## 5.1 Introducción

En esta unidad vamos a continuar haciendo consultas a la base de datos. Nos ocuparemos de nuevas cláusulas que acompañan a la sentencia SELECT y que permiten llegar a consultas más complejas. Veremos las órdenes que nos permiten agrupar filas de una tabla según alguna condición, o sin ninguna condición, para obtener algún resultado referente a ese grupo de filas agrupadas. Un ejemplo de esta agrupación es averiguar cuál es la suma de salarios por cada departamento de la tabla EMPLE.

También nos ocuparemos de otro tipo de combinaciones de tablas: la que nos permite seleccionar algunas filas de una tabla aunque éstas no tengan su correspondencia con la otra tabla. Estudiaremos cómo podemos combinar los resultados de varias sentencias SELECT utilizando operadores de conjuntos.

## 5.2 Agrupación de elementos. GROUP BY y HAVING

Hasta ahora hemos utilizado la **orden SELECT** para recuperar filas de una tabla y la **cláusula WHERE** para seleccionar el número de filas que se recuperan. También hemos empleado funciones de grupo para trabajar con conjuntos de filas. Se puede pensar en estos conjuntos como si fueran un grupo; así calculamos, por ejemplo, el salario medio de todos los empleados: `SELECT AVG(SALARIO) FROM EMPLE;` o la suma de todos los salarios: `SELECT SUM(SALARIO) FROM EMPLE;`.

Pero, a veces, nos interesa consultar los datos según grupos determinados. Así, para saber cuál es el salario medio de cada departamento de la tabla EMPLE, las cláusulas que conocemos hasta ahora no son suficientes. Necesitamos realizar un agrupamiento por departamento. Para ello utilizaremos la cláusula **GROUP BY**. La consulta sería la siguiente:

```
SELECT DEPT_NO, AVG(SALARIO) FROM EMPLE GROUP BY DEPT_NO;
```

La sentencia SELECT posibilita agrupar uno o más conjuntos de filas. El agrupamiento se lleva a cabo mediante la cláusula GROUP BY por las columnas especificadas y en el orden especificado. Éste es su formato:

```
SELECT ...
FROM ...
GROUP BY columna1, columna2, columna3,...
HAVING condición
ORDER BY ...
```

Los datos seleccionados en la sentencia SELECT que lleva el GROUP BY deben ser: una constante, una función de grupo (SUM, COUNT, AVG, ...), una columna expresada en el GROUP BY.

# 5. Cláusulas avanzadas de selección

## 5.2 Agrupación de elementos. GROUP BY y HAVING

La cláusula **GROUP BY** sirve para calcular propiedades de uno o más conjuntos de filas. Además, si se selecciona más de un conjunto de filas, GROUP BY controla que las filas de la tabla original sean agrupadas en una temporal. Del mismo modo que existe la condición de búsqueda **WHERE** para filas individuales, también hay una condición de búsqueda para grupos de filas: **HAVING.** La cláusula HAVING se emplea para controlar cuál de los conjuntos de filas se visualiza. Se evalúa sobre la tabla que devuelve el GROUP BY. No puede existir sin GROUP BY.

### Caso práctico

**1 Visualiza a partir de la tabla EMPLE el número de empleados que hay en cada departamento.**

Para hacer esta consulta, tenemos que agrupar las filas de la tabla EMPLE por departamento (GROUP BY DEPT_NO) y contarlas (COUNT(*)). La consulta es la siguiente:

```
SQL> SELECT DEPT_NO, COUNT(*) FROM EMPLE GROUP BY DEPT_NO;

   DEPT_NO      COUNT(*)
   -------   ----------
        10             3
        20             5
        30             6
```

COUNT es una función de grupo y da información sobre un grupo de filas, no sobre filas individuales de la tabla. La cláusula GROUP BY DEPT_NO obliga a COUNT a contar las filas que se han agrupado por cada departamento.

Si en la consulta anterior sólo queremos visualizar los departamentos con más de 4 empleados, tendríamos que escribir lo siguiente:

```
SQL> SELECT DEPT_NO, COUNT(*) FROM EMPLE GROUP BY DEPT_NO HAVING COUNT(*) > 4;

   DEPT_NO      COUNT(*)
   -------   ----------
        20             5
        30             6
```

### Actividades propuestas

**1 Visualiza los departamentos en los que el salario medio es mayor o igual que la media de todos los salarios.**

La cláusula **HAVING** es similar a la cláusula WHERE, pero trabaja con grupos de filas; pregunta por una característica de grupo, es decir, pregunta por los resultados de las funciones de grupo, lo cual WHERE no puede hacer. En el ejemplo anterior se visualizan las filas cuyo número de empleados sea mayor de 4 (**HAVING** COUNT(*) > 4).

Si queremos ordenar la salida descendentemente por número de empleados, obteniendo el resultado de la Tabla 5.1, utilizamos la siguiente orden ORDER BY:

```
SQL> SELECT DEPT_NO, COUNT(*) FROM EMPLE
  2  GROUP BY DEPT_NO
  3  HAVING COUNT(*) > 4
  4  ORDER BY COUNT(*) DESC;
```

| DEPT_NO | COUNT(*) |
|---|---|
| 30 | 6 |
| 20 | 5 |

**Tabla 5.1.** *HAVING y ORDER BY.*

Cuando usamos la cláusula ORDER BY con columnas y funciones de grupo hemos de tener en cuenta que ésta se ejecuta detrás de las cláusulas WHERE, GROUP BY y HAVING. En ORDER BY podemos especificar funciones de grupo, columnas de GROUP BY o su combinación.

La evaluación de las cláusulas en tiempo de ejecución se efectúa en el siguiente orden:

```
WHERE    Selecciona las filas.
GROUP BY Agrupa estas filas.
HAVING   Filtra los grupos. Selecciona y elimina los grupos.
ORDER BY Clasifica la salida. Ordena los grupos.
```

## Caso práctico

**2 Consideramos las tablas EMPLE y DEPART. Obtén la suma de salarios, el salario máximo y el salario mínimo por cada departamento; la salida de los cálculos debe estar formateada:**

```
SQL> SELECT DEPT_NO, TO_CHAR (SUM(SALARIO), '99G999D99') "Suma",
  2  TO_CHAR (MAX(SALARIO), '99G999D99') "Máximo",
  3  TO_CHAR (MIN(SALARIO), '99G999D99') "Mínimo"
  4  FROM EMPLE GROUP BY DEPT_NO;

   DEPT_NO        Suma          Máximo        Mínimo
---------- ------------ ------------ ------------
        10     8.675,00     4.100,00     1.690,00
        20    11.370,00     3.000,00     1.040,00
        30    10.415,00     3.005,00     1.335,00
```

Calcula el número de empleados que realizan cada OFICIO en cada DEPARTAMENTO. Los datos que se visualizan son: departamento, oficio y número de empleados. Necesitamos agrupar por departamento y dentro de cada departamento, por oficio:

```
SQL> SELECT DEPT_NO, OFICIO, COUNT(*) FROM EMPLE GROUP BY DEPT_NO, OFICIO;

   DEPT_NO      OFICIO          COUNT(*)
---------- ----------- ----------
        10  DIRECTOR              1
        10  EMPLEADO              1
        10  PRESIDENTE            1
        20  ANALISTA              2
```

*(Continúa)*

*(Continuación)*

```
        20          DIRECTOR                1
        20          EMPLEADO                2
        30          DIRECTOR                1
        30          EMPLEADO                1
        30          VENDEDOR                4

9 filas seleccionadas.
```

Busca el número máximo de empleados que hay en algún departamento:

```
SQL> SELECT  MAX(COUNT(*)) "Máximo"  FROM EMPLE GROUP BY DEPT_NO;

    Máximo
----------
         6
```

**Actividades propuestas**

**2** **Obtén los nombres de departamentos que tengan más de 4 personas trabajando.**

Visualiza el número de departamento, el nombre de departamento y el número de empleados del departamento con más empleados.

## 5.3 Combinación externa (OUTER JOIN)

Ya hemos tratado el concepto de *combinación de tablas*. Existe una variedad de combinación de tablas que se llama **OUTER JOIN** y que nos permite seleccionar algunas filas de una tabla aunque éstas no tengan correspondencia con las filas de la otra tabla con la que se combina. El formato es el siguiente:

```
SELECT tabla1.colum1, tabla1.colum2, tabla2.colum1
FROM tabla1, tabla2
WHERE tabla1.colum1 = tabla2.colum1(+);
```

Esta SELECT seleccionará todas las filas de la tabla *tabla1*, aunque no tengan correspondencia con las filas de la tabla *tabla2*; se denota con el símbolo **(+)** detrás de la columna de la *tabla2* (que es la tabla donde no se encuentran las filas) en la cláusula WHERE. Se obtendrá una fila con las columnas de *tabla1* y el resto de columnas de la *tabla2* se rellena con NULL.

Veamos el funcionamiento del OUTER JOIN con un ejemplo.

Sean las tablas EMPLE Y DEPART, comprobamos su contenido ejecutando desde SQL:

```
SELECT * FROM DEPART; y SELECT * FROM EMPLE ORDER BY DEPT_NO;
```

Si nos fijamos en la tabla EMPLE, vemos que el departamento 40 no existe, sin embargo, en DEPART sí existe.

Queremos realizar una consulta donde se visualice el número de departamento, el nombre y el número de empleados que tiene. Para hacer esta consulta combinamos las dos tablas:

```
SQL> SELECT D.DEPT_NO, DNOMBRE, COUNT(E.EMP_NO)
  2  FROM EMPLE E, DEPART D
  3  WHERE E.DEPT_NO = D.DEPT_NO
  4  GROUP BY D.DEPT_NO, DNOMBRE;

   DEPT_NO      DNOMBRE          COUNT(E.EMP_NO)
   -------      --------------   ---------------
        10      CONTABILIDAD                   3
        20      INVESTIGACION                  5
        30      VENTAS                         6
```

Observamos que con esta sentencia sólo aparecen los departamentos que tienen empleados. El departamento 40 se pierde. Para que aparezca este departamento, que no se corresponde con ninguna fila en la tabla EMPLE, usamos la combinación externa, para lo cual especificamos un **(+)** a continuación del nombre de la columna de la tabla en la que no aparece el departamento (la tabla EMPLE):

```
SQL>  SELECT D.DEPT_NO, DNOMBRE, COUNT(E.EMP_NO)
  2  FROM EMPLE E, DEPART D
  3  WHERE E.DEPT_NO (+) = D.DEPT_NO
  4  GROUP BY D.DEPT_NO, DNOMBRE;

   EPT_NO       DNOMBRE          COUNT(E.EMP_NO)
   -------      --------------   ---------------
        10      CONTABILIDAD                   3
        20      INVESTIGACION                  5
        30      VENTAS                         6
        40      PRODUCCION                     0
```

**Actividades propuestas**

**3** **Analiza lo que ocurre si en lugar de COUNT(E.EMP_NO) ponemos COUNT(*) en la sentencia SELECT anterior. Analiza también lo que ocurre si a la derecha de SELECT ponemos E.DEPT_NO en lugar de D.DEPT_NO.**

# 5.4 Operadores UNION, INTERSECT y MINUS

Los **operadores relacionales UNION, INTERSECT** y **MINUS** son *operadores de conjuntos*. Los **conjuntos** son las filas resultantes de cualquier sentencia SELECT válida que permiten combinar los resultados de varias SELECT para obtener un único resultado.

Supongamos que tenemos dos listas de centros de enseñanza de una ciudad y que queremos enviar a esos centros una serie de paquetes de libros.

Dependiendo de ciertas características de los centros, podemos enviar libros a todos los centros de ambas listas (UNION), a los centros que estén en las dos listas (INTERSECT) o a los que están en una lista y no están en la otra (MINUS). El formato de SELECT con estos operadores es el siguiente:

```
SELECT ... FROM ... WHERE...
Operador_de_conjunto
SELECT ... FROM ... WHERE...
```

## A. Operador UNION

El **operador UNION** combina los resultados de dos consultas. Las filas duplicadas que aparecen se reducen a una fila única. Éste es su formato:

```
SELECT COL1, COL2, ... FROM TABLA1 WHERE CONDICION
UNION
SELECT COL1, COL2, ... FROM TABLA2 WHERE CONDICION;
```

**Caso práctico**

**3 Disponemos de tres tablas:**

ALUM contiene los nombres de alumnos que se han matriculado este curso en el centro, NUEVOS contiene los nombres de los alumnos que han reservado plaza para el próximo curso y ANTIGUOS contiene los nombres de antiguos alumnos del centro. La descripción es la misma para las tres:

```
SQL> DESC ALUM

Nombre          ¿Nulo?   Tipo
----------      -------  ---------
NOMBRE                   VARCHAR2(20)
EDAD                     NUMBER(2)
LOCALIDAD                VARCHAR2(15)
```

*(Continúa)*

*(Continuación)*

Visualizamos el contenido de las tablas:

```
SQL> SELECT * FROM ALUM;
SQL> SELECT * FROM NUEVOS;
SQL> SELECT * FROM ANTIGUOS;
```

Visualiza los nombres de los alumnos actuales y de los futuros alumnos. Obtenemos los nombres de alumnos que aparezcan en las tablas ALUM y NUEVOS de la siguiente manera:

```
SQL> SELECT NOMBRE FROM ALUM UNION SELECT NOMBRE FROM NUEVOS;

NOMBRE
----------
ANA
ERNESTO
JUAN
LUISA
MAITE
MARÍA
PEDRO
RAQUEL
SOFÍA
9 filas seleccionadas.
```

**UNION ALL** combina los resultados de dos consultas. Cualquier duplicación de filas que se dé en el resultado final aparecerá en la consulta. En la consulta anterior, usando UNION ALL aparecerán nombres duplicados, como podemos observar en la Tabla 5.2:

```
SQL> SELECT NOMBRE FROM ALUM UNION ALL SELECT NOMBRE FROM
NUEVOS;
```

| NOMBRE |
|---|
| JUAN |
| PEDRO |
| ANA |
| LUISA |
| MARíA |
| ERNESTO |
| RAQUEL |
| JUAN |
| MAITE |
| SOFíA |
| ANA |
| ERNESTO |

12 filas seleccionadas.

**Tabla 5.2.** *Consulta con UNION ALL.*

## B. Operador INTERSECT

El **operador INTERSECT** devuelve las filas que son iguales en ambas consultas. Todas las filas duplicadas serán eliminadas antes de la generación del resultado final. Su formato es:

```
SELECT COL1, COL2, ... FROM TABLA1 WHERE CONDICION
INTERSECT
SELECT COL1, COL2, ... FROM TABLA2 WHERE CONDICION;
```

**Caso práctico**

**4 Obtén los nombres de alumnos que están actualmente en el centro y que estuvieron en el centro hace ya un tiempo.**

Necesitamos los nombres que están en la tabla ALUM y que, además, aparezcan en la tabla de ANTIGUOS alumnos:

```
SQL> SELECT NOMBRE FROM ALUM INTERSECT SELECT NOMBRE FROM ANTIGUOS;

NOMBRE
---------
ERNESTO
MARÍA
```

**Actividades propuestas**

**4 Esta consulta también se puede hacer usando el operador IN. Escribe la consulta anterior utilizando el operador IN.**

## C. Operador MINUS

El **operador MINUS** devuelve aquellas filas que están en la primera SELECT y no en la segunda. Las filas duplicadas del primer conjunto se reducirán a una fila única antes de que empiece la comparación con el otro conjunto. Su formato es éste:

```
SELECT COL1, COL2, ... FROM TABLA1 WHERE CONDICION
MINUS
SELECT COL1, COL2, ... FROM TABLA2 WHERE CONDICION;
```

**Caso práctico**

**5 Obtén los nombres y la localidad de alumnos que están actualmente en el centro y que nunca estuvieron anteriormente en él.**

Ordenamos la salida por LOCALIDAD. Necesitamos los nombres que están en la tabla ALUM y que, además, no aparezcan en la tabla de ANTIGUOS alumnos:

```
SQL> SELECT NOMBRE, LOCALIDAD FROM ALUM MINUS SELECT NOMBRE, LOCALIDAD FROM ANTIGUOS
ORDER BY LOCALIDAD;

NOMBRE     LOCALIDAD
---------- ----------
ANA        ALCALÁ
JUAN       COSLADA
PEDRO      COSLADA
RAQUEL     TOLEDO
LUISA      TORREJÓN
```

*(Continúa)*

*(Continuación)*

Esta consulta también se puede hacer usando el **operador NOT IN:**

```
SQL> SELECT NOMBRE, LOCALIDAD FROM ALUM WHERE NOMBRE NOT IN (SELECT NOMBRE FROM
ANTIGUOS) ORDER BY LOCALIDAD;
```

- Seleccionamos los nombres de la tabla ALUM que estén en NUEVOS y no en ANTIGUOS: `SQL> SELECT NOMBRE FROM ALUM INTERSECT SELECT NOMBRE FROM NUEVOS MINUS SELECT NOMBRE FROM ANTIGUOS;`

  Esta misma consulta se puede obtener con el **operador IN:** `SQL> SELECT NOMBRE FROM ALUM WHERE NOMBRE IN (SELECT NOMBRE FROM NUEVOS MINUS SELECT NOMBRE FROM ANTIGUOS);`

- Seleccionamos los nombres de la tabla ALUM que estén en NUEVOS o en ANTIGUOS: `SQL> SELECT NOMBRE FROM ALUM WHERE NOMBRE IN (SELECT NOMBRE FROM NUEVOS UNION SELECT NOMBRE FROM ANTIGUOS);`

  La consulta también se puede hacer sin usar UNION, pero recurriendo al **operador OR** (hemos de tener en cuenta que el uso de operadores de conjunto en lugar de IN, AND y OR es decisión del programador): `SQL> SELECT NOMBRE FROM ALUM WHERE NOMBRE IN (SELECT NOMBRE FROM NUEVOS) OR NOMBRE IN (SELECT NOMBRE FROM ANTIGUOS);`

**Actividades propuestas**

**5 Visualiza los nombres de los alumnos de la tabla ALUM que aparezcan en alguna de estas tablas: NUEVOS y ANTIGUOS.**

Escribe las distintas formas en que se puede poner la consulta anterior llegando al mismo resultado.

## D. Reglas para la utilización de operadores de conjuntos

Los operadores de conjuntos pueden encadenarse. Los conjuntos se evaluan de izquierda a derecha; para forzar precedencia se pueden utilizar paréntesis.

Estos operadores se pueden manejar con consultas de diferentes tablas, siempre que se apliquen las siguientes reglas:

- Las columnas de las dos consultas se relacionan en orden, de izquierda a derecha.
- Los nombres de columna de la primera sentencia SELECT no tienen por qué ser los mismos que los nombres de columna de la segunda.
- Las SELECT necesitan tener el mismo número de columnas.
- Los tipos de datos deben coincidir, aunque la longitud no tiene que ser la misma.

# Conceptos básicos

Para generar los resultados de una instrucción SELECT hay que seguir estos pasos:

1. **FROM:** Se evalúa la cláusula FROM para comprobar la tabla o tablas a usar.
2. **WHERE:** Si hay cláusula WHERE aplicar condición de búsqueda a cada fila.
3. **GROUP BY:** Agrupa estas filas seleccionadas con WHERE.
4. **HAVING:** Aplica la condición de búsqueda a los grupos de filas. Selecciona los grupos.
5. **SELECT:** Extrae las columnas de cada fila que queda.
6. **SELECT DISTINCT:** Elimina filas duplicadas.
7. **ORDER BY:** Clasifica los resultados.
8. Si la instrucción es una UNION, INTERSECT o MINUS se repiten los pasos para cada SELECT.

Aquí tienes un resumen del formato de la sentencia SELECT con las cláusulas vistas hasta ahora:

```
SELECT [ALL|DISTINCT] {expresión1, expresión2, ..., expresiónn | * }
FROM   {tabla1 [,tabla2, ...., tablan] }
[WHERE condición_de_búsqueda]
[GROUP BY expresión [,expresión ...]]
[HAVING condición_de_búsqueda]
[{UNION | INTERSECT | MINUS} SELECT ...]
[ORDER BY {expresión | posición_de_columna}
[DESC|ASC]
[, {expresión | posición_de_columna}
[DESC|ASC] ] ...];
```

Donde:

- Una expresión puede ser una constante, una referencia a la columna de una tabla o una expresión aritmética.
- Los corchetes ([]) indican un elemento opcional encerrado entre ellos.
- Las barras verticales (|) indican una elección entre dos o más elementos.
- Las llaves ({ }) indican una elección entre elementos requeridos.

# Actividades complementarias

**Tablas EMPLE Y DEPART**

1. Partiendo de la tabla EMPLE, visualiza por cada oficio de los empleados del departamento 'VENTAS' la suma de salarios.
2. Selecciona aquellos apellidos de la tabla EMPLE cuyo salario sea igual a la media del salario en su departamento.
3. A partir de la tabla EMPLE, visualiza el número de empleados de cada departamento cuyo oficio sea 'EMPLEADO'.
4. Desde la tabla EMPLE, visualiza el departamento que tenga más empleados cuyo oficio sea 'EMPLEADO'.
5. A partir de las tablas EMPLE y DEPART, visualiza el número de departamento y el nombre de departamento que tenga más empleados cuyo oficio sea 'EMPLEADO'.
6. Busca los departamentos que tienen más de dos personas trabajando en la misma profesión.

**Tablas ALUM, ANTIGUOS Y NUEVOS**

7. Visualiza los nombres de los alumnos de la tabla ALUM que aparezcan en estas dos tablas: ANTIGUOS y NUEVOS.
8. Escribe las distintas formas en que se puede poner la consulta anterior llegando al mismo resultado.
9. Visualiza aquellos nombres de la tabla ALUM que no estén en la tabla ANTIGUOS ni en la tabla NUEVOS.

**Tablas PERSONAL, PROFESORES Y CENTROS (hacer DESC de las tablas)**

10. Realiza una consulta en la que aparezca por cada centro y en cada especialidad el número de profesores. Si el centro no tiene profesores, debe aparecer un 0 en la columna de número de profesores. Las columnas a visualizar son: nombre de centro, especialidad y número de profesores.
11. Obtén por cada centro el número de empleados. Si el centro carece de empleados, ha de aparecer un 0 como número de empleados.
12. Obtén la especialidad con menos profesores.

**Tablas BANCOS, SUCURSALES, CUENTAS y MOVIMIENTOS (hacer DESC de las tablas)**

**TABLA BANCOS:** Contiene los datos de los bancos, una fila por cada banco. Un banco se identifica por el *COD_BANCO*.

**TABLA SUCURSALES:** Contiene los datos de las sucursales. Una fila por sucursal. Cada sucursal se identifica por el *COD_BANCO+COD_SUCUR*.

**TABLA CUENTAS:** Contiene los datos de las cuentas abiertas en las sucursales de los bancos. Una cuenta se identifica por las columnas *COD_BANCO+COD_SUCUR+NUM_CTA*. Contiene los saldos de las cuentas. SALDO_DEBE contiene la suma de Reintegros y SALDO_HABER la suma de Ingresos.

**TABLA MOVIMIENTOS:** Contiene los movimientos de las cuentas. Una fila representa un movimiento de una cuenta. La columna TIPO_MOV puede ser I (ingreso) o R (reintegro).

13. Obtén el banco con más sucursales. Los datos a obtener son:

| Nombre Banco | NºSucursales |
|---|---|
| xxxxxx | xx |

14. El saldo actual de los bancos de 'GUADALAJARA', 1 fila por cada banco:

| Nombre Banco | Saldo Debe | Saldo Haber |
|---|---|---|
| xxxxxx | xx,xx | xx,xx |

15. Datos de la cuenta o cuentas con más movimientos:

| Nombre Cta | Nºmovimientos |
|---|---|
| xxxxxx | xx |

16. El nombre de la sucursal que haya tenido más suma de reintegros:

| Nombre sucursal | Suma Reintegros |
|---|---|
| xxxxxx | xx,xx |