Sintaxis General:

Las cosas que están entre corchetes son opcionales.

```
SELECT [ALL|DISTINCT] <lista de seleccion>
        FROM <clausula de origen>
        [WHERE <condicion>]
        [GROUP BY <expresion> [HAVING <condicion>]]
        [{UNION| UNION ALL|MINUS|INTERSECT} <query>]
        [ORDER BY <expresion> [ASC|DESC]];
[INNER|OUTER|RIGHT|LEFT|FULL|CROSS JOIN] OTRA CONSUILTA
```

Explicación paso a paso:

- `<lista de seleccion>`: es la lista de columnas que quieres seleccionar, separadas por comas
- `[ALL|DISTINCT]`. Si no se especifica se usa ALL por defecto.
  - **ALL:** Cuando se usa `[ALL]`, la consulta devolverá todos los registros del conjunto de resultados, **incluidas las filas duplicadas**. En otras palabras, si hay duplicados en el conjunto de resultados, se mostrarán todas las ocurrencias
  - **DISTINCT:** Cuando se usa `[DISTINCT]`, la consulta devolverá solo los registros únicos del conjunto de resultados, **eliminando las filas duplicadas**. Esto significa que solo se mostrará una ocurrencia de cada combinación única de valores en las columnas seleccionadas.
- `FROM <clausula de origen>` : la **clausula de origen es la tabla o query de la que se quiere extraer la información**. Puede ser una tabla como tal o otra consulta (consulta = query), de esta forma se pueden agrupar consultas dentro de otras consultas, ej: `SELECT Cantidad FROM (SELECT Cantidad, Producto FROM Ventas);`. Este ejemplo no es útil ya que se puede hacer directamente en una sola consulta pero eso, se puede hacer. Los paréntesis son por claridad pero no son necesarios
- `[WHERE <condicion>]`: es la condición que se quiere que cumplan los elementos devueltos, ej: `SELECT Nombre, Edad FROM Clientes WHERE Edad > 18 AND Nombre = 'Juan';` Devolvera el nombre y edad de todos (aun que haya duplicados) clientes llamados Juan mayores de edad.
  - Algunas condiciones utiles:
    1. **Igualdad:**
       - `columna = valor`: Selecciona filas donde el valor de la columna sea igual al valor especificado.
    2. **Desigualdad:**
       - `columna <> valor` o `columna ≠ valor`: Selecciona filas donde el valor de la columna no sea igual al valor especificado.
    3. **Mayor que y menor que:**

- `columna > valor`: Selecciona filas donde el valor de la columna sea mayor que el valor especificado.
- `columna < valor`: Selecciona filas donde el valor de la columna sea menor que el valor especificado.

4. **Mayor o igual y menor o igual:**
   - `columna ⩾ valor`: Selecciona filas donde el valor de la columna sea mayor o igual al valor especificado.
   - `columna ⩽ valor`: Selecciona filas donde el valor de la columna sea menor o igual al valor especificado.
5. **IN:**
   - `columna IN (valor1, valor2, ...)`: Selecciona filas donde el valor de la columna coincida con cualquiera de los valores especificados.
6. **BETWEEN:**
   - `columna BETWEEN valor1 AND valor2`: Selecciona filas donde el valor de la columna esté dentro del rango especificado.
7. **LIKE:**
   - `columna LIKE 'patrón'`: Selecciona filas donde el valor de la columna coincida con el patrón especificado (puede incluir comodines `%` y `_`).
8. **IS NULL / IS NOT NULL:**
   - `columna IS NULL`: Selecciona filas donde el valor de la columna sea nulo.
   - `columna IS NOT NULL`: Selecciona filas donde el valor de la columna no sea nulo.
9. **AND y OR:**
   - `condición1 AND condición2`: Selecciona filas donde ambas condiciones sean verdaderas.
   - `condición1 OR condición2`: Selecciona filas donde al menos una de las condiciones sea verdadera.
10. **NOT:**
    - `NOT condición`: Invierte la condición, seleccionando filas donde la condición sea falsa.

- **`[GROUP BY <expresion> [HAVING <condicion>]]`** Cuando se usan ciertas clausulas en la lista de seleccion se debe especificar como se agrupa esa clausula, se entiende mejor con un ejemplo: `SELECT Producto, SUM(Cantidad) AS TotalVentas FROM Ventas GROUP BY Producto HAVING SUM(Cantidad) ⩾ 5;`. En esta consulta se suma la cantidad de todos los productos que se llamen igual y se muestran aquellos cuya SUM(Cantidad) >= 5.
- **`[{UNION| UNION ALL|MINUS|INTERSECT} <query>]`**
  1. **UNION:** La operación `UNION` se utiliza para combinar los resultados de dos o más consultas en un solo conjunto de resultados. Esta operación eliminará automáticamente las filas duplicadas en el conjunto de resultados combinado.
  2. **UNION ALL:** Similar a `UNION`, pero a diferencia de `UNION`, `UNION ALL` no eliminará las filas duplicadas. El conjunto de resultados combinado contendrá todas las filas de ambas consultas, incluidas las filas duplicadas.
  3. **MINUS:** La operación `MINUS` (también conocida como `EXCEPT` en algunos sistemas) se utiliza para comparar el conjunto de resultados de la primera consulta con el conjunto de

resultados de la segunda consulta. Devolverá todas las filas que estén en el conjunto de resultados de la primera consulta pero no en el conjunto de resultados de la segunda consulta.

4. **INTERSECT:** La operación `INTERSECT` se utiliza para encontrar las filas que son comunes a ambos conjuntos de resultados de las dos consultas. Devolverá solo las filas que aparecen en ambas consultas.
   - Ejemplo: `SELECT Producto FROM Ventas_2019 UNION SELECT Producto FROM Ventas_2020;` Esto te dará una lista de productos vendidos en ambos años sin duplicados.

- **`ORDER BY`** Se usa para establecer el orden en el que se organiza la query, ej: `SELECT Nombre, Categoría, Precio FROM Productos ORDER BY Categoría, Precio;`. Esto ordenará los resultados primero por la columna "Categoría" en orden ascendente y luego, dentro de cada categoría, por la columna "Precio" en orden ascendente.
- **`[[INNER|OUTER|RIGHT|LEFT|FULL|CROSS|SELF] JOIN]`**: Existen varios tipos de joins (uniones) que se utilizan para combinar datos de dos o más tablas en una consulta. Si no se especifica se usa inner join por defecto.
  1. **INNER JOIN:**
     - Condición de Unión: `ON tabla1.columna = tabla2.columna`
     - Descripción: Devuelve filas que tienen coincidencias en ambas tablas en función de la condición de unión especificada.
  2. **LEFT JOIN (LEFT OUTER JOIN):**
     - Condición de Unión: `ON tabla1.columna = tabla2.columna`
     - Descripción: Devuelve todas las filas de la tabla izquierda y las filas coincidentes de la tabla derecha. Si no hay coincidencias en la tabla derecha, se rellenan con valores NULL.
  3. **RIGHT JOIN (RIGHT OUTER JOIN):**
     - Condición de Unión: `ON tabla1.columna = tabla2.columna`
     - Descripción: Devuelve todas las filas de la tabla derecha y las filas coincidentes de la tabla izquierda. Las filas sin coincidencias en la tabla izquierda se rellenan con valores NULL.
  4. **FULL JOIN (FULL OUTER JOIN):**
     - Condición de Unión: `ON tabla1.columna = tabla2.columna`
     - Descripción: Devuelve todas las filas de ambas tablas, incluidas las coincidencias y las no coincidencias. Las filas sin coincidencias se rellenan con valores NULL en ambos lados.
  5. **CROSS JOIN (Cartesian Join):**
     - Condición de Unión: No se especifica.
     - Descripción: Combina todas las filas de una tabla con todas las filas de otra tabla, creando un producto cartesiano.
  6. **SELF JOIN:**

     - Condición de Unión: `ON tabla1.columna = tabla2.columna`

     - Descripción: Se utiliza para combinar filas dentro de la misma tabla, generalmente utilizando una columna que se relaciona con otra columna en la misma tabla.

> Nota: cuando hay varias querys unidas por un join y tienen columnas que se llaman igual, debes diferenciar cada columna como `tabla.columna`

Ejemplos varios

Supongamos que tenemos una tabla llamada "Empleados" con las columnas "ID", "Nombre", "Salario" y "Departamento".

1. Seleccionar nombres y salarios de empleados cuyo salario sea mayor a $5000:

```
SELECT Nombre, Salario
FROM Empleados
WHERE Salario > 5000;
```

2. Contar el número de empleados por departamento y mostrar solo aquellos con al menos 3 empleados:
   (nota: AS renombra una columna para que en vez de llamarse count(*) tenga un nombre entendible)

```
SELECT Departamento, COUNT(*) AS CantidadEmpleados
FROM Empleados
GROUP BY Departamento
HAVING COUNT(*) ⩾ 3;
```

3. Mostrar la lista de nombres de empleados de los departamentos "Ventas" y "Marketing":

```
SELECT Nombre
FROM Empleados
WHERE Departamento IN ('Ventas', 'Marketing');
```

4. Combinar la lista de empleados de dos departamentos diferentes usando `UNION`:

```
SELECT Nombre, Departamento
FROM Empleados
WHERE Departamento = 'Ventas'
UNION
SELECT Nombre, Departamento
FROM Empleados
WHERE Departamento = 'Marketing';
```

5. Mostrar los empleados que trabajan en el departamento "Ventas" pero no en el departamento "Marketing" usando `MINUS` (o `EXCEPT` en algunos sistemas):

```
SELECT Nombre, Departamento
FROM Empleados
WHERE Departamento = 'Ventas'
MINUS
SELECT Nombre, Departamento
```

```
FROM Empleados
WHERE Departamento = 'Marketing';
```

6. Encontrar los nombres de empleados que trabajan en ambos departamentos "Ventas" y "Marketing" usando `INTERSECT`:

```
SELECT Nombre
FROM Empleados
WHERE Departamento = 'Ventas'
INTERSECT
SELECT Nombre
FROM Empleados
WHERE Departamento = 'Marketing';
```

7. Mostrar la lista de empleados ordenados por salario de forma descendente:

```
SELECT Nombre, Salario
FROM Empleados
ORDER BY Salario DESC;
```

8. Calcular el salario promedio por departamento y mostrar solo aquellos con un salario promedio mayor a $6000:

```
SELECT Departamento, AVG(Salario) AS SalarioPromedio
FROM Empleados
GROUP BY Departamento
HAVING AVG(Salario) > 6000;
```

Ejemplos mas complejos, de la practica:

Pueden ser muy lio sin el contexto pero es para que se vea lo jodidamente rebuscado que se puede volver una consulta y en ellos se hace uso de todas las funcionalidades de las consultas.

1. Forma parte de una vista cuya descripción es la siguiente:
   - Vista fans (operatividad completa): asistentes a más de un concierto del intérprete actual (si el intérprete tiene menos de dos conciertos, no tendrá fans). Se aportará el email, nombre completo y edad de los fans. Si se borra un fan de esta vista, no se deben eliminar sus datos (ni de la tabla clientes ni de la tabla asistentes) pero ese cliente será “vetado” (dejará de aparecer en esta vista, aunque siga almacenado como cliente en la tabla global; para este fin, se puede crear una tabla adicional que recoja qué clientes están vetados para qué intérpretes). Si se inserta un fan, se insertará como cliente (si no existía ya); si tiene menos de dos asistencias a conciertos del intérprete actual, se insertará su asistencia al último o los dos últimos conciertos. Si era un fan “vetado”, dejará de estar vetado (volverá a aparecer en la vista). Las modificaciones (update) sobre esta vista no deberán tener efecto.

```
SELECT CLIENTS.e_mail as e_mail, CLIENTS.NAME as name, ROUND(((SYSDATE-
CLIENTS.birthdate)/365.24)) AS EDAD
```

```
    FROM (
        SELECT ATTENDANCES.CLIENT AS CLIENT, COUNT(*) AS n_asistencias,
ATTENDANCES.PERFORMER AS PERFORMER
        FROM ATTENDANCES
        WHERE ATTENDANCES.PERFORMER=melopack.get_interprete_actual()
        GROUP BY ATTENDANCES.CLIENT, ATTENDANCES.PERFORMER
        HAVING COUNT(*)>1
    ) ASISTENCIAS JOIN CLIENTS ON ASISTENCIAS.CLIENT=CLIENTS.e_mail
    WHERE ASISTENCIAS.CLIENT=CLIENTS.e_mail AND NOT EXISTS(
        SELECT * FROM fans_vetados WHERE fans_vetados.email=CLIENTS.e_mail
AND fans_vetados.PERFORMER=ASISTENCIAS.PERFORMER
    )
    AND ASISTENCIAS.PERFORMER=melopack.get_interprete_actual()
    GROUP BY CLIENTS.e_mail, CLIENTS.NAME, ((SYSDATE-
ClIENTS.birthdate)/365.24)
);
```

2. Vista events: actividad en conciertos del intérprete actual, con una fila por cada mes y año (con algún concierto de ese intérprete), incluyendo la cantidad de conciertos (de ese mes), cantidad de espectadores, la duración media de los conciertos, y cantidad media de interpretaciones.

```
SELECT
        conciertos.mes_y_ano,
        n_conciertos,
        n_espectadores_totales,
        duracion_media_de_conciertos,
        ROUND(NVL(n_actuaciones / n_conciertos, 0),2) AS
n_media_interpretaciones
    FROM (
            SELECT
                TO_CHAR(PERFORMANCES.when, 'MM/YYYY') AS mes_y_ano,
                COUNT (*) AS n_actuaciones
            FROM
                PERFORMANCES
            WHERE
                PERFORMANCES.performer = melopack.get_interprete_actual()
            GROUP BY
                TO_CHAR(PERFORMANCES.when, 'MM/YYYY')
        ) AS actuaciones
    RIGHT JOIN(
        SELECT
            TO_CHAR(CONCERTS.when, 'MM/YYYY') AS mes_y_ano,
            COUNT (*) AS n_conciertos,
            SUM(attendance) AS n_espectadores_totales,
            ROUND(AVG(duration),2) AS duracion_media_de_conciertos
        FROM
            CONCERTS
        WHERE
            CONCERTS.performer = melopack.get_interprete_actual()
        GROUP BY
            TO_CHAR(CONCERTS.when, 'MM/YYYY')
```

```
    ) AS conciertos
    ON conciertos.mes_y_ano = actuaciones.mes_y_ano)
```