Las instrucciones SQL3 de manipulación pueden operar de tres modos:

1. Interactivo (proporcionando instrucciones SQL directamente).
2. SQL embebido: instrucciones imbuidas en lenguaje anfitrión (C, JAVA,…).
3. Módulos: llamadas explícitas a procedimientos desde procesos externos.
   Operaciones de Actualización:

- Inserción de tuplas (INSERT)
- Borrado de tuplas (DELETE)
- Modificación de tuplas (UPDATE
  Operaciones de Recuperacion
- Consulta o *Query* (SELECT)

Gestión transaccional en PL/SQL

Transacción: conjunto de instrucciones de actualización que deben ser llevadas a cabo de modo atómico (como conjunto, “o todo o nada”).

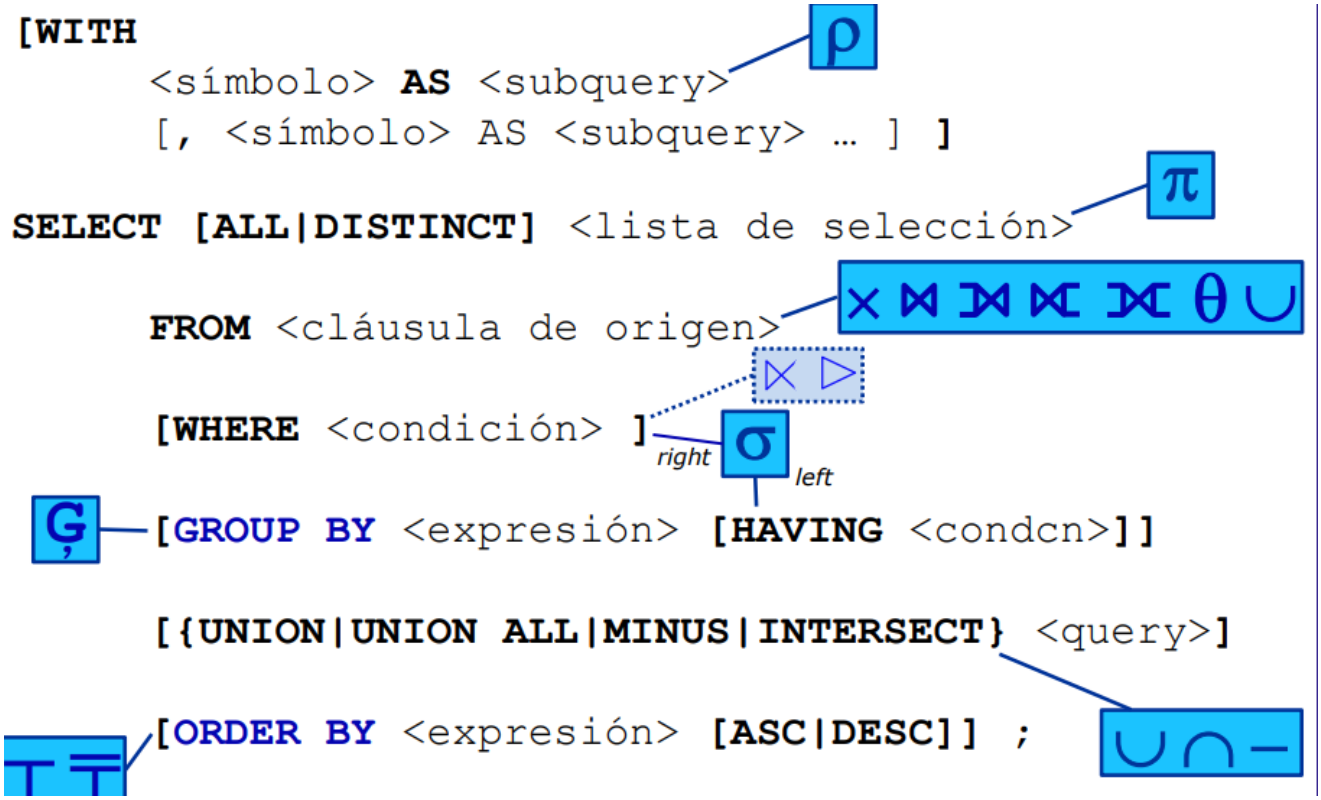
Instrucciones: COMMIT (realizar) y ROLLBACK (deshacer)

- COMMIT [WORK]
- ROLLBACK [WORK] [TO [SAVEPOINT] <savepoint>]
- SAVEPOINT <savepoint>

SINTAXIS QUERY

```
[WITH
         <simbolo> AS <subquery>
         [,simbolo AS <subquieru> … ]]
        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]];
```

lo que está entre corchetes es opcional



Proyeccion en el query

- la proyección se refleja en la <LISTA DE SELECCION>
  - Lista de datos (del workspace) separados por comas.
  - Puede ser todo el área de trabajo (*) o bien incluir:
    - atributos del esquema de relación del área de trabajo
    - pseudo-columnas, como ROWNUM y table.ROWID,...
    - constantes (como 1 o 'X') y variables ligadas (:NEW, …)
    - funciones: aplicadas sobre lo anterior (o nularias)
      - aritméticas (+, -, …), strings (||, SUBSTR, …), codificación (CASE, NVL, …), conversión (TO_CHAR, …), sistema (SYSDATE, USER, …), …
      - de agregación (reciben un colectivo y devuelven un solo valor)
      - funciones compiladas (de usuario o procedentes de paquetes)
- Admite renombrado elemento AS alias

Area de trabajo (workspace)

Es una tabla temporal (vinculada a la ubicación de los datos)

- La cláusula FROM define el área de trabajo (que es una tabla)
- Puede componerse de una tabla, o varias tablas combinadas.
- Dado que el área de trabajo es una tabla, un caso particular de tabla en la clausula FROM es otro área de trabajo, es decir, otra consulta (subquery)
- Admite renombrado (obligatorio en self-joins)
- Existen diversas combinaciones

Combinaciones

Combinación Elemental: el Producto Cartesiano

```
- FROM Gente CROSS JOIN Clientes… $\equiv$ FROM Gente, Clien
```

Combinación General JOIN:

todas las columnas de ambas tablas

- FROM people [INNER] JOIN clients [ON (= [AND...])] - sin especificación: equijoin por todos los pares de columnas que cumplan la expresion (=,!=,<,>,...)

Combinación natural:

no duplica las columnas incluidas en la igualdad

- FROM X NATURAL [INNER] JOIN Y...
  - Combinación Natural por pares de columnas que se llamen igual en ambas tablas
- FROM X [INNER] JOIN Y USING (<columns>)...
  - Combinación Natural por pares de columnas especificados (que se llaman igual…)

Consultas: otras combinaciones

Combinación sin pérdidas, Combinación Externa (outer):

- FROM Gente {LEFT|RIGHT|FULL} [OUTER] JOIN Clientes [USING <\columnas> | ON <col_a>=<col_b> [AND...]]
  - Mismo uso que la (inner) JOIN; 'outer' es opcional (recomendable por claridad)...
- FROM X LEFT OUTER JOIN Y...
  - Combinación Externa por la izquierda (se respetan todas las tuplas de la primera tabla)
- FROM X RIGHT JOIN Y...
  - Combinación Externa por la derecha (se respetan todas las tuplas de la segunda tabla)
- ... FROM X FULL OUTER JOIN Y...
  - Combinación Externa completa (se respetan todas las tuplas de ambas tablas)

Combinación por una union:

- FROM Gente UNION JOIN Clientes
  - Observación: ambos esquemas deben ser compatibles

Expresión condicional WHERE

(WHERE) puede ser:

- una comparación de expresiones (=, !=, <, >, <=, >=)

- comprobación de inclusion test (en un rango): <expresión> [NOT] BETWEEN <expresión> AND <expresión>
- comprobación de valor nulo: <expresión> IS [NOT] NULL
- test de semejanza (patrón): <expresión_char> [NOT] LIKE <patrón>
- expresión lógica: {NOT, AND, OR} a partir de otras expresiones condicionales
- test de existencia: EXISTS subquery
- test de inclusión (en un conjunto dado, o en una subquery): <expresión> [NOT] IN {<expresión_list>|subquery}
  SUBQUERIES en la cláusula WHERE: pueden ser ineficientes
- EXISTS se detiene en el primer encaje, pero se ejecuta anidado siempre.
- IN puede optimizarse (no ejecutar anidado); conviene moverlo a la cl. WITH
- NOT IN puede usarse en anti-combinaciones (si se hace de modo eficiente)

Agrupación

La cláusula GROUP BY define el criterio de agrupación.

El área de trabajo agrupada puede incluir columnas del criterio de agrupación y funciones de agregación sobre el resto.

Algunas funciones de agregación:

- COUNT, AVG, SUM, MIN, MAX, CONCAT (LISTAGG), ...
- MEDIAN, VARIANCE, STDDEV, CORR, COVAR, etc.
  Admiten dos tipos de condiciones:
- condición individual (WHERE) ejercida antes de agrupar;
- condición colectiva (HAVING) sobre la tabla ya agrupada

Ejemplos de Agrupacion y orden

Curso	Nombre	Apellido	ptos.	Edad
2º	Fulano	Pérez	113	32
1º	Mengana	Gómez	47	19
1º	Zutano	Smith	89	44
2º	Peranceja	García	45	10
1º	John	Doe	67	78

```
SELECT curso, count('x'),
       avg(edad), sum(ptos)
       FROM Gente
       GROUP BY curso;
```

Curso	count	AVG(edad)	SUM(ptos)
2º	2	21	158
1º	3	47	203

```
SELECT * FROM Gente ORDER BY apellido;
```

Curso	Nombre	Apellido	DNI	Edad
1º	John	Doe	6789	78
2º	Peranceja	García	2345	10
1º	Mengana	Gómez	4567	19
2º	Fulano	Pérez	0123	32
1º	Zutano	Smith	8901	45

Resumen

Sintaxis General:

Las cosas que están entre corchetes son opcionales.
Las cosas que estan entre <> es

```
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.
- `[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)

```sql
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":

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

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

```sql
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):

```sql
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`:

```sql
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:

```sql
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)
```