

Gestión de Bases de Datos

Tema 4: Lenguaje de Manipulación de Datos en ORACLE




IES Gonzalo Nazareno
CONSEJERÍA DE EDUCACIÓN

Raúl Ruiz Padilla
rruizp@gmail.com
Enero 2012

© Raúl Ruiz Padilla, Enero de 2012

Algunos derechos reservados.
Este artículo se distribuye bajo la licencia
"Reconocimiento-CompartirIgual 3.0 España" de Creative
Commons, disponible en
<http://creativecommons.org/licenses/by-sa/3.0/es/deed.es>

Este documento (o uno muy similar)
esta disponible en (o enlazado desde)
<http://informatica.gonzalonazareno.org>



Índice

1. Consultas sencillas.
2. Subconsultas.
3. Combinaciones de tablas.
4. Inserción de registros. Consultas de datos anexados.
5. Modificación de registros. Consultas de actualización.
6. Borrado de registros. Consultas de eliminación.
7. cláusulas avanzadas de selección.
 - 7.1. Group by y having
 - 7.2. Outer joins. Combinaciones externas.
 - 7.3. Consultas con operadores de conjuntos.
 - 7.4. Subconsultas correlacionadas.
8. Control de transacciones en SQL.



1. Consultas sencillas. Sintaxis.

Para realizar consultas sobre los datos existentes se emplea la sentencia SELECT, cuyo formato básico es el siguiente:

```
SELECT [ALL | DISTINCT] [column1, ..column | *]  
FROM tabla1, .., tablan  
[WHERE condicion]  
[ORDER BY col1 [DESC | ASC], col2 [DESC | ASC],...]  
;
```



1. Consultas sencillas. ALL y DISTINCT

- ALL, recupera todas las filas, opción por defecto.
- DISTINCT, recupera las filas distintas.

```
SELECT DISTINCT deptno  
FROM emp;
```

```
SELECT DISTINCT sal  
FROM emp;
```



1. Consultas sencillas. FROM

- FROM, obligatoria, se especifican las tablas donde está la información necesaria.

```
SELECT *
```

```
FROM emp;
```

- Si el usuario no es propietario de las tablas:

```
SELECT *
```

```
FROM usuario.emp;
```

- Se puede poner alias a las tablas

```
SELECT *
```

```
FROM emp e;
```



1. Consultas sencillas. WHERE

- WHERE condición

Condición tendrá la forma: expresión **operador** expresión

- Operadores

IN, NOT IN, BETWEEN, NOT BETWEEN, LIKE

- Condiciones múltiples

AND, OR, NOT y ()

- Ejemplos:

WHERE SAL > 1000;

WHERE (SAL > 1000) AND (COMM IS NOT NULL);

WHERE (comm IS NULL) AND (UPPER(ENAME) = 'MARY');



1. Consultas sencillas. Proyección

- La proyección es la operación por la que se seleccionan determinadas columnas de las tablas consultadas.

```
SELECT [ALL|DISTINCT] colum1, ..column  
FROM tabla1,...tablan;
```

Todas las columnas:

```
SELECT empno, ename, ...  
FROM EMP;
```

O bien:

```
SELECT *  
FROM emp;
```

Algunas columnas:

```
SELECT deptno, dname  
FROM dept;
```



1. Consultas sencillas. Selección

- La selección es la operación por la que se seleccionan determinadas filas de las tablas consultadas. Se realiza mediante la cláusula WHERE condición.
- Ejemplos:

```
SELECT ename, job, mgr  
FROM emp  
WHERE deptno = 10;
```

```
SELECT *  
FROM emp  
WHERE job = 'CLERK' AND deptno = 10;
```




1. Consultas sencillas. Alias de columnas y tablas.

Si el nombre de la columna resulta demasiado largo, corto o es poco significativo, se puede hacer uso de los alias de columnas.

También pueden usarse los alias de tablas para no tener que usar el nombre completo de la tabla en el resto de la SELECT si tenemos que especificar a qué tabla pertenece una columna.

Ejemplo:

```
SELECT ename "nombre empleado", dname "nombre departamento"  
FROM emp e, dept d  
WHERE e.deptno = d.deptno;
```



1. Consultas sencillas. Ordenación.

- La ordenación de la consulta se realiza con la cláusula ORDER BY columna [ASC|DESC].
- Por defecto se ordena de forma ascendente (ASC).

```
SELECT ename  
FROM emp  
ORDER BY sal * 12;
```

- Para tener 2 o más criterios de ordenación, el principal es el situado más a la izquierda y en caso de igualdad se van aplicando los siguientes:

```
SELECT ename, job, sal * 12 AS salario_anual  
FROM emp  
ORDER BY job ASC, sal * 12 DESC;
```



1. Consultas sencillas. Vistas.

- Se puede poner un nombre a una consulta, a esto se le llama crear una vista. La sintaxis es la siguiente:

```
CREATE VIEW nombrevista AS  
SELECT...
```

- Posteriormente se puede hacer una consulta usando la vista en la cláusula FROM en lugar de una tabla.

```
SELECT *  
FROM nombre_vista  
WHERE ...
```

Ejemplo: CREATE VIEW EMP30 AS
SELECT *
FROM EMP
WHERE DEPTNO=30;

Y:
SELECT ENAME, JOB
FROM EMP30;



2. Subconsultas. Definición y sintaxis.

- Definición: Son consultas que se usan dentro de la cláusula WHERE de otra consulta.

Sintaxis:

SELECT

FROM

WHERE columna operador (SELECT
FROM....
WHERE...);



2. Subconsultas. Ejemplo.

Consulta: Obtén los nombres de los empleados con el mismo oficio que Gil.

Primero averiguamos el oficio de Gil, así:

```
SELECT job  
FROM emp  
WHERE ename = 'Gil';
```

Y después mostramos los empleados cuyo oficio es el que hemos obtenido anteriormente:


```
SELECT ename  
FROM emp  
WHERE job = (SELECT job  
              FROM emp  
              WHERE ename = 'Gil');
```



2. Subconsultas. Ejemplo.

Consulta los nombres y oficios de los empleados del departamento 20 cuyo oficio sea igual al de cualquiera de los empleados del departamento SALES.

```
SELECT ename, job
FROM emp
WHERE deptno = 20
AND job IN (SELECT job
            FROM emp
            WHERE deptno = (SELECT deptno
                           FROM dept
                           WHERE dname = 'SALES'));
```



2. Subconsultas. Condiciones de Búsqueda. (I)

- Las subconsultas aparecen como parte de una condición de búsqueda de una cláusula WHERE o HAVING.
- Las condiciones de búsqueda son:
 - Test de comparación en subconsultas(<, >..)
 - Test de pertenencia a un conjunto de valores (IN).
 - Test de existencia (EXISTS, NO EXISTS).
 - Test de comparación cuantificada (ANY, ALL).



2. Subconsultas. Condiciones de Búsqueda. (II)

Test de comparación en subconsultas.

Operadores utilizables (<, >, =, >=, <=, <>, !=).

Compara el valor de una expresión con un valor único producido por una subconsulta.

Ejemplo:

```
SELECT ename  
FROM emp  
WHERE job = (SELECT job  
              FROM emp  
              WHERE ename = 'JAMES');
```



2. Subconsultas. Condiciones de Búsqueda. (III)

Test de pertenencia a un conjunto de valores (IN).

Comprueba si el valor de una expresión es uno de los valores producidos por una subconsulta.

Ejemplo:

```
SELECT ename  
FROM emp  
WHERE job IN (SELECT job  
              FROM emp  
              WHERE deptno = 20);
```



2. Subconsultas. Condiciones de Búsqueda. (IV)

Test de existencia (EXISTS, NO EXISTS).

Si la subconsulta contiene filas, el test adopta el valor verdadero, si la subconsulta no contiene ninguna fila, el test toma el valor falso.

Se puede sustituir por otras combinaciones de operadores.

Ejemplo:

```
SELECT dname
FROM dept
WHERE EXISTS (SELECT *
              FROM emp
              WHERE emp.deptno = dept.deptno);
```



2. Subconsultas. Condiciones de Búsqueda. (IV)

Test de comparación cuantificada (ANY, ALL).

Se usan junto a operadores relacionales: <, >..

ANY y ALL se pueden sustituir por otras combinaciones de operadores y funciones.

ANY: compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta, si alguna de las comparaciones devuelve TRUE, ANY devuelve TRUE, si la subconsulta no devuelve nada, devolverá FALSE.

ALL: compara el valor de una expresión con cada uno del conjunto de valores producido por una subconsulta, si todas las comparaciones devuelven TRUE, ALL devuelve TRUE, en caso contrario, devolverá FALSE.

2. Subconsultas. Condiciones de Búsqueda. (IV)

Ejemplos de test de comparación cuantificada .

- Ejemplo ANY:

```
SELECT *  
FROM emp  
WHERE sal = ANY (SELECT sal  
                  FROM emp  
                  WHERE deptno = 30);
```

- Ejemplo ALL:

```
SELECT *  
FROM emp  
WHERE sal < ALL (SELECT sal  
                  FROM emp  
                  WHERE deptno = 30);
```



2. Subconsultas.

Subconsultas que generan valores simples

En algunos casos, podemos estar seguros de que una subconsulta va a devolver SIEMPRE una sola fila. En ese caso podremos usar un operador relacional.

Si usamos `=` y la subconsulta devuelve más de un valor se origina un error.

```
SELECT ename  
FROM emp  
WHERE job = (SELECT job  
              FROM emp  
              WHERE empno = 7082);
```

Actividad: Averigua cuándo podemos usar un operador relacional sabiendo que no fallará nunca.



2. Subconsultas.

Subconsultas que generan conjuntos de valores

La mayoría de las subconsultas pueden devolver más de una fila. En estos casos debo utilizar el operador de pertenencia a un conjunto de valores (IN).

Ejemplo:

```
SELECT ename  
FROM emp  
WHERE job IN (SELECT job  
              FROM emp  
              WHERE deptno = 20);
```



2. Subconsultas. Ejemplos.

Visualizar los datos de los empleados que trabajan en DALLAS o CHICAGO:

```
SELECT *  
FROM emp  
WHERE deptno IN (SELECT deptno  
                  FROM dept  
                  WHERE loc IN ('DALLAS', 'CHICAGO'));
```

Muestra los nombres de los empleados con el mismo oficio y salario que JAMES:

```
SELECT ename, sal  
FROM emp  
WHERE (job, sal) = (SELECT job, sal  
                    FROM emp  
                    WHERE ename = 'JAMES');
```



3. Combinacion de tablas.

Cuando los datos que queremos mostrar se encuentran en varias tablas diferentes, es necesario colocarlas todas en el FROM.

Cuando se hace esto, el gestor hace el producto cartesiano de todas las tablas, esto es, combina cada fila de una de ellas con todas las filas de las otras. Esto genera filas sin sentido que es necesario filtrar con la llamada “condición de join”.

A esto se le llama combinación de tablas.

Formato:

SELECT columnas de las tablas

FROM tabla1, tabla2...

WHERE tabla1.columna1 = tabla2.columna2; // condición de join

3. Combinacion de tablas. Ejemplo funcionamiento.

Veamos un ejemplo del funcionamiento de la combinación:

EMPNO	ENAME	DEPTNO
123	PEPE	10
456	ANA	10
789	EVA	20

DEPTNO	DNAME	LOC
10	VENTAS	CORIA
20	I+D	LORA
30	PERSONAL	PILAS

Si no se especifica la condición de join, al nombre de su departamento, debemos obtener información de ambas tablas. Si hacemos “SELECT ENAME, DNAME FROM EMP, DEPT” olvidando la condición de join el resultado será el producto cartesiano de ambas tablas:

ENAME	DNAME
PEPE	VENTAS
PEPE	I+D
PEPE	PERSONAL
ANA	VENTAS
ANA	I+D
ANA	PERSONAL
EVA	VENTAS
EVA	I+D
EVA	PERSONAL



3. Combinacion de tablas. Reglas a seguir.

- Se pueden unir tantas tablas como deseemos.
- El número de condiciones de join a incluir en la sentencia será el número de tablas menos uno.
- En la proyección podemos usar columnas de todas las tablas incluidas.
- Si hay columnas con el mismo nombre en varias tablas, se deben especificar para evitar ambigüedades con la sintaxis:

Nombre_tabla.nombre_columna.

- Se debe intentar evitar realizar combinaciones de tablas muy grandes por el gran consumo de recursos del servidor que conllevan.



3. Combinacion de tablas. Ejemplos.

- 1.Consultar nombre de empleados, salario, nombre del departamento al que pertenecen y localidad de este.
- 2.Muestra los nombres de los empleados con un salario entre 500 y 1000, cuyo oficio sea MANAGER.
- 3.Muestra los nombres de los departamentos que no tengan empleados.
- 4.Muestra los nombres de los departamentos que tengan algún empleado.



4. Inserción de registros.


Si queremos añadir datos individuales a una tabla debemos emplear la orden INSERT...VALUES.

```
INSERT INTO nombre_tabla [(column1 [, (...)]]  
VALUES (valor1 [, valor2]...);
```

Si no se especifican columnas, se entiende que serán todas las columnas de la tabla.

Los valores que se darán a cada columna deben tener el mismo tipo de dato con que se definió la columna.

Si una columna no está en la lista recibirá NULL. Si al crear la tabla la habíamos definido como NOT NULL, la orden INSERT fallará.



4. Inserción de registros. Ejemplos.

```
INSERT INTO profesores (apellidos, especialidad, cod_centro)  
VALUES ('Quiroga Martín, Ma Isabel', 'INFORMATICA', 45) ;
```

```
INSERT INTO profesores (apellidos, especialidad, cod_centro)  
VALUES('Seco Jiménez, Ernesto', 'LENGUA');  
Error..
```

```
INSERT INTO profesores  
VALUES(22, 23444800, 'Gonzalez Sevilla, Miguel A.', 'HISTORIA');
```



4. Inserción de registros. Consultas de datos anexados.

En ocasiones, interesa introducir en una tabla un conjunto de datos provenientes de otra fuente. En ese caso, es necesario realizar una consulta de datos anexados. El formato es el siguiente:

```
INSERT INTO nombre_tabla [(column1 [, (...)])]  
SELECT...
```

Se insertarán en la tabla tantos datos como filas devuelva la consulta. Veamos un ejemplo:

```
INSERT INTO emp_bcn  
SELECT *  
FROM emp  
WHERE deptno IN (SELECT deptno  
                  FROM dept  
                  WHERE loc = 'BARCELONA');
```



4. Inserción de registros. Consultas de datos anexados.

También es posible insertar un registro individual combinando información que ya conocemos con otra que hay que extraer de la base de datos. Veamos un ejemplo de esto:

Insertar un empleado con código 1111 y nombre GARCIA en el departamento en el que trabaja PEPE con el mismo sueldo de éste. La fecha de ingreso será la del sistema.

```
INSERT INTO emp (empno, ename, hiredate, sal, deptno)
SELECT 1111, 'GARCIA', sysdate, sal, deptno
FROM emp
WHERE ename = 'PEPE';
```



4. Inserción de registros. Consultas de datos anexados.

También es posible insertar un registro individual combinando información que ya conocemos con otra que hay que extraer de la base de datos. Veamos un ejemplo de esto:

Insertar un empleado con código 1111 y nombre GARCIA en el departamento en el que trabaja PEPE con el mismo sueldo de éste. La fecha de ingreso será la del sistema.

```
INSERT INTO emp (empno, ename, hiredate, sal, deptno)
SELECT 1111, 'GARCIA', sysdate, sal, deptno
FROM emp
WHERE ename = 'PEPE';
```

Actividades:

- Dadas las tablas, ALUM y NUEVOS, inserta en ALUM los nuevos alumnos.
- Inserta en EMP, el empleado 2000, SAAVEDRA, fecha de alta la de sistema, el salario el 20% mas que MILLER y el resto de los datos los de este.

5. Modificación de datos.

Para modificar el valor de los datos existentes en la tabla se emplea la orden UPDATE. La sintaxis es la siguiente:

```
UPDATE nombre_tabla  
SET column1 = valor1, ..column = valorn  
WHERE condicion;
```

Si se omite la cláusula WHERE se actualizarán todas las filas.

Ejemplo:

```
UPDATE emp  
SET sal = sal + 100, comm = NVL(comm, 0) + 10  
WHERE empno = 7369;
```



5. Modificación de datos. Consultas de actualización.

En una orden UPDATE podemos usar una consulta para obtener el valor que se va a poner en una columna.

En este caso, debe devolver una fila y el mismo nº de columnas y tipo de datos que las que hay entre paréntesis en el SET:

```
UPDATE emp  
SET sal = (SELECT max(sal) FROM emp)  
WHERE empno = 7082;
```

O bien podemos usarla para determinar qué filas se van a actualizar:

```
UPDATE emp  
SET sal = 1000  
WHERE deptno = (SELECT deptno FROM dept WHERE loc =  
'LORA');
```




5. Modificación de datos. Consultas de actualización.

También pueden combinarse ambos formatos como en el siguiente ejemplo:

Haz que los empleados del departamento 'ACCOUNTING' tengan el nombre en minúsculas y ganen el doble que 'JAMES':

```
UPDATE emp
SET ename = LOWER(ename),
    sal =      (SELECT sal * 2
                FROM emp
                WHERE ename = 'JAMES')
WHERE deptno = (SELECT deptno
                FROM dept
                WHERE dname = 'ACCOUNTING');
```



6. Borrado de datos. Consultas de eliminación.

Si deseamos eliminar uno o varios registros de una tabla, debemos emplear la orden DELETE. Su sintaxis es la siguiente:

```
DELETE [FROM] nombre_tabla  
[WHERE condicion];
```

Se borran los registros que cumplan la condición, en la que puede incluirse una sentencia SELECT si es necesario. A esto se le llama consulta de eliminación.

Si se omite la cláusula WHERE se borrarán todos los registros de la tabla.

Ejemplo:

```
DELETE EMP  
WHERE DEPTNO = 20;
```



7. Cláusulas avanzadas de selección.

7.1. GROUP BY y HAVING.

Cuando necesitamos obtener alguna información acerca de varios conjuntos de registros de la tabla debemos usar la cláusula GROUP BY, que permite hallar subtotales.

Por ejemplo, cuando necesitemos saber el salario medio por departamento será necesario realizar un agrupamiento por este campo, así:

```
SELECT deptno, AVG(sal)
FROM emp
GROUP BY deptno;
```



7. Cláusulas avanzadas de selección.

7.1. GROUP BY y HAVING.

El funcionamiento del GROUP BY y el HAVING es el siguiente:

GROUP BY: Se emplea para calcular propiedades de 1 o más conjuntos de filas, las filas resultantes de la agrupación se almacenan en una tabla temporal.

HAVING: condición de selección de los grupos de filas. Solo se muestran aquellos conjuntos de filas que cumplen con la condición especificada en la cláusula HAVING.

Se evalúa sobre la tabla temporal, no puede existir sin realizar previamente un GROUP BY.



7. Cláusulas avanzadas de selección.

7.1. GROUP BY y HAVING. Ejemplos.

Muestra el número de empleados de cada departamento:

```
SELECT deptno, COUNT(*)  
FROM emp  
GROUP BY deptno;
```

Muestra los departamentos con más de cuatro empleados:

```
SELECT deptno, COUNT(*)  
FROM emp  
GROUP BY deptno  
HAVING COUNT(*) > 4;
```

Muestra los departamentos en los que el salario medio es mayor que el salario medio de la empresa:

```
SELECT deptno, AVG(sal)  
FROM emp  
GROUP BY deptno  
HAVING AVG(sal) >= (SELECT AVG(sal)  
                     FROM emp);
```



7. Cláusulas avanzadas de selección.

7.1. GROUP BY y HAVING.

Es importante que recordéis que HAVING es como el WHERE de los agrupamientos.

Es frecuente que a veces no sepáis si colocar las condiciones en el WHERE o en el HAVING. La solución es muy sencilla:

Si la condición es sobre un atributo de un registro individual va en el WHERE. Si la condición es sobre un atributo del grupo de registros va en el HAVING.

Orden de evaluación de las Cláusulas de la orden SELECT:

1. WHERE (selecciona filas individuales)
2. GROUP BY (agrupa filas)
3. HAVING (selecciona grupos de filas)
4. ORDER BY (ordena los grupos de filas)



7. Cláusulas avanzadas de selección.

7.1. GROUP BY y HAVING. Más ejemplos.

Muestra la suma de los salarios de cada departamento junto con el salario máximo y mínimo de los mismos:

```
SELECT deptno,  
       TO_CHAR(SUM(sal), '99G999D99') AS suma,  
       TO_CHAR(MAX(sal), '99G999D99') AS Máximo,  
       TO_CHAR(MIN(sal), '99G999D99') AS Mínimo  
FROM emp  
GROUP BY deptno;
```

Muestra en orden de mayor a menor cuantos empleados hay de cada oficio en cada departamento:

```
SELECT deptno, job, COUNT(*)  
FROM emp  
GROUP BY deptno, job  
ORDER BY deptno;
```



7. Cláusulas avanzadas de selección.

7.1. GROUP BY y HAVING. Más ejemplos.

Muestra los nombres de los departamentos que tienen más de cuatro analistas:

```
SELECT dname, COUNT(empno)
FROM emp e, dept d
WHERE e.deptno = d.deptno AND job='ANALYST'
GROUP BY dname
HAVING COUNT(empno) > 4;
```

Muestra los nombres de los departamentos con mayor número de empleados:

```
SELECT d.deptno, dname, COUNT(empno)
FROM emp e, dept d
WHERE e.deptno = d.deptno
GROUP BY d.deptno, dname
HAVING COUNT(empno) = (SELECT MAX(COUNT(*))
                        FROM emp
                        GROUP BY deptno);
```



7. Cláusulas avanzadas de selección.

7.2 Combinaciones externas.

Permite combinar tablas y seleccionar filas de estas aunque no tengan correspondencia entre ellas.

Se usan cuando deseo mostrar información de todos los registros de una de las tablas tengan o no correspondencia en la otra tabla.

La sintaxis es la siguiente:

```
SELECT tabla1.col1, ..tabla1.coln,tabla2.col1,...tabla2.coln  
FROM tabla1, tabla2  
WHERE tabla1.colum1 = tabla2.colum1(+);
```



7. Cláusulas avanzadas de selección.

7.2 Combinaciones externas. Ejemplo.

Probemos la siguiente sentencia:

```
SELECT d.deptno, d.dname, COUNT(empno)
FROM emp e, dept d
WHERE e.deptno = d.deptno
GROUP BY d.deptno, d.dname;
```

¿Qué pasa con el departamento 40?

Prueba ahora la siguiente sentencia:

```
SELECT d.deptno, d.dname, COUNT(empno)
FROM emp e, dept d
WHERE e.deptno(+) = d.deptno
GROUP BY d.deptno, d.dname;
```

Con **OUTER JOIN**, se muestran todos los departamentos aunque no tengan empleados, nº empleados estará a NULL.



7. Cláusulas avanzadas de selección.

7.2 Combinaciones externas. Ejemplo.

Actividad: Analiza que ocurre si COUNT(*) en vez de COUNT(empno) y si en la proyección de la SELECT ponemos e.deptno en vez de d.deptno.

```
SELECT e.deptno, d.dname, COUNT(*)  
FROM emp e, dept d  
WHERE e.deptno(+) = d.deptno  
GROUP BY d.deptno, d.dname;
```

Intenta comprender porqué pasa eso, sabiendo que la combinación externa añade una fila nueva al producto cartesiano dejando en blanco los datos que no existen.



7. Cláusulas avanzadas de selección.

7.3. Operadores de conjuntos: Unión.

UNION: Une los resultados de 2 consultas, las filas duplicadas se reducen a una.

```
SELECT column1, column2,..columnn  
FROM tabla1  
WHERE condicion  
UNION  
SELECT column1, column2,..columnn  
FROM tabla2  
WHERE condicion;
```

Ejemplo:

```
SELECT nombre FROM alum  
UNION  
SELECT nombre FROM nuevos;
```

UNION ALL: Igual pero las filas duplicadas no se reducen a una.

7. Cláusulas avanzadas de selección.

7.3. Operadores de conjuntos: Intersección.

INTERSECTION: Muestra las filas que son idénticas en ambas consultas.

```
SELECT column1, column2,...column  
FROM tabla1  
WHERE condicion  
INTERSECT  
SELECT column1, column2,...column  
FROM tabla2  
WHERE condicion;
```

Ejemplo:

```
SELECT nombre FROM alum  
INTERSECT  
SELECT nombre FROM antiguos;
```

Es equivalente a:

```
SELECT nombre FROM alum  
WHERE nombre IN  
(SELECT nombre FROM antiguos);
```



7. Cláusulas avanzadas de selección.

7.3. Operadores de conjuntos: Diferencia.

MINUS: Muestra las filas que aparecen en la 1ª consulta y no en la 2ª.

```
SELECT colum1, colum2,..column  
FROM tabla1  
WHERE condicion  
MINUS  
SELECT colum1, colum2,..column  
FROM tabla2  
WHERE condicion;
```

Ejemplo:

```
SELECT nombre, localidad FROM alum  
MINUS  
SELECT nombre, localidad FROM antiguos;
```

Es equivalente a:

```
SELECT nombre, localidad FROM alum  
WHERE nombre NOT IN  
(SELECT nombre FROM antiguos);
```



7. Cláusulas avanzadas de selección.

7.3. Operadores de conjuntos: Reglas.

- Las columnas de ambas tablas se asocian de izquierda a derecha.
- Las SELECTs tienen que tener el mismo nº de columnas.
- Los nombres de las columnas de ambas SELECTs no tienen que coincidir.
- Los tipos de datos tienen que coincidir, aunque la longitud no tiene que ser la misma.



7. Cláusulas avanzadas de selección.


7.4. Subconsultas correlacionadas.

En ocasiones, es necesario relacionar campos de la consulta principal con campos de la subconsulta.

Veamos este ejemplo:

Mostrar los datos de los empleados cuyos salarios sean iguales al máximo salario de **su** departamento.

Haríamos una subconsulta para ver el máximo salario del departamento del empleado y después lo compararíamos con el salario del empleado.




7. Cláusulas avanzadas de selección.

7.4. Subconsultas correlacionadas.

La consulta debe quedar así:

```
SELECT *  
FROM emp e  
WHERE sal = (SELECT MAX(sal)  
              FROM emp  
              WHERE deptno = e.deptno);
```

Hacemos referencia desde la subconsulta a una columna o varias de la consulta más externa. A veces el nombre de las columnas coincide, por lo tanto usaremos **alias para la tabla más externa**.



8. Control de transacciones en SQL.

Transacción: Secuencia de una o más ordenes SQL, que juntas forman una unidad de trabajo.

ROLLBACK, aborta la transacción, volviendo las tablas a la situación del último COMMIT.

ROLLBACK automático, se produce cuando haya algún fallo del sistema y no hayamos validado el trabajo hasta entonces.



8. Control de transacciones en SQL.

COMMIT, para validar las operaciones DML que hayamos realizado en la BD.

AUTOCOMMIT, parámetro que en SQL*Plus e iSQL*Plus, sirve para validar automáticamente las transacciones en la BD, siempre que esté en ON.

Por defecto está en OFF, por tanto INSERT, UPDATE y DELETE no se ejecutan automáticamente, hasta que no hagamos COMMIT

Para saber si está activado:

SHOW AUTOCOMMIT

Para activarlo:

SET AUTOCOMMIT ON



8. Control de transacciones en SQL.

Hay algunas órdenes SQL que fuerzan COMMIT implícito:

QUIT

EXIT

CONNECT

DISCONNECT

CREATE TABLE

DROP TABLE

CREATE VIEW

DROP VIEW

GRANT

ALTER

REVOKE

AUDIT

NOAUDIT

