WHERE EXISTS (SELECT \* FROM cursos*  *WHERE maestros.id = maestro\_id);* |
| --- |

*El resultado es una tabla que contiene el nombre y apellido de todos los maestros que tienen al menos un curso asignado:*

**

*3.3 GROUP BY y HAVING*

*La sentencia GROUP BY en SQL se utiliza para agrupar filas que tienen el mismo valor en una o más columnas, permitiendo que se realicen cálculos de agregación como COUNT, SUM, AVG, MAX y MIN en grupos específicos de filas.*

*La sintaxis básica de la sentencia GROUP BY es la siguiente:*

| *SELECT columna1, columna2, columnaN, funcion\_agregacion(columna) FROM nombre\_tabla GROUP BY columna1, columna2, columnaN;* |
| --- |

*La sentencia HAVING se utiliza para filtrar los resultados de la consulta basándose en las funciones de agregación especificadas en la cláusula GROUP BY. A diferencia de la cláusula WHERE, que filtra las filas antes de la agrupación, la cláusula HAVING filtra los grupos de filas después de la agrupación. La sintaxis básica de la sentencia HAVING es la siguiente:*

| *SELECT columna1, columna2, columnaN, funcion\_agregacion(columna) FROM nombre\_tabla GROUP BY columna1, columna2, columnaN HAVING condicion;* |
| --- |

*Donde “condicion” es la condición que se utiliza para filtrar los grupos de filas en función de las funciones de agregación.*

*Por ejemplo, si queremos obtener la calificación promedio de los alumnos y la calificación máxima y mínima pero considerando solamente aquellos cuya calificación mínima sea de 7 (imaginemos que este grupo es candidato a una beca), podríamos utilizar la sentencia GROUP BY y HAVING de la siguiente manera:*

| *SELECT estudiante\_id,   AVG(calificacion) AS media,   MAX(calificacion) AS maxima,   MIN(calificacion) AS minima FROM calificaciones GROUP BY estudiante\_id HAVING minima > 7;* |
| --- |

*En este ejemplo, la cláusula GROUP BY agrupa las calificaciones por alumno y la cláusula HAVING filtra los grupos de filas para incluir sólo aquellos con calificación mínima mayor a 7. El resultado obtenido es el siguiente:*

**

*3.4 Expresiones de tabla comunes (CTE)*

*Las expresiones de tabla comunes (CTE, por sus siglas en inglés) son una característica de SQL que permiten crear tablas temporales que se pueden usar en una consulta. Las CTE son especialmente útiles para simplificar consultas complejas, hacer que las consultas sean más fáciles de leer y evitar la repetición de código.*

*La sintaxis básica de una CTE es la siguiente:*

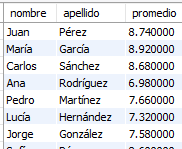
| *WITH nombre\_cte AS (  SELECT ...  FROM ...  WHERE ... ) SELECT ... FROM nombre\_cte* |
| --- |

*Donde “nombre\_cte” es el nombre que le das a la CTE, y la consulta dentro del paréntesis es la definición de la tabla temporal.*

*Un ejemplo de CTE podría ser el siguiente. Asume que deseamos obtener el promedio de los alumnos pero en la consulta deseamos agregar el nombre y el apellido y no solo el id del estudiante. La consulta para obtener esta tabla es la siguiente:*

| *WITH alumnos\_nombre AS (  SELECT estudiantes.id, nombre, apellido, curso\_id, calificacion  FROM estudiantes  LEFT JOIN calificaciones  ON estudiantes.id = calificaciones.estudiante\_id ) SELECT nombre, apellido, AVG(calificacion) AS promedio FROM alumnos\_nombre GROUP BY id;* |
| --- |

*El resultado es la siguiente tabla:*

**

*En este ejemplo, “alumnos\_nombre” es el nombre que le damos a la CTE que hace el JOIN entre la tabla estudiantes y la tabla calificaciones, para obtener el nombre y apellido de cada alumno. Luego, la consulta principal utiliza esta CTE para obtener los promedios de cada estudiante..*

*Las CTE también pueden ser recursivas, lo que significa que una CTE se puede definir en términos de sí misma. Por ejemplo, se puede utilizar una CTE recursiva para calcular la jerarquía de empleados de una empresa.*

*Supongamos que tenemos una tabla “employees” con información de los empleados de una empresa, donde cada registro contiene el id del empleado, el id de su supervisor, su nombre y su salario. Queremos crear una consulta que nos permita calcular el salario total de cada empleado, incluyendo los salarios de sus subordinados.*

*Para ello, podemos utilizar una CTE recursiva de la siguiente manera:*

| *WITH RECURSIVE employee\_cte (id, name, salary, manager\_id, level) AS (  -- CTE no recursiva para obtener los datos iniciales  SELECT id, name, salary, manager\_id, 0  FROM employees  WHERE manager\_id IS NULL  UNION ALL    -- CTE recursiva para unir los datos de los subordinados  SELECT e.id, e.name, e.salary, e.manager\_id, ec.level + 1  FROM employees e  INNER JOIN employee\_cte ec on e.manager\_id = ec.id ) SELECT id, name, SUM(salary) AS total\_salary FROM employee\_cte GROUP BY id, name ORDER BY total\_salary DESC;* |
| --- |

*En este ejemplo, la CTE “employee\_cte” se divide en dos partes: la primera es una CTE no recursiva que selecciona los datos iniciales de la tabla employees para los empleados que no tienen supervisor (“manager\_id” es NULL). La segunda parte es una CTE recursiva que une los datos de los subordinados utilizando un JOIN con la misma CTE, incrementando el nivel de jerarquía en 1 en cada iteración.*

*Finalmente, se realiza una consulta sobre la CTE “employee\_cte” para obtener el id, nombre y salario total de cada empleado (sumando los salarios de él y sus subordinados), agrupando por “id” y “name” y ordenando por salario total en orden descendente.*

*En general, las CTE son una herramienta poderosa para simplificar consultas complejas, mejorar la legibilidad del código y evitar la repetición de código. Sin embargo, es importante tener en cuenta que algunas bases de datos pueden tener limitaciones en la cantidad de CTE que se pueden utilizar en una consulta, por lo que es importante ser consciente de las limitaciones de la base de datos que se está utilizando.*

*En resumen, las consultas avanzadas de SQL permiten a los estudiantes manejar conjuntos de datos más complejos y extraer información más significativa de ellos. Al comprender los JOIN, subconsultas, GROUP BY y HAVING, así como las expresiones de tabla comunes, los estudiantes tendrán una comprensión sólida de las herramientas disponibles para procesar y analizar datos de manera efectiva.*