

4. CONSULTA DE BASES DE DATOS. LENGUAJE DE MANIPULACIÓN DE DATOS

4.1. INTRODUCCIÓN

A lo largo de esta unidad nos centraremos en la cláusula **SELECT**. Sin duda es el comando más versátil del lenguaje SQL. El comando SELECT permite:

- Obtener datos de ciertas columnas de una tabla (proyección).
- Obtener registros (filas) de una tabla de acuerdo con ciertos criterios (selección).
- Mezclar datos de tablas diferentes (asociación, join).

4.2. CONSULTAS

Para realizar consultas a una base de datos relacional hacemos uso de la sentencia SELECT. La sintaxis básica del comando SELECT es la siguiente:

```
SELECT * | {[ DISTINCT ] columna | expresión [[AS] alias], ...}  
FROM nombre_tabla;
```

Donde:

*. El asterisco significa que se seleccionan todas las columnas.

DISTINCT. Hace que no se muestren los valores duplicados.

columna. Es el nombre de una columna de la tabla que se desea mostrar.

expresión. Una expresión válida SQL.

alias. Es un nombre que se le da a la cabecera de la columna en el resultado de esta instrucción.

Ejemplos:

```
-- Selección de todos los registros de la tabla CLIENTES  
SELECT * FROM CLIENTES;
```

```
-- Selección de algunos campos de la tabla CLIENTES  
SELECT nombre, apellido1, apellido2 FROM CLIENTES;
```

4.3. CÁLCULOS

4.3.1. Cálculos Aritméticos

Los operadores + (suma), - (resta), * (multiplicación) y / (división), se pueden utilizar para hacer cálculos en las consultas. Cuando se utilizan como expresión en una consulta SELECT, **no modifican los datos originales**.

Ejemplo:

-- Consulta con 3 columnas

```
SELECT nombre, precio, precio*1.16 FROM ARTICULOS;
```

-- Ponemos un alias a la tercera columna.

-- Las comillas dobles en el alias hacen que se respeten mayúsculas y minúsculas, de otro modo siempre aparece en mayúsculas

```
SELECT nombre, precio, precio*1.16 AS "Precio + IVA" FROM ARTICULOS;
```

La prioridad de esos operadores es: tienen más prioridad la multiplicación y división, después la suma y la resta. En caso de igualdad de prioridad, se realiza primero la operación que esté más a la izquierda. Como es lógico se puede evitar cumplir esa prioridad usando paréntesis; el interior de los paréntesis es lo que se ejecuta primero. **Cuando una expresión aritmética se calcula sobre valores NULL, el resultado de la expresión es siempre NULL.**

4.3.2. Concatenación

El operador || es el de la concatenación. Sirve para unir textos.

Ejemplo:

```
SELECT tipo, modelo, tipo || '-' || modelo "Clave Pieza" FROM PIEZAS;
```

El resultado de esa consulta tendría esta estructura:

TIPO	MODELO	Clave Pieza
AR	6	AR-6
AR	7	AR-7
AR	8	AR-8
AR	9	AR-9
AR	12	AR-12
AR	15	AR-15
AR	20	AR-20
AR	21	AR-21
BI	10	BI-10
BI	20	BI-20
BI	22	BI-22
BI	24	BI-24

4.3.3. Condiciones - WHERE

Se puede realizar consultas que restrinjan los datos de salida de las tablas. Para ello se utiliza la cláusula **WHERE**. Esta cláusula permite colocar una condición que han de cumplir todos los registros que queremos que se muestren. Las filas que no la cumplan no aparecerán en la ejecución de la consulta.

Nota

Con el comando SELECT indicamos las columnas que queremos que aparezcan en nuestra consulta. Con el comando WHERE indicamos las filas que queremos que aparezcan en nuestra consulta (serán las que cumplan las condiciones que especifiquemos detrás del WHERE).

Ejemplo:

```
-- Tipo y modelo de las piezas cuyo precio es mayor que 3
SELECT tipo, modelo
FROM PIEZAS
WHERE precio > 3;
```

4.3.3.1. Operadores de comparación

Los operadores de comparación que se pueden utilizar en la cláusula WHERE son:

Operador	Significado
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
=	Igual
<>	Distinto
!=	Distinto

Se pueden utilizar tanto para comparar números como para comparar textos y fechas. En el caso de los textos, las comparaciones se hacen en orden alfabético. Sólo que es un orden alfabético estricto. Es decir, el orden de los caracteres en la tabla de códigos. Así la letra Ñ y las vocales acentuadas nunca quedan bien ordenadas ya que figuran con códigos más altos. Las mayúsculas figuran antes que las minúsculas (la letra “Z” es menor que la “a”).

4.3.3.2. Operadores lógicos

Son:

Operador	Significado
AND	Devuelve verdadero si las expresiones a su izquierda y derecha son ambas verdaderas
OR	Devuelve verdadero si cualquiera de las dos expresiones a izquierda y derecha del OR, son verdaderas
NOT	Invierte la lógica de la expresión que está a su derecha. Si era verdadera, mediante NOT pasa a ser falso.

Ejemplos:

-- Personas entre 25 y 50 años

```
SELECT nombre, apellidos  
FROM PERSONAS  
WHERE edad >= 25 AND edad <= 50;
```

```
-- Personas de más de 60 y menos de 20  
SELECT nombre, apellidos  
FROM PERSONAS  
WHERE edad > 60 OR edad < 20;
```

4.3.3.3. BETWEEN

El operador BETWEEN nos permite obtener datos que se encuentren entre dos valores determinados (incluyendo los dos extremos).

Ejemplo:

```
-- Selección de las piezas cuyo precio está entre 3 y 8  
-- (ambos valores incluidos)  
SELECT tipo, modelo, precio  
FROM PIEZAS  
WHERE precio BETWEEN 3 AND 8;
```

El operador **NOT BETWEEN** nos permite obtener los los valores que son menores (estrictamente) que el más pequeño y mayores (estrictamente) que el más grande. Es decir, no incluye los extremos.

Ejemplo:

```
-- Selección de las piezas cuyo precio sea menor que 3 o mayor que 8  
-- (los de precio 3 y precio 8 no estarán incluidos)  
SELECT tipo, modelo, precio  
FROM PIEZAS  
WHERE precio NOT BETWEEN 3 AND 8;
```

4.3.3.5. LIKE

El operador LIKE se usa sobre todo con textos, permite obtener registros cuyo valor en un campo cumpla una condición textual. LIKE utiliza una cadena que puede contener estos símbolos:

Símbolo	Significado
%	Una serie cualquiera de caracteres
_	Un carácter cualquiera

Ejemplos:

-- Selección el nombre de las personas que empiezan por A

```
SELECT nombre
FROM PERSONAS
WHERE nombre LIKE 'A%';
```

-- Selección el nombre y los apellidos de las personas

cuyo primer apellido sea Jiménez, Giménez, Ximénez

```
SELECT nombre, apellido1, apellido2
FROM PERSONAS
WHERE apellido1 LIKE '_iménez';
```

Si queremos que en la cadena de caracteres se busquen los caracteres "%" o
"_" le anteponemos el símbolo escape:

Ejemplo:

-- Seleccionamos el tipo, el modelo y el precio de las piezas

-- cuyo porcentaje de descuento sea 3%

```
SELECT tipo, modelo, precio
FROM PIEZAS
WHERE descuento LIKE '3\%' ESCAPE '\';
```

4.3.3.6. IS NULL

La cláusula IS NULL devuelve “verdadero” si una expresión contiene un nulo, y “Falso” en caso contrario. La cláusula IS NOT NULL devuelve “verdadero” si una expresión NO contiene un nulo, y “Falso” en caso contrario.

Ejemplos:

-- Devuelve el nombre y los apellidos de las personas que NO tienen teléfono

```
SELECT nombre, apellido1, apellido2
```

```
FROM PERSONAS  
WHERE telefono IS NULL;
```

```
-- Devuelve el nombre y los apellidos de las personas que Sí tienen teléfono  
SELECT nombre, apellido1, apellido2  
FROM PERSONAS  
WHERE telefono IS NOT NULL;
```

4.3.3.7. Precedencia de operadores

A veces las expresiones que se producen en los SELECT son muy extensas y es difícil saber que parte de la expresión se evalúa primero, por ello se indica la siguiente tabla de precedencia:

Orden de precedencia	Operador
1	*(Multiplicar) / (dividir)
2	+ (Suma) - (Resta)
3	(Concatenación)
4	Comparaciones (>, <, !=, ...)
5	IS [NOT] NULL, [NOT]LIKE, IN
6	NOT
7	AND
8	OR

4.4. SUBCONSULTAS

Se trata de una técnica que permite utilizar el resultado de una tabla SELECT en otra consulta SELECT. Permite solucionar problemas en los que el mismo dato aparece dos veces. La sintaxis es:

```
SELECT lista_expresiones  
FROM tablas  
WHERE expresión OPERADOR  
( SELECT lista_expresiones  
      FROM tablas );
```

Se puede colocar el SELECT dentro de las cláusulas WHERE, HAVING o FROM. El operador puede ser >,<,>=,<=,!!=, = o IN.

Ejemplo:

```
-- Obtiene los empleados cuyas pagas sean inferiores a lo que gana Martina.  
SELECT nombre_empleado, paga  
FROM EMPLEADOS  
WHERE paga < ( SELECT paga  
                FROM EMPLEADOS  
               WHERE nombre_empleado='Martina');
```

Lógicamente el resultado de la subconsulta debe incluir el campo que estamos analizando.

Se pueden realizar esas subconsultas las veces que haga falta:

```
SELECT nombre_empleado, paga  
FROM EMPLEADOS  
WHERE paga < ( SELECT paga  
                FROM EMPLEADOS  
               WHERE nombre_empleado='Martina')  
AND paga > ( SELECT paga  
                FROM EMPLEADOS  
               WHERE nombre_empleado='Luis');
```

La última consulta obtiene los empleados cuyas pagas estén entre lo que gana Luis y lo que gana Martina. **Una subconsulta que utilice los valores >,<,>=,... tiene que devolver un único valor, de otro modo ocurre un error.** Pero a veces se utilizan consultas del tipo: mostrar el sueldo y nombre de los empleados cuyo sueldo supera al de cualquier empleado del departamento de ventas.

La subconsulta necesaria para ese resultado mostraría los sueldos del departamento de ventas. Pero no podremos utilizar un operador de comparación directamente ya que compararíamos un valor con muchos valores. La solución a esto es utilizar instrucciones especiales entre el operador y la consulta. Esas instrucciones son:

Instrucción	Significado
ANY	Compara con cualquier registro de la subconsulta. La instrucción es válida si hay un registro en la subconsulta que permite que la comparación sea cierta
ALL	Compara con todos los registros de la consulta. La instrucción resulta cierta si es cierta toda comparación con los registros de la subconsulta
IN	No usa comparador, ya que sirve para comprobar si un valor se encuentra en el resultado de la subconsulta
NOT IN	Comprueba si un valor no se encuentra en una subconsulta

Ejemplo:

```
-- Obtiene el empleado que más cobra.
SELECT nombre, sueldo
FROM EMPLEADOS
WHERE sueldo >= ALL ( SELECT sueldo
                      FROM EMPLEADOS );
```

Ejemplo:

```
-- Obtiene los nombres de los empleados cuyos DNI están en la tabla de
directivos.
-- Es decir, obtendrá el nombre de los empleados que son directivos.
SELECT nombre, sueldo
FROM EMPLEADOS
WHERE DNI IN ( SELECT DNI
                  FROM DIRECTIVOS );
```

4.5. ORDENACIÓN

El orden inicial de los registros obtenidos por un SELECT guarda una relación con al orden en el que fueron introducidos. Para ordenar en base a criterios más interesantes, se utiliza la cláusula **ORDER BY**. En esa cláusula se coloca una lista de campos que indica la forma de ordenar. Se ordena primero por el primer campo de la lista, si hay coincidencias por el segundo, si ahí también las hay por el tercero, y así sucesivamente. Se puede colocar las palabras **ASC** O **DESC** (por defecto se toma **ASC**). Esas palabras significan en ascendente

(de la A a la Z, de los números pequeños a los grandes) o en descendente (de la Z a la a, de los números grandes a los pequeños) respectivamente.

Sintaxis completa de SELECT:

```
SELECT * | {[DISTINCT] columna | expresión [[AS] alias], ... }  
FROM nombre_tabla  
[WHERE condición]  
[ORDER BY columna1[{ASC|DESC}]][,columna2[{ASC|DESC}]]...;
```

Ejemplo:

```
-- Devuelve el nombre y los apellidos  
-- de las personas que tienen teléfono, ordenados por  
-- apellido1, luego por apellido2 y finalmente por nombre
```

```
SELECT nombre, apellido1, apellido2  
FROM PERSONAS  
WHERE telefono IS NOT NULL  
ORDER BY apellido1, apellido2, nombre;
```

4.6. FUNCIONES

Oracle incorpora una serie de instrucciones que permiten realizar cálculos avanzados, o bien facilitar la escritura de ciertas expresiones. Todas las funciones reciben datos para poder operar (parámetros) y devuelven un resultado (que depende de los parámetros enviados a la función. Los argumentos se pasan entre paréntesis:

```
NOMBRE_FUNCIÓN [ ( parámetro1 [, parámetros2] ... ) ];
```

Si una función no precisa parámetros (como SYSDATE) no hace falta colocar los paréntesis.

Las funciones pueden ser de dos tipos:

- Funciones que operan con una sola fila
- Funciones que operan con varias filas.

En este apartado, solo veremos las primeras. Más adelante se estudiarán las que operan sobre varias filas.

4.6.1. Funciones de caracteres

Para convertir el texto a mayúsculas o minúsculas:

Función	Descripción
LOWER(texto)	Convierte el texto a minúsculas (funciona con los caracteres españoles)
UPPER(texto)	Convierte el texto a mayúsculas
INITCAP(texto)	Coloca la primera letra de cada palabra en mayúsculas

En la siguiente tabla mostramos las llamadas funciones de transformación:

Función	Descripción
RTRIM(texto)	Elimina los espacios a la derecha del texto
LTRIM(texto)	Elimina los espacios a la izquierda que posea el texto
TRIM(texto)	Elimina los espacios en blanco a la izquierda y la derecha del texto y los espacios dobles del interior.
TRIM(caracteres FROM texto)	Elimina del texto los caracteres indicados. Por ejemplo TRIM('h' FROM nombre) elimina las hachas de la columna <i>nombre</i> que estén a la izquierda y a la derecha
SUBSTR(texto,n[,m])	Obtiene los <i>m</i> siguientes caracteres del texto a partir de la posición <i>n</i> (si <i>m</i> no se indica se cogen desde <i>n</i> hasta el final).
LENGTH(texto)	Obtiene el tamaño del texto
INSTR(texto, textoBuscado [,posInicial [, nAparición]])	Obtiene la posición en la que se encuentra el texto buscado en el texto inicial. Se puede empezar a buscar a partir de una posición inicial concreta e incluso indicar el número de aparición del texto buscado. Ejemplo, si buscamos la letra <i>a</i> y ponemos 2 en <i>nAparición</i> , devuelve la posición de la segunda letra a del texto). Si no lo encuentra devuelve 0
REPLACE(texto, textoABuscar, [textoReemplazo])	Buscar el texto a buscar en un determinado texto y lo cambia por el indicado como texto de reemplazo. Si no se indica texto de reemplazo, entonces está función elimina el texto a buscar
LPAD(texto, anchuraMáxima, [caracterDeRelleno]) RPAD(texto, anchuraMáxima, [caracterDeRelleno])	Rellena el texto a la izquierda (LPAD) o a la derecha (RPAD) con el carácter indicado para ocupar la anchura indicada. Si el texto es más grande que la anchura indicada, el texto se recorta. Si no se indica carácter de relleno se llenará el espacio marcado con espacios en blanco.
REVERSE(texto)	Invierte el texto (le da la vuelta)

Expresiones importantes

LENGTH(cadena): Recibe una cadena, cuenta y devuelve el número de caracteres. Oracle y MySQL.

CHAR_LENGTH(cadena) o CHARACTER_LENGTH (cadena): Recibe una cadena, cuenta y devuelve el número de caracteres. MySQL

Nota:

En MySQL, la diferencia entre LENGTH y CHARACTER_LENGTH es que en CHARACTER_LENGTH un carácter "multibyte" cuenta como un solo carácter. En LENGTH cuenta el número de bytes de la cadena. Así que en el caso de tener una cadena con 5 caracteres que ocupan 2 bytes cada uno, LENGTH devolvería 10 y CHARACTER_LENGTH sólo 5.

4.6.2. Funciones numéricas

Funciones para redondear el número de decimales o redondear a números enteros:

Función	Descripción
ROUND(n,decimales)	Redondea el número al siguiente número con el número de decimales indicado más cercano. ROUND(8.239,2) devuelve 8.24
TRUNC(n,decimales)	Los decimales del número se cortan para que sólo aparezca el número de decimales indicado

En el siguiente cuadro mostramos la sintaxis SQL de funciones matemáticas habituales:

Función	Descripción
MOD(n1,n2)	Devuelve el resto resultado de dividir n1 entre n2
POWER(valor,exponente)	Eleva el valor al exponente indicado
SQRT(n)	Calcula la raíz cuadrada de n
SIGN(n)	Devuelve 1 si n es positivo, cero si vale cero y -1 si es negativo
ABS(n)	Calcula el valor absoluto de n
EXP(n)	Calcula eⁿ , es decir el exponente en base e del número n
LN(n)	Logaritmo neperiano de n
LOG(n)	Logaritmo en base 10 de n
SIN(n)	Calcula el seno de n (n tiene que estar en radianes)
COS(n)	Calcula el coseno de n (n tiene que estar en radianes)
TAN(n)	Calcula la tangente de n (n tiene que estar en radianes)
ACOS(n)	Devuelve en radianes el arco coseno de n
ASIN(n)	Devuelve en radianes el arco seno de n
ATAN(n)	Devuelve en radianes el arco tangente de n
SINH(n)	Devuelve el seno hiperbólico de n
COSH(n)	Devuelve el coseno hiperbólico de n
TANH(n)	Devuelve la tangente hiperbólica de n

4.6.3. Funciones de fecha

Las fechas se utilizan muchísimo en todas las bases de datos. Oracle proporciona dos tipos de datos para manejar fechas, los tipos **DATE** y **TIMESTAMP**. En el primer caso se almacena una fecha concreta (que incluso puede contener la hora), en el segundo caso se almacena un instante de tiempo más concreto que puede incluir incluso fracciones de segundo. Hay que tener en cuenta que a los valores de tipo fecha se les pueden sumar números y se entendería que esta suma es de días. Si tiene decimales entonces se suman días, horas, minutos y segundos. La diferencia entre dos fechas también obtiene un número de días.

Funciones para obtener la fecha y hora actual

Función	Descripción
SYSDATE	Obtiene la fecha y hora actuales
SYSTIMESTAMP	Obtiene la fecha y hora actuales en formato TIMESTAMP

Funciones para calcular fechas:

Función	Descripción
ADD_MONTHS(fecha,n)	Añade a la fecha el número de meses indicado por n
MONTHS_BETWEEN(fecha1, fecha2)	Obtiene la diferencia en meses entre las dos fechas (puede ser decimal)
NEXT_DAY(fecha,día)	Indica cual es el día que corresponde a añadir a la fecha el día indicado. El día puede ser el texto ' Lunes ', ' Martes ', ' Miércoles ',... (si la configuración está en español) o el número de día de la semana (1=lunes, 2=martes,...)
LAST_DAY(fecha)	Obtiene el último día del mes al que pertenece la fecha. Devuelve un valor DATE
EXTRACT(valor FROM fecha)	Extrae un valor de una fecha concreta. El valor puede ser day (día), month (mes), year (año), etc.
GREATEST(fecha1, fecha2,..)	Devuelve la fecha más moderna la lista
LEAST(fecha1, fecha2,..)	Devuelve la fecha más antigua la lista
ROUND(fecha [,formato])	Redondea la fecha al valor de aplicar el formato a la fecha. El formato puede ser: 'YEAR' Hace que la fecha refleje el año completo 'MONTH' Hace que la fecha refleje el mes completo más cercano a la fecha 'HH24' Redondea la hora a las 00:00 más cercanas 'DAY' Redondea al día más cercano
TRUNC(fecha [formato])	Igual que el anterior pero trunca la fecha en lugar de redondearla.

4.6.4. Funciones de conversión

Oracle es capaz de convertir datos automáticamente a fin de que la expresión final tenga sentido. En ese sentido son fáciles las conversiones de texto a número y viceversa.

Ejemplos:

```
-- El resultado es 8  
SELECT 5+3'  
FROM DUAL;
```

```
-- El resultado es 53  
SELECT 5||'3'  
FROM DUAL;
```

Pero en determinadas ocasiones queremos realizar conversiones explícitas. Para hacerlo se utilizan las expresiones como TO_CHAR(), TO_NUMBER() o TO_DATE() vistas en el tema 3.

4.7. AGRUPACIONES

Es muy común utilizar consultas en las que se desee agrupar los datos a fin de realizar **cálculos en vertical, es decir calculados a partir de datos de distintos registros**. Para ello se utiliza la cláusula **GROUP BY** que permite indicar en base a qué registros se realiza la agrupación. Con GROUP BY la instrucción SELECT queda de esta forma:

```
SELECT lista_expresiones  
FROM lista_tablas  
[ WHERE condiciones ]  
[ GROUP BY grupos ]  
[ HAVING condiciones_de_grupos ]  
[ ORDER BY columnas ];
```

En el apartado GROUP BY, se indican las columnas por las que se agrupa. **La función de este apartado es crear un único registro por cada valor distinto en las columnas del grupo.**

Vamos a ver un ejemplo de cómo funciona GROUP BY. Supongamos que tenemos la siguiente tabla EXISTENCIAS:

TI	MODELO	N_ALMACEN	CANTIDAD
AR	6	1	2500
AR	6	2	5600
AR	6	3	2430
AR	9	1	250
AR	9	2	4000
AR	9	3	678
AR	15	1	5667
AR	20	3	43
BI	10	2	340
BI	10	3	23
BI	38	1	1100
BI	38	2	540
BI	38	3	

Si por ejemplo agrupamos en base a las columnas tipo y modelo, la sintaxis sería la siguiente:

```
SELECT tipo, modelo
FROM EXISTENCIAS
GROUP BY tipo, modelo;
```

Al ejecutarla, en la tabla de existencias, se creará un único registro por cada tipo y modelo distintos, generando la siguiente salida:

TI	MODELO
AR	6
AR	9
AR	15
AR	20
BI	10
BI	38

Es decir, es un resumen de los datos anteriores. **Pero observamos que los datos n_almacen y cantidad no están disponibles directamente ya que son distintos en los registros del mismo grupo.**

Si los hubiésemos seleccionado también en la consulta habríamos ejecutado una consulta ERRÓNEA. Es decir, al ejecutar:

```
SELECT tipo, modelo, cantidad  
FROM EXISTENCIAS  
GROUP BY tipo, modelo;
```

Habríamos obtenido un mensaje de error.

ERROR en línea 1:

ORA-00979: no es una expresión GROUP BY

Es decir, esta consulta es errónea, porque se está intentando mostrar datos de registros que están agrupados por otros datos.

4.7.1. Funciones de cálculo con grupo (o funciones colectivas)

Lo interesante de la creación de grupos es la posibilidad de cálculo que ofrece. Para ello se utilizan las funciones que permiten trabajar con los registros de un grupo. Estas son:

Función	Significado
COUNT(*)	Cuenta los elementos de un grupo. Se utiliza el asterisco para no tener que indicar un nombre de columna concreto, el resultado es el mismo para cualquier columna
SUM(expresión)	Suma los valores de la expresión
AVG(expresión)	Calcula la media aritmética sobre la expresión indicada
MIN(expresión)	Mínimo valor que toma la expresión indicada
MAX(expresión)	Máximo valor que toma la expresión indicada
STDDEV(expresión)	Calcula la desviación estándar
VARIANCE(expresión)	Calcula la varianza

Expresiones importantes

- **COUNT()**
- **SUM()**
- **AVG()**
- **MIN()**
- **MAX()**

Las funciones anteriores se aplicarán sobre todos los elementos del grupo. Así, por ejemplo, podemos calcular la suma de las cantidades para cada tipo y modelo de la tabla EXISTENCIAS. (Es como si lo hiciéramos

manualmente con las antiguas fichas sobre papel: primero las separaríamos en grupos poniendo juntas las que tienen el mismo tipo y modelo y luego para cada grupo sumaríamos las cantidades). La sintaxis SQL de dicha consulta quedaría:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
GROUP BY tipo, modelo;
```

Y se obtiene el siguiente resultado, en el que se suman las cantidades para cada grupo.

TI	MODELO	SUM(CANTIDAD)
AR	6	10530
AR	9	4928
AR	15	5667
AR	20	43
BI	10	363
BI	38	1740

Nota

Estas funciones se pueden utilizar sin la necesidad de emplear GROUP BY aplicándose en este caso a toda una columna de todos los registros. Se considera como un único grupo todos los registros.

IMPORTANTE

Si se utilizan estas funciones fuera de un grupo, no pueden usarse para mostrarse junto a otra información:

```
SELECT tipo, modelo, MAX(cantidad)
FROM EXISTENCIAS;
```

El campo tipo y modelo serían los del primer registro y no los del registro con mayor cantidad. Para realizar esto se debe utilizar una subconsulta:

```
SELECT tipo, modelo, cantidad
FROM EXISTENCIAS
WHERE cantidad = (SELECT MAX(cantidad)
                  FROM EXISTENCIAS);
```

4.7.2. Condiciones HAVING

A veces se desea restringir el resultado de una función agrupada y no aplicarla a todos los grupos. Por ejemplo, imaginemos que queremos realizar la consulta

anterior, es decir queremos calcular la suma de las cantidades para cada tipo y modelo de la tabla EXISTENCIAS, pero queremos que se muestren solo los registros en los que la suma de las cantidades calculadas sea mayor que 500. Si planteamos la consulta del modo siguiente:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
WHERE SUM(cantidad) > 500
GROUP BY tipo, modelo;
```

Habríamos ejecutado una consulta ERRÓNEA.

ERROR en línea 3:

ORA-00934: función de grupo no permitida aquí

La razón es que Oracle calcula **primero el WHERE y luego los grupos**; por lo que esa condición no la puede realizar al no estar establecidos los grupos. Es decir, no puede saber que grupos tienen una suma de cantidades mayor que 500 cuando todavía no ha aplicado los grupos. Por ello se utiliza la cláusula **HAVING**, cuya ejecución se efectúa una vez realizados los grupos. Se usaría de esta forma:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
GROUP BY tipo, modelo
HAVING SUM(cantidad) > 500;
```

Ahora bien, **esto no implica que con la cláusula GROUP BY no podamos emplear un WHERE**. Esta expresión puede usarse para imponer condiciones sobre las filas de la tabla antes de agruparlas. Por ejemplo, la siguiente expresión es correcta:

```
SELECT tipo, modelo, SUM(cantidad)
FROM EXISTENCIAS
WHERE tipo='AR'
GROUP BY tipo, modelo
HAVING SUM(cantidad) > 500;
```

De la tabla EXISTENCIAS tomará solo aquellas filas cuyo tipo sea AR, luego agrupará según tipo y modelo y dejará sólo aquellos grupos en los que

$\text{SUM}(\text{cantidad}) > 500$ y por último mostrará tipo, modelo y la suma de las cantidades para aquellos grupos que cumplan dicha condición. En definitiva, el orden de ejecución de la consulta marca lo que se puede utilizar con WHERE y lo que se puede utilizar con HAVING.

Pasos en la ejecución de una consulta SELECT el gestor de bases de datos sigue el siguiente orden:

1. Se aplica la cláusula FROM, de manera que determina sobre qué tablas se va a ejecutar la consulta.
2. Se seleccionan las filas deseadas utilizando WHERE. (Solo quedan las filas que cumplen las condiciones especificadas en el WHERE).
3. Se establecen los grupos indicados en la cláusula GROUP BY.
4. Se calculan los valores de las funciones de totales o colectivas que se especifiquen en el HAVING (COUNT, SUM, AVG, ...)
5. Se filtran los registros que cumplen la cláusula HAVING
6. Se aplica la cláusula SELECT que indica las columnas que mostraremos en la consulta.
7. El resultado se ordena en base al apartado ORDER BY.

4.8. OBTENER DATOS DE MÚLTIPLES TABLAS

Es más que habitual necesitar en una consulta datos que se encuentran distribuidos en varias tablas. **Las bases de datos relacionales se basan en que los datos se distribuyen en tablas que se pueden relacionar mediante un campo.** Ese campo es el que permite integrar los datos de las tablas. A continuación, veremos cómo se pueden realizar las consultas entre varias tablas. Para ello partiremos del siguiente ejemplo: Supongamos que disponemos de una tabla de EMPLEADOS cuya clave principal es el DNI y otra tabla de TAREAS que se refiere a las tareas realizadas por los empleados. Suponemos que cada empleado realizará múltiples tareas, pero que cada tarea es realizada por un único empleado. Si el diseño está bien hecho, en la tabla de TAREAS aparecerá el DNI del empleado (como clave foránea) para saber qué empleado realizó la tarea.

4.8.1. Producto cruzado o cartesiano de tablas

En el ejemplo anterior, si se quiere obtener una lista de los datos de las tareas y los empleados, se podría hacer de esta forma:

```
SELECT cod_tarea, descripción_tarea, dni_empleado, nombre_empleado  
FROM TAREAS, EMPLEADOS;
```

Aunque la sintaxis es correcta ya que, efectivamente, en el apartado FROM se pueden indicar varias tablas separadas por comas, al ejecutarla produce un producto cruzado de las tablas. Es decir, aparecerán todos los registros de las tareas relacionados con todos los registros de empleados (y no para cada

empleado sus tareas específicas). El producto cartesiano pocas veces es útil para realizar consultas. Nosotros necesitamos discriminar ese producto para que sólo aparezcan los registros de las tareas relacionadas con sus empleados correspondientes. A eso se le llama asociar tablas (join) y se ve en el siguiente apartado.

4.8.2. Asociando tablas

La forma de realizar correctamente la consulta anterior (asociado las tareas con los empleados que la realizaron) sería:

```
SELECT cod_tarea, descripción_tarea, dni_empleado, nombre_empleado  
FROM TAREAS, EMPLEADOS  
WHERE TAREAS.dni_empleado = EMPLEADOS.dni_empleado;
```

Nótese que se utiliza la notación **tabla.columna** para evitar la ambigüedad, ya que el mismo nombre de campo se puede repetir en ambas tablas. Para evitar repetir continuamente el nombre de la tabla, se puede utilizar un alias de tabla:

```
SELECT T.cod_tarea, T.descripción_tarea, E.dni_empleado,  
E.nombre_empleado  
FROM TAREAS T, EMPLEADOS E  
WHERE T.dni_empleado = E.dni_empleado;
```

A la sintaxis WHERE se le pueden añadir condiciones sin más que encadenarlas con el operador AND.

Ejemplo:

```
SELECT T.cod_tarea, T.descripción_tarea  
FROM TAREAS T, EMPLEADOS E  
WHERE T.dni_empleado = E.dni_empleado AND E.nombre_empleado =  
'Javier';
```

Finalmente indicar que se pueden enlazar más de dos tablas a través de sus claves principales y foráneas. Por cada relación necesaria entre tablas, aparecerá una condición (igualando la clave principal y la foránea correspondiente) en el WHERE.

Ejemplo:

```
SELECT T.cod_tarea, T.descripción_tarea, E.nombre_empleado,  
U.nombre_utensilio
```

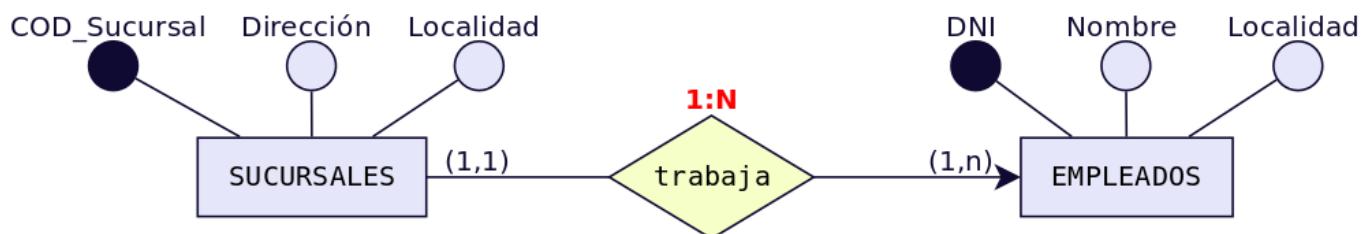
```

FROM TAREAS T, EMPLEADOS E, UTENSILIOS U
WHERE T.dni_empleado = E.dni_empleado AND T.cod_tarea = U.cod_tarea;

```

4.8.3. Combinación de tablas (JOIN)

Existe otra forma más moderna e intuitiva de trabajar con varias tablas. Para ello se utiliza la cláusula **JOIN**. Supongamos que tenemos una base de datos de una entidad bancaria. Disponemos de una tabla con sus empleados y otra tabla con sus sucursales. En una sucursal trabajan varios empleados. Los empleados viven en una localidad y trabajan en una sucursal situada en la misma localidad o en otra localidad. El esquema E-R es el siguiente:



Los datos de las tablas son:

EMPLEADOS			
DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL
11111111A	ANA	ALMERÍA	0001
22222222B	BERNARDO	GRANADA	0001
33333333C	CARLOS	GRANADA	
44444444D	DAVID	JEREZ	0003
SUCURSALES			
COD_SUCURSAL	DIRECCIÓN	LOCALIDAD	
0001	C/ ANCHA, 1	ALMERÍA	
0002	C/ NUEVA, 1	GRANADA	
0003	C/ CORTÉS, 33	CÁDIZ	

Se observa que Ana vive en Almería y trabaja en la sucursal 0001 situada en Almería. Bernardo vive en Granada, pero trabaja en la sucursal 0001 de Almería. Carlos es un empleado del que no disponemos el dato acerca de la sucursal en la que trabaja. David es un empleado que vive en Jerez de la Frontera y trabaja en la sucursal 0003 en Cádiz. Existe otra sucursal 0002 en Granada donde no aparece registrado ningún empleado.

Existen diversas formas de combinar (JOIN) las tablas según la información que deseemos obtener. Los tipos de JOIN se clasifican en:

- INNER JOIN (o simplemente JOIN): Combinación interna.

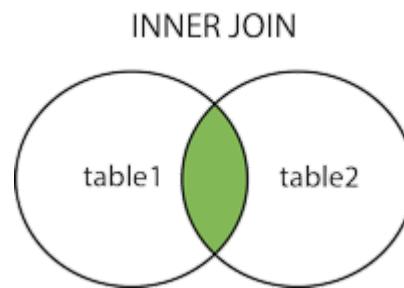
- JOIN
- SELF JOIN
- NATURAL JOIN
- OUTER JOIN: Combinación externa.
 - LEFT OUTER JOIN (o simplemente LEFT JOIN)
 - RIGHT OUTER JOIN (o simplemente RIGHT JOIN)
 - FULL OUTER JOIN (o simplemente FULL JOIN)
 - CROSS JOIN: Combinación cruzada.

Pasamos a continuación a explicar cada uno de ellos.

4.8.4.1. INNER JOIN

También se conoce como EQUI JOIN o combinación de igualdad. **Esta combinación devuelve todas las filas de ambas tablas donde hay una coincidencia.** Este tipo de unión se puede utilizar en la situación en la que sólo debemos seleccionar las filas que tienen valores comunes en las columnas que se especifican en la cláusula ON.

INNER JOIN selecciona todas las filas de ambas tablas **siempre que haya una coincidencia** entre las columnas. Si hay registros en la Tabla1 que no tienen coincidencias en la Tabla2, **¡estos registros no se mostrarán!**



4.8.4.1.1. JOIN

En lugar de **INNER JOIN** es más frecuente encontrarlo escrito como **JOIN** simplemente:

Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
    TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 JOIN TABLA2
    ON TABLA1.columnaX = TABLA2.columnaY;
```

Por ejemplo, para ver los empleados con sucursal asignada:

```
SELECT E.*, S.LOCALIDAD
FROM EMPLEADOS E JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL	LOCALIDAD
22222222B	BERNARDO	GRANADA	0001	ALMERÍA
11111111A	ANA	ALMERÍA	0001	ALMERÍA
44444444D	DAVID	JEREZ	0003	CÁDIZ

En esta consulta, utilizamos la combinación interna basada en la columna «COD_SUCURSAL» que es común en las tablas «EMPLEADOS» y «SUCURSALES». Esta consulta dará todas las filas de ambas tablas que tienen valores comunes en la columna «COD_SUCURSAL»

4.8.4.1.2. SELF JOIN

En algún momento podemos necesitar **unir una tabla consigo misma**. Este tipo de combinación se denomina **SELF JOIN**. En este JOIN, necesitamos abrir dos copias de una misma tabla en la memoria. Dado que el nombre de tabla es el mismo para ambas instancias, usamos los alias de tabla para hacer copias idénticas de la misma tabla que se abran en diferentes ubicaciones de memoria.

Nota

Observa que no existe la cláusula SELF JOIN, solo **JOIN**.

Sintaxis:

```
SELECT ALIAS1.columna1, ALIAS1.columna2, ..., ALIAS2.columna1, ...
FROM TABLA ALIAS1 JOIN TABLA ALIAS2
ON ALIAS1.columnaX = ALIAS2.columnaY;
```

Ejemplo:

```
SELECT E1.NOMBRE, E2.NOMBRE, E1.LOCALIDAD
FROM EMPLEADOS E1 JOIN EMPLEADOS E2
ON E1.LOCALIDAD = E2.LOCALIDAD;
```

NOMBRE	NOMBRE	LOCALIDAD
ANA	ANA	ALMERÍA
CARLOS	BERNARDO	GRANADA
BERNARDO	BERNARDO	GRANADA
CARLOS	CARLOS	GRANADA
BERNARDO	CARLOS	GRANADA
DAVID	DAVID	JEREZ

Esto muestra las combinaciones de los empleados que viven en la misma localidad. En este caso no es de mucha utilidad, pero el SELF JOIN puede ser muy útil en relaciones reflexivas.

4.8.4.1.3. NATURAL JOIN

NATURAL JOIN establece una relación de igualdad entre las tablas a través de los campos que tengan el mismo nombre en ambas tablas. Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
    TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 NATURAL JOIN TABLA2;
```

En este caso **no existe cláusula ON** puesto que se realiza la combinación teniendo en cuenta las columnas del mismo nombre. Por ejemplo:

```
SELECT *
FROM EMPLEADOS E NATURAL JOIN SUCURSALES S;
```

LOCALIDAD	COD_SUCURSAL	DNI	NOMBRE	DIRECCIÓN
ALMERÍA	0001	11111111A	ANA	C/ ANCHA, 1

En el resultado de la consulta nos aparece la combinación donde la (LOCALIDAD, COD_SUCURSAL) de EMPLEADOS es igual a (LOCALIDAD, COD_SUCURSAL) de SUCURSALES. Es decir, estamos mostrando todos los empleados que tienen asignada una sucursal y dicha sucursal está en la localidad donde vive el empleado. **El NATURAL JOIN elimina columnas duplicadas**, por eso no aparecen los campos LOCALIDAD ni SUCURSAL duplicados. Este tipo de consulta **no permite indicar estos campos en la cláusula SELECT**. Por ejemplo: SELECT E.LOCALIDAD o SELECT E.COD_SUCURSAL sería incorrecto.

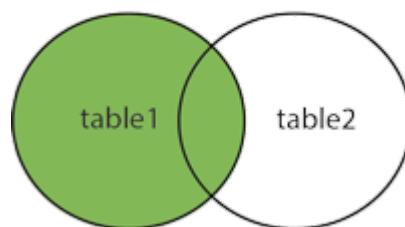
4.8.4.2. OUTER JOIN

La combinación externa o OUTER JOIN es muy útil cuando deseamos averiguar qué campos están a NULL en un lado de la combinación. En nuestro

ejemplo, podemos ver qué empleados no tienen sucursal asignada; también podemos ver que sucursales no tienen empleados asignados.

4.8.4.2.1. LEFT JOIN

También conocido como **LEFT OUTER JOIN**, nos permite obtener todas las filas de la primera tabla asociadas a filas de la segunda tabla. **Si no existe correspondencia en la segunda tabla, dichos valores aparecen como NULL.**



Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
      TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 LEFT JOIN TABLA2
ON TABLA1.columnaX = TABLA2.columnaY;
```

Por ejemplo:

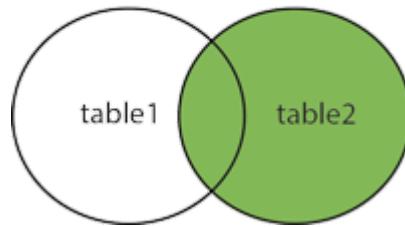
```
SELECT E.*, S.LOCALIDAD
FROM EMPLEADOS E LEFT JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL	LOCALIDAD
11111111A	ANA	ALMERÍA	0001	ALMERÍA
22222222B	BERNARDO	GRANADA	0001	ALMERÍA
33333333C	CARLOS	GRANADA		
44444444D	DAVID	JEREZ	0003	CÁDIZ

Todos los empleados tienen una sucursal asignada salvo el empleado Carlos.

4.8.4.2.2. RIGHT JOIN

También conocido como **RIGHT OUTER JOIN**, nos permite obtener todas las filas de la segunda tabla asociadas a filas de la primera tabla. **Si no existe correspondencia en la segunda tabla, dichos valores aparecen como NULL.**



Su sintaxis es:

```
SELECT TABLA1.columna1, TABLA1.columna2, ...
    TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 RIGHT JOIN TABLA2
ON TABLA1.columnaX = TABLA2.columnaY;
```

Por ejemplo:

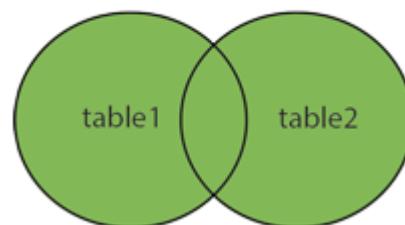
```
SELECT E.DNI, E.NOMBRE, S.*
FROM EMPLEADOS E RIGHT JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;
```

DNI	NOMBRE	COD_SUCURSAL	DIRECCIÓN	LOCALIDAD
11111111A	ANA	0001	C/ ANCHA, 1	ALMERÍA
2222222B	BERNARDO	0001	C/ ANCHA, 1	ALMERÍA
4444444D	DAVID	0003	C/ CORTÉS, 33	CÁDIZ
		0002	C/ NUEVA, 1	GRANADA

Todas las sucursales tienen algún empleado asignado salvo la sucursal 0002.

4.8.4.2.3. FULL JOIN (Oracle)

También conocido como **FULL OUTER JOIN**, nos permite obtener todas las filas de la primera tabla asociadas a filas de la segunda tabla. **Si no existe correspondencia en alguna de las tablas, dichos valores aparecen como NULL.**



Su sintaxis es:

```

SELECT TABLA1.columna1, TABLA1.columna2, ...
    TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 FULL JOIN TABLA2
ON TABLA1.columnaX = TABLA2.columnaY;

```

Por ejemplo:

```

SELECT E.DNI, E.NOMBRE, S.COD_SUCURSAL, S.LOCALIDAD
FROM EMPLEADOS E FULL JOIN SUCURSALES S
ON E.COD_SUCURSAL = S.COD_SUCURSAL;

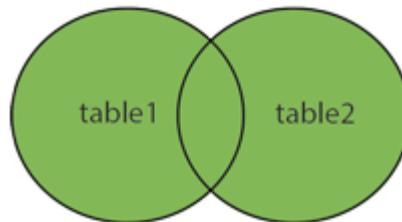
```

DNI	NOMBRE	COD_SUCURSAL	LOCALIDAD
22222222B	BERNARDO	0001	ALMERÍA
11111111A	ANA	0001	ALMERÍA
		0002	GRANADA
44444444D	DAVID	0003	CÁDIZ
33333333C	CARLOS		

Como puede observarse fácilmente, vemos que en la sucursal 0002 no hay ningún empleado asignado y que el empleado Carlos no tiene asignada ninguna sucursal.

4.8.4.3. CROSS JOIN

El **CROSS JOIN** o combinación cruzada produce el mismo resultado del producto cartesiano, es decir nos da **todas las combinaciones posibles**.



Su sintaxis es:

```

SELECT TABLA1.columna1, TABLA1.columna2, ...
    TABLA2.columna1, TABLA2.columna2, ...
FROM TABLA1 CROSS JOIN TABLA2;

```

Por ejemplo:

```
SELECT E.DNI, E.NOMBRE, E.LOCALIDAD, S.COD_SUCURSAL,
S.LOCALIDAD
FROM EMPLEADOS E CROSS JOIN SUCURSALES S;
```

DNI	NOMBRE	LOCALIDAD	COD_SUCURSAL	LOCALIDAD
11111111A	ANA	ALMERÍA	0001	ALMERÍA
11111111A	ANA	ALMERÍA	0002	GRANADA
11111111A	ANA	ALMERÍA	0003	CÁDIZ
22222222B	BERNARDO	GRANADA	0001	ALMERÍA
22222222B	BERNARDO	GRANADA	0002	GRANADA
22222222B	BERNARDO	GRANADA	0003	CÁDIZ
33333333C	CARLOS	GRANADA	0001	ALMERÍA
33333333C	CARLOS	GRANADA	0002	GRANADA
33333333C	CARLOS	GRANADA	0003	CÁDIZ
44444444D	DAVID	JEREZ	0001	ALMERÍA
44444444D	DAVID	JEREZ	0002	GRANADA
44444444D	DAVID	JEREZ	0003	CÁDIZ

En el ejemplo que estamos viendo nos mostraría 12 filas (4x3: 4 filas de empleados x 3 filas de sucursales). El primer cliente se combina con todas las sucursales. El segundo cliente igual. Y así sucesivamente. Esta combinación asocia todas las filas de la tabla izquierda con cada fila de la tabla derecha. Este tipo de unión es necesario cuando necesitamos seleccionar todas las posibles combinaciones de filas y columnas de ambas tablas. **Este tipo de unión no es generalmente preferido ya que toma mucho tiempo y da un resultado enorme que no es a menudo útil.**

4.9. COMBINACIONES ESPECIALES

4.9.1. Uniones

La palabra **UNION** permite añadir el resultado de un SELECT a otro SELECT. Para ello ambas instrucciones tienen que utilizar el mismo número y tipo de columnas.

Ejemplo:

- Tipos y modelos de piezas que se encuentren el almacén 1, en el 2 o en ambos.

```
SELECT tipo,modelo FROM existencias
```

```
WHERE n_almacen = 1
```

```
UNION
```

```
SELECT tipo,modelo FROM existencias  
WHERE n_almacen = 2;
```

Nota

Observa que sólo se pone punto y coma al final.

Es decir, UNION crea una sola tabla con registros que estén presentes en cualquiera de las consultas. **Si están repetidas sólo aparecen una vez, para mostrar los duplicados se utiliza UNION ALL en lugar de la palabra UNION.**

4.10. CONSULTA DE VISTAS

A efectos de uso, la consulta de una vista es idéntica a **la consulta de una tabla**. Supongamos que tenemos la siguiente vista:

```
CREATE VIEW RESUMEN AS  
SELECT L.IdLocalidad, L.Nombre, L.Poblacion,  
P.IdProvincia, P.Nombre, P.Superficie,  
C.IdComunidad, C.Nombre  
FROM LOCALIDADES L JOIN PROVINCIAS P  
ON L.IdProvincia = P.IdProvincia  
JOIN COMUNIDADES C ON P.IdComunidad = C.IdComunidad;
```

Podemos consultar ahora sobre la vista como si de una tabla se tratase

```
SELECT * FROM resumen;
```

4.11. Limitar resultados

Podemos limitar el número de registros que queremos mostrar. Oracle emplea una forma diferente a la que emplea MySQL.

En Oracle hacemos lo siguiente:

```
SELECT * FROM tabla WHERE rownum <= n;
```

Donde n es el número de registros que queremos mostrar.

En MySQL hacemos lo siguiente:

```
SELECT * FROM table LIMIT n;
```

Donde n es el número de registros que queremos mostrar.