Select

```
SELECT [ALL|DISTINCT]
       { * | expr_1 [AS c_alias_1] [, ...
             [, expr_k [AS c_alias_k]]]}
FROM table_name_1 [t_alias_1]
     [, ... [, table_name_n [t_alias_n]]]
[WHERE condition]
[GROUP BY name_of_attr_i
          [,... [, name_of_attr_j]] [HAVING condition]]
[{UNION [ALL] | INTERSECT | EXCEPT} SELECT ...]
[ORDER BY name_of_attr_i [ASC|DESC]
          [, ... [, name_of_attr_j [ASC|DESC]]]];
```

Select sencillas. Sobre las sgtes bases

```
PROVE       PNUM|  PNOM    |  PCIUD
           -----+---------+--------
             1  |  Maria  | Rosario
             2  |  Juan   | Paraná
             3  |  Ana    | Santa Fe
             4  |  Pedro  | Cordoba

ARTI        ANUM|  ANOM       |  APRE
           -----+-------------+---------
             1  |  Tornillos  |   10
             2  |  Tuercas    |    8
             3  |  Aros       |   15
             4  |  Levas      |   25

VENTAS   PNUM| ANUM
        -----+-----
          1  |  1
          1  |  2
          2  |  4
          3  |  1
          3  |  3
          4  |  2
          4  |  3
          4  |  4
```

Ejemplo 4. Query sencilla con cualificación

Para recuperar todas las tuplas de la tabla ARTI donde el atributo APRE es mayor que 10, formularemos la siguiente consulta:

```
SELECT * FROM ARTI
  WHERE APRE > 10;
```

y obtenemos la siguiente tabla:

```
 ANUM|  ANOM       |  APRE
-----+-------------+--------
  3  |  Aros       |   15
  4  |  Levas      |   25
```

Utilizando "*" en la instrucción SELECT solicitaremos todos los atributos de la tabla. Si queremos recuperar sólo los atributos ANOM y APRE de la tabla ARTI utilizaremos la instrucción:

```
SELECT ANOM, APRE  FROM ARTI WHERE APRE > 10;
```

En este caso el resultado es:

```
 ANOM        |  APRE
-------------+--------
 Aros        |   15
 Levas       |   25
```

Nótese que la SELECT SQL corresponde a la "proyección" en álgebra relaciona, no a la "selección"

Las cualificaciones en la clausula WHERE pueden también conectarse lógicamente utilizando las palabras claves OR, AND, y NOT:

```
SELECT ANOM, APRE
FROM ARTI
WHERE ANOM = 'Aros' AND
      (APRE = 0 OR APRE < 15);
```

dará como resultado:

```
 ANOM        |  APRE
-------------+--------
 Aros        |   15
```

Las operaciones aritméticas se pueden utilizar en la lista de objetivos y en la clausula WHERE. Por ejemplo, si queremos conocer cuanto cuestan si tomamos dos piezas de un artículo, podríamos utilizar la siguiente consulta:

```
SELECT ANOM, APRE * 2 AS DOBLE
FROM ARTI
WHERE APRE * 2 < 50;
```

y obtenemos:

```
 ANOM        |  DOBLE
-------------+---------
 Tornillos   |    20
 Tuercas     |    16
 Aros        |    30
```

Nótese que la palabra DOBLE tras la palabra clave AS es el nuevo título de la segunda columna. Esta técnica puede utilizarse para cada elemento de la lista objetivo para asignar un nuevo título a la columna resultante. Este nuevo título recibe el calificativo de "un alias". El alias no puede utilizarse en todo el resto de la consulta.

Joins (Cruces)

El siguiente ejemplo muestra como las *joins (cruces)* se realizan en SQL.

Para cruzar tres tablas PROVE, ARTI y VENTAS a través de sus atributos comunes, formularemos la siguiente instrucción:

```
SELECT P.PNOM, A.ANOM
FROM PROVE P, ARTI A, VENTAS V
WHERE P.PNUM = V.PNUM AND
      A.ANUM = V.ANUM;
```

y obtendremos la siguiente tabla como resultado:

```
 PNOM  | ANOM
-------+----------
 María | Tornillos
 María | Tuercas
 Juan  | Levas
 Ana   | Tornillos
 Ana   | Aros
 Pedro | Tuercas
 Pedro | Aros
 Pedro | Levas
```

En la clausula FROM hemos introducido un alias al nombre para cada relación porque hay atributos con nombre común (PNUM y ANUM) en las relaciones. Ahora podemos distinguir entre los atributos con nombre común simplificando la adicción de un prefijo al nombre del atributo con el nombre del alias seguido de un punto. La join se calcula de la misma forma, tal como se muestra en *Una Inner Join (Una Join Interna)*. Primero el producto cartesiano: PROVE × ARTI × VENTAS Ahora seleccionamos únicamente aquellas tuplas que satisfagan las condiciones dadas en la clausula WHERE (es decir, los atributos con nombre común deben ser iguales). Finalmente eliminamos las columnas repetidas (P.PNOM, A.ANOM).

Operadores Agregados

SQL proporciona operadores agregados (como son AVG, COUNT, SUM, MIN, MAX) que toman el nombre de un atributo como argumento. El valor del operador agregado se calcula sobre todos los valores de la columna especificada en la tabla completa. Si se especifican grupos en la consulta, el cálculo se hace sólo sobre los valores de cada grupo (vean la siguiente sección).

Ejemplo 5. Aggregates

Si queremos conocer el coste promedio de todos los artículos de la tabla PART, utilizaremos la siguiente consulta:

```
SELECT AVG(APRE) AS AVG_APRE  FROM ARTI;
```

El resultado es:

```
AVG_APRE
-----------
   14.5
```

Si queremos conocer cuantos artículos se recogen en la tabla ARTI, utilizaremos la instrucción:

```
SELECT COUNT(ANUM)
FROM ARTI;
```

y obtendremos:

```
COUNT
-------
   4
```

Agregación por Grupos

SQL nos permite particionar las tuplas de una tabla en grupos. En estas condiciones, los operadores agregados descritos antes pueden aplicarse a los grupos (es decir, el valor del operador agregado no se calculan sobre todos los valores de la columna especificada, sino sobre todos los valores de un grupo. El operador agregado se calcula individualmente para cada grupo).

El particionamiento de las tuplas en grupos se hace utilizando las palabras clave **GROUP BY** seguidas de una lista de atributos que definen los grupos. Si tenemos **GROUP BY A_1, □, A_k** habremos particionado la relación en grupos, de tal modo que dos tuplas son del mismo grupo si y sólo si tienen el mismo valor en sus atributos A_1, □, A_k.

Ejemplo 6. Agregados

Si queremos conocer cuántos artículos han sido vendidos por cada proveedor formularemos la consulta:

```
SELECT P.PNUM, P.PNOM, COUNT(V.PNUM)
FROM PROVE P, VENTAS V
WHERE P.PNUM = V.PNUM
GROUP BY P.PNUM, P.PNOM;
```

y obtendremos:

```
PNUM | PNOM   | COUNT
-----+--------+-------
  1  | María  |   2
  2  | Juan   |   1
  3  | Ana    |   2
  4  | Pedro  |   3
```

Demos ahora una mirada a lo que está ocurriendo aquí. Primero, la join de las tablas PROVE y VENTAS:

P.PNUM	P.PNOM	V.PNUM
1	María	1
1	María	2
2	Juan	4
3	Ana	1
3	Ana	3
4	Pedro	2
4	Pedro	3
4	Pedro	4

Ahora particionamos las tuplas en grupos reuniendo todas las tuplas que tiene el mismo atributo en S.SNO y S.SNAME:

P.PNUM	P.PNOM	V.PNUM
1	María	1
		2
2	Juan	4
3	Ana	1
		3
4	Pedro	2
		3
		4

En nuestro ejemplo, obtenemos cuatro grupos y ahora podemos aplicar el operador agregado COUNT para cada grupo, obteniendo el resultado total de la consulta dada anteriormente.

Nótese que para el resultado de una consulta utilizando GROUP BY y operadores agregados para dar sentido a los atributos agrupados, debemos primero obtener la lista objetivo. Los demás atributos que no aparecen en la clausula GROUP BY se seleccionarán utilizando una función agregada. Por otro lado, no se pueden utilizar funciones agregadas en atributos que aparecen en la clausula GROUP BY.

Having

La clausula HAVING trabaja de forma muy parecida a la clausula WHERE, y se utiliza para considerar sólo aquellos grupos que satisfagan la cualificación dada en la misma. Las expresiones permitidas en la clausula HAVING deben involucrar funcionen agregadas. Cada expresión que utilice sólo atributos planos deberá recogerse en la clausula WHERE. Por otro lado, toda expresión que involucre funciones agregadas debe aparecer en la clausula HAVING.

Ejemplo 7. Having

Si queremos solamente los proveedores que venden más de un artículo, utilizaremos la consulta:

```
SELECT P.PNUM, P.PNOM, COUNT(V.PNUM)
FROM PROVE P, VENTAS V
WHERE P.PNUM = V.PNUM
GROUP BY P.PNUM, P.PNOM
HAVING COUNT(V.PNUM) > 1;
```

y obtendremos:

```
 PNUM| PNOM   | COUNT
-----+-------+-------
  1  | María |   2
  3  | Ana   |   2
  4  | Pedro  |   3
```

Subconsultas

En las clausulas WHERE y HAVING se permite el uso de subconsultas (subselects) en cualquier lugar donde se espere un valor. En este caso, el valor debe derivar de la evaluación previa de la subconsulta. El uso de subconsultas amplía el poder expresivo de SQL.

Ejemplo 8. Subselect

Si queremos conocer los artículos que tienen mayor precio que el artículo llamado 'Tornillos', utilizaremos la consulta:

```
SELECT *
FROM ARTI
WHERE APRE > (SELECT APRE FROM ARTI
              WHERE ANOM='Tornillos');
```

El resultado será:

```
 ANUM|  ANOM        |  APRE
-----+-------------+--------
  3  |  Aros        |   15
  4  |  Levas       |   25
```

Cuando revisamos la consulta anterior, podemos ver la palabra clave SELECT dos veces. La primera al principio de la consulta - a la que nos referiremos como la SELECT externa - y la segunda en la clausula WHERE, donde empieza una consulta anidada - nos referiremos a ella como la SELECT interna. Para cada tupla de la SELECT externa, la SELECT interna deberá ser evaluada. Tras cada

evaluación, conoceremos el precio de la tupla llamada 'Tornillos', y podremos chequear si el precio de la tupla actual es mayor.

Si queremos conocer todos los proveedores que no venden ningún artículo (por ejemplo, para poderlos eliminar de la base de datos), utilizaremos:

```
SELECT *
FROM PROVE P
WHERE NOT EXISTS
          (SELECT * FROM VENTAS V
           WHERE V.PNUM = P.PNUM);
```

En nuestro ejemplo, obtendremos un resultado vacío, porque cada proveedor vende al menos un artículo. Nótese que utilizamos P.PNUM de la SELECT externa en la clausula WHERE de la SELECT interna. Como hemos descrito antes, la subconsulta se evalúa para cada tupla de la consulta externa, es decir, el valor de P.PNUM se toma siempre de la tupla actual de la SELECT externa.

Unión, Intersección, Excepción

Estas operaciones calculan la unión, la intersección y la diferencia de la teoría de conjuntos de las tuplas derivadas de dos subconsultas.

Ejemplo 9. Union, Intersect, Except

La siguiente consulta es un ejemplo de UNION:

```
SELECT P.PNUM, P.PNOM, P.CIUD
FROM PROVE P
WHERE P.PNOM = 'Juan'
UNION
SELECT P.PNUM, P.PNOM, P.PCIUD
FROM PROVE P
WHERE P.PNOM = 'Ana';
```

Dará el resultado:

```
PNUM | PNOM  |  PCIUD
-----+-------+--------
  2  | Juan  | Paraná
  3  | Ana   | Santa Fe
```

Aquí tenemos un ejemplo para INTERSECT:

```
SELECT P.PNUM, P.PNOM, P.PCIUD
FROM PROVE P
WHERE P.PNUM > 1
```

```
   INTERSECT
   SELECT P.PNUM, P.PNOM, P.PCIUD
   FROM PROVE P
   WHERE P.PNUM > 2;
```

que dará como resultado:

```
                     PNUM| PNOM  |  PCIUD
                    -----+-------+--------
                      2  | Juan  | Paraná
         La única tupla devuelta por ambas partes de la consulta es
la única que tiene $PNUM=2$.
```

Finalmente, un ejemplo de EXCEPT:

```
   SELECT P.PNUM, P.PNOM, P.PCIUD
   FROM PROVE P
   WHERE P.PNUM > 1
   EXCEPT
   SELECT P.PNUM, P.PNOM, P.PCIUD
   FROM PROVE P
   WHERE P.PNUM > 3;
```

que dará como resultado:

```
                     PNUM| PNOM |  PCIUD
                    -----+------+---------
                      2  | Juan | Paraná
                      3  | Ana  | Santa Fe
```

Definición de Datos

El lenguaje SQL incluye un conjunto de comandos para definición de datos.

Create Table

El comando fundamental para definir datos es el que crea una nueva relación (una nueva tabla). La sintaxis del comando CREATE TABLE es:

```
   CREATE TABLE table_name
               (name_of_attr_1 type_of_attr_1
                [, name_of_attr_2 type_of_attr_2
                [, ...]]);
```

Ejemplo 10. Creación de una tabla

Para crear las tablas definidas en *La Base de Datos de Proveedores y Artículos* se utilizaron las siguientes instrucciones de SQL:

```
CREATE TABLE PROVE
             (PNUM  INTEGER,
              PNOM  VARCHAR(20),
              PCIUD VARCHAR(20));
CREATE TABLE ARTI
             (ANUM  INTEGER,
              ANOM  VARCHAR(20),
              APRE  DECIMAL(4 , 2));
CREATE TABLE VENTAS
             (PNUM INTEGER,
              ANUM INTEGER);
```

Tipos de Datos en SQL

A continuación sigue una lista de algunos tipos de datos soportados por SQL:

- INTEGER: entero binario con signo de palabra completa (31 bits de precisión).
- SMALLINT: entero binario con signo de media palabra (15 bits de precisión).
- DECIMAL (p[,q]): número decimal con signo de p dígitos de precisión, asumiendo q a la derecha para el punto decimal. ($15 \geq p \geq q$q ≥ 0). Si q se omite, se asume que vale 0.
- FLOAT: numérico con signo de doble palabra y coma flotante.
- CHAR(n): cadena de caracteres de longitud fija, de longitud n.
- VARCHAR(n): cadena de caracteres de longitud variable, de longitud máxima n.

Create Index

Se utilizan los índices para acelerar el acceso a una relación. Si una relación `R` tiene un índice en el atributo `A` podremos recuperar todas la tuplas t que tienen t(`A`) = a en un tiempo aproximadamente proporcional al número de tales tuplas t más que en un tiempo proporcional al tamaño de `R`.

Para crear un índice en SQL se utiliza el comando CREATE INDEX. La sintaxis es:

```
CREATE INDEX index_name
ON table_name ( name_of_attribute );
```

Ejemplo 11. Create Index

Para crear un índice llamado I sobre el atributo SNAME de la relación SUPPLIER, utilizaremos la siguiente instrucción:

```
CREATE INDEX I
ON PROVE (PNOM);
```

El índice creado se mantiene automáticamente. es decir, cada vez que una nueva tupla se inserte en la relación PROVE, se adaptará el índice I. Nótese que el único cambio que un usuario puede percibir cuando se crea un índice es un incremento en la velocidad.

Create View

Se puede ver una vista como una *tabla virtual*, es decir, una tabla que no existe *físicamente* en la base de datos, pero aparece al usuario como si existiese. Por contra, cuando hablamos de una *tabla base*, hay realmente un equivalente almacenado para cada fila en la tabla en algún sitio del almacenamiento físico.

Las vistas no tienen datos almacenados propios, distinguibles y físicamente almacenados. En su lugar, el sistema almacena la definición de la vista (es decir, las reglas para acceder a las tablas base físicamente almacenadas para materializar la vista) en algún lugar de los catálogos del sistema (vea *System Catalogs*). Para una discusión de las diferentes técnicas para implementar vistas, refiérase a *SIM98*.

En SQL se utiliza el comando **CREATE VIEW** para definir una vista. La sintaxis es:

```
CREATE VIEW view_name
AS select_stmt
```

donde `select_stmt` es una instrucción select válida, como se definió en *Select*. Nótese que `select_stmt` no se ejecuta cuando se crea la vista. Simplemente se almacena en los *catálogos del sistema* y se ejecuta cada vez que se realiza una consulta contra la vista.

Sea la siguiente definición de una vista (utilizamos de nuevo las tablas de *La Base de Datos de Proveedores y Artículos*):

```
CREATE VIEW Rosario_Proveedores
   AS SELECT P.PNOM, A.ANOM
      FROM PROVE P, ARTI A, VENTAS V
      WHERE P.PNUM = V.PNUM AND
            A.ANUM = V.ANUM AND
            P.PCIUD = 'Rosario';
```

Ahora podemos utilizar esta *relación virtual* Rosario_Proveedores como si se tratase de otra tabla base:

```
SELECT * FROM Rosario_Proveedores
WHERE A.ANOM = 'Tornillos';
```

Lo cual nos devolverá la siguiente tabla:

```
 PNOM  | ANOM
-------+----------
 María | Tornillos
```

Para calcular este resultado, el sistema de base de datos ha realizado previamente un acceso *oculto* a las tablas de la base PROVE, VENTAS y ARTI. Hace esto ejecutando la consulta dada en la definición de la vista contra aquellas tablas base. Tras eso, las cualificaciones adicionales (dadas en la consulta contra la vista) se podrán aplicar para obtener la tabla resultante.

Drop Table, Drop Index, Drop View

Se utiliza el comando DROP TABLE para eliminar una tabla (incluyendo todas las tuplas almacenadas en ella):

```
DROP TABLE table_name;
```

Para eliminar la tabla SUPPLIER, utilizaremos la instrucción:

```
DROP TABLE PROVE;
```

Se utiliza el comando DROP INDEX para eliminar un índice:

```
DROP INDEX index_name;
```

Finalmente, eliminaremos una vista dada utilizando el comando DROP VIEW:

```
DROP VIEW view_name;
```

Manipulación de Datos

Insert Into
Una vez que se crea una tabla (vea *Create Table*), puede ser llenada con tuplas mediante el comando

INSERT INTO. La sintaxis es:

```
INSERT INTO table_name (name_of_attr_1
                        [, name_of_attr_2 [,...]])
VALUES (val_attr_1
        [, val_attr_2 [, ...]]);
```

Para insertar la primera tupla en la relación SUPPLIER

```
INSERT INTO PROVE (PNUM, PNOM, PCIUD)
VALUES (1, 'María', 'Rosario');
```

Para insertar la primera tupla en la relación VENTAS, utilizamos:

```
INSERT INTO VENTAS (PNUM, ANUM)
VALUES (1, 1);
```

Update
Para cambiar uno o más valores de atributos de tuplas en una relación, se utiliza el comando UPDATE. La sintaxis es:

```
UPDATE table_name
SET name_of_attr_1 = value_1
    [, ... [, name_of_attr_k = value_k]]
WHERE condition;
```

Para cambiar el valor del atributo PRICE en el artículo 'Tornillos' de la relación PART, utilizamos:

```
UPDATE ARTI
SET APRE = 15
WHERE ANOM = 'Tornillos';
```

El nuevo valor del atributo APRE de la tupla cuyo nombre es 'Tornillos' es ahora 15.

Delete

Para borrar una tupla de una tabla particular, utilizamos el comando DELETE FROM. La sintaxis es:

```
DELETE FROM table_name
WHERE condition;
```

Para borrar el proveedor llamado 'Smith' de la tabla SUPPLIER, utilizamos la siguiente instrucción:

```
DELETE FROM PROVE
WHERE PNOM = 'María';
```

Consultas de Unión Internas

Consultas de Combinación entre tablas

Las vinculaciones entre tablas se realiza mediante la cláusula `INNER` que combina registros de dos tablas siempre que haya concordancia de valores en un campo común. Su sintaxis es:

```
SELECT campos FROM tb1 INNER JOIN tb2 ON tb1.campo1 comp tb2.campo2
```

En donde: **tb1, tb2** Son los nombres de las tablas desde las que se combinan los registros.
campo1, campo2 Son los nombres de los campos que se combinan. Si no son numéricos, los campos deben ser del mismo tipo de datos y contener el mismo tipo de datos, pero no tienen que tener el mismo nombre.
comp Es cualquier operador de comparación relacional : =, <, >, <=, >=, o <>.

Se puede utilizar una operación `INNER JOIN` en cualquier cláusula `FROM`. Esto crea una combinación por equivalencia, conocida también como unión interna. Las combinaciones Equi son las más comunes; éstas combinan los registros de dos tablas siempre que haya concordancia de valores en un campo común a ambas tablas. Se puede utilizar `INNER JOIN` con las tablas Departamentos y Empleados para seleccionar todos los empleados de cada departamento. Por el contrario, para seleccionar todos los departamentos (incluso si alguno de ellos no tiene ningún empleado asignado) se emplea `LEFT JOIN` o todos los empleados (incluso si alguno no está asignado a ningún departamento), en este caso `RIGHT JOIN`.

Si se intenta combinar campos que contengan datos Memo u Objeto OLE, se produce un error. Se pueden combinar dos campos numéricos cualesquiera, incluso si son de diferente tipo de datos. Por ejemplo, puede combinar un campo Numérico para el que la propiedad Size de su objeto Field está establecida como Entero, y un campo Contador.

El ejemplo siguiente muestra cómo podría combinar las tablas Categorías y Productos basándose en el campo IDCategoria:

```
SELECT Nombre_Categoría, NombreProducto
FROM Categorias INNER JOIN Productos
ON Categorias.IDCategoria = Productos.IDCategoria;
```

En el ejemplo anterior, IDCategoria es el campo combinado, pero no está incluido en la salida de la consulta ya que no está incluido en la instrucción `SELECT`. Para incluir el campo combinado, incluir el nombre del campo en la instrucción `SELECT`, en este caso, Categorias.IDCategoria.
También se pueden enlazar varias cláusulas `ON` en una instrucción `JOIN`, utilizando la sintaxis siguiente:

```
SELECT campos
FROM tabla1 INNER JOIN tabla2
ON tb1.campo1 comp tb2.campo1 AND
ON tb1.campo2 comp tb2.campo2) OR
ON tb1.campo3 comp tb2.campo3)];
```

También puede anidar instrucciones `JOIN` utilizando la siguiente sintaxis:

```
SELECT campos
FROM tb1 INNER JOIN
(tb2 INNER JOIN [( ]tb3
[INNER JOIN [( ]tablax [INNER JOIN ...)]
ON tb3.campo3 comp tbx.campox)]
ON tb2.campo2 comp tb3.campo3)
ON tb1.campo1 comp tb2.campo2;
```

Un LEFT JOIN o un RIGHT JOIN puede anidarse dentro de un INNER JOIN, pero un INNER JOIN no puede anidarse dentro de un LEFT JOIN o un RIGHT JOIN.
Por ejemplo:

```
SELECT DISTINCTROW Sum([Precio unidad] * [Cantidad]) AS [Ventas],
[Nombre] & " " & [Apellidos] AS [Nombre completo] FROM [Detalles de
pedidos],
Pedidos, Empleados, Pedidos INNER JOIN [Detalles de pedidos] ON Pedidos.
[ID de pedido] = [Detalles de pedidos].[ID de pedido], Empleados INNER
JOIN
Pedidos ON Empleados.[ID de empleado] = Pedidos.[ID de empleado] GROUP BY
[Nombre] & " " & [Apellidos];
```

Crea dos combinaciones equivalentes: una entre las tablas Detalles de pedidos y Pedidos, y la otra entre las tablas Pedidos y Empleados. Esto es necesario ya que la tabla Empleados no contiene datos de ventas y la tabla Detalles de pedidos no contiene datos de los empleados. La consulta produce una lista de empleados y sus ventas totales.

Si empleamos la cláusula INNER en la consulta se seleccionarán sólo aquellos registros de la tabla de la que hayamos escrito a la izquierda de INNER JOIN que contengan al menos un registro de la tabla que hayamos escrito a la derecha. Para solucionar esto tenemos dos cláusulas que sustituyen a la palabra clave INNER, estas cláusulas son LEFT y RIGHT. LEFT toma todos los registros de la tabla de la izquierda aunque no tengan ningún registro en la tabla de la izquierda. RIGHT realiza la misma operación pero al contrario, toma todos los registros de la tabla de la derecha aunque no tenga ningún registro en la tabla de la izquierda.

La sintaxis expuesta anteriormente pertenece a ACCESS, en donde todas las sentencias con la sintaxis funcionan correctamente. Los manuales de SQL-SERVER dicen que esta sintaxis es incorrecta y que hay que añadir la palabra reservada OUTER: LEFT OUTER JOIN y RIGHT OUTER JOIN. En la práctica funciona correctamente de una u otra forma.
No obstante, los INNER JOIN ORACLE no es capaz de interpretarlos, pero existe una sintaxis en formato ANSI para los INNER JOIN que funcionan en todos los sistemas. Tomando como referencia la siguiente sentencia:

```
SELECT Facturas.*, Albaranes.*
FROM Facturas INNER JOIN Albaranes ON Facturas.IdAlbaran =
Albaranes.IdAlbaran
WHERE Facturas.IdCliente = 325
```

La transformación de esta sentencia a formato ANSI sería la siguiente:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran AND  Facturas.IdCliente =
325
```

Como se puede observar los cambios realizados han sido los siguientes:
1. Todas las tablas que intervienen en la consulta se especifican en la cláusula FROM.
2. Las condiciones que vinculan a las tablas se especifican en la cláusula WHERE y se vinculan mediante el operador lógico AND.

Referente a los `OUTER JOIN`, no funcionan en ORACLE y además no conozco una sintaxis que funcione en los tres sistemas. La sintaxis en ORACLE es igual a la sentencia anterior pero añadiendo los caracteres `(+)` detrás del nombre de la tabla en la que deseamos aceptar valores nulos, esto equivale a un `LEFT JOIN`:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran = Albaranes.IdAlbaran (+) AND Facturas.IdCliente
= 325
```

Y esto a un `RIGHT JOIN`:

```
SELECT Facturas.*, Albaranes.* FROM Facturas, Albaranes
WHERE Facturas.IdAlbaran (+) = Albaranes.IdAlbaran AND Facturas.IdCliente
= 325
```

En SQL-SERVER se puede utilizar una sintaxis parecida, en este caso no se utiliza los caracteres `(+)` sino los caracteres `=*` para el `LEFT JOIN` y `*=` para el `RIGHT JOIN`.

- **Consultas de Autocombinación**

La autocombinación se utiliza para unir una tabla consigo misma, comparando valores de dos columnas con el mismo tipo de datos. La sintaxis en la siguiente:

```
SELECT alias1.columna, alias2.columna, ...
FROM tabla1 as alias1, tabla2 as alias2
WHERE alias1.columna = alias2.columna
AND otras condiciones
```

Por ejemplo, para visualizar el número, nombre y puesto de cada empleado, junto con el número, nombre y puesto del supervisor de cada uno de ellos se utilizaría la siguiente sentencia:

```
SELECT t.num_emp, t.nombre, t.puesto, t.num_sup,s.nombre, s.puesto
FROM empleados AS t, empleados AS s WHERE t.num_sup = s.num_emp
```

- **Consultas de Combinaciones no Comunes**

La mayoría de las combinaciones están basadas en la igualdad de valores de las columnas que son el criterio de la combinación. Las no comunes se basan en otros operadores de combinación, tales como `NOT`, `BETWEEN`, `<>`, etc.

Por ejemplo, para listar el grado salarial, nombre, salario y puesto de cada empleado ordenando el resultado por grado y salario habría que ejecutar la siguiente sentencia:

```
SELECT grados.grado,empleados.nombre, empleados.salario, empleados.puesto
FROM empleados, grados
WHERE empleados.salario
BETWEEN grados.salarioinferior AND grados.salariosuperior
ORDER BY grados.grado, empleados.salario
```

Para listar el salario medio dentro de cada grado salarial habría que lanzar esta otra sentencia:

```
SELECT grados.grado, AVG(empleados.salario) FROM empleados, grados
WHERE empleados.salario
BETWEEN grados.salarioinferior AND grados.salariosuperior
GROUP BY grados.grado
```