Introducción a SQL

por Matías Godoy
mattogodoy@gmail.com

Índice

Bases de datos relacionales

Antes de saber de que se trata SQL, debemos tener al menos una noción de lo que son las bases de datos relacionales.

Una base de datos contiene uno o mas objetos llamados **tablas**, que almacenan la información de dicha base de datos. Las tablas están definidas unívocamente por sus nombres y están compuestas por columnas y filas.
Las columnas contienen el nombre de columna, el tipo de datos que contiene, y otros atributos que describen la información que contendrá.
Las filas contienen los registros de la base de datos. Es decir, la información.

Una Base de Datos Relacional, es aquella que cumple con el modelo relacional, el cual nos permite establecer interconexiones (relaciones) entre los datos guardados en tablas, dándonos la posibilidad de mantener la información de forma ordenada y coherente.

Supongamos una base de datos que contiene información de empleados de una empresa, y está conformada por 3 tablas:

- *empleados*
- *localidades*
- *provincias*

Estas tablas contienen la información sobre los empleados de esta empresa, y el lugar en que trabaja cada uno de ellos.

El contenido de la tabla *empleados* es:

ID_Empleado	Nombre	Apellidos	ID_Localidad	Telefono	Direccion	Sueldo	Antiguedad
0	Jose	Blanco	3	656123456	Calle 1, 632	24000	3
1	María	Prado	5	695486632	Alcalá 23	26000	5
2	Antonio	Leiva	13	645634981	Pradillo 324	22000	1
3	Carmen	Alonso	3	946123656	Gran Via 15	24000	2

El contenido de la tabla *localidades* es:

ID_Localidad	Localidad	ID_Provincia
0	Benidorm	0
1	Calpe	0
2	Jávea	0
3	Barcelona	1
4	Sabadell	1
5	Sitges	1
6	Baza	2
7	Santa Fe	2
8	Colmenar Viejo	3
9	Fuenlabrada	3
10	Madrid	3
11	Las Rozas	3
12	Béjar	4
13	Toledo	5

El contenido de la tabla *provincias* es:

ID_Provincia	Provincia
0	Alicante
1	Barcelona
2	Granada
3	Madrid
4	Salamanca
5	Toledo

Como podemos ver, cada registro de cada tabla tiene su ID propio, el cual es único e irrepetible. De esta manera, estas tablas pueden relacionarse por medio de esta "clave".
Por ejemplo, en el registro con ID_localidad = 8 de la tabla *localidades* (Colmenar Viejo), podemos ver que tiene un campo ID_Provincia = 3. Si buscamos en la tabla *provincias*, encontraremos que este ID corresponde a la provincia de Madrid, y de esta manera sabremos que Colmenar Viejo es una localidad ubicada en la provincia de Madrid.

Estas relaciones pueden representarse de esta manera:

Como vemos en este caso, las relaciones son **1 a n**, es decir, **1** Provincia puede tener **n** Localidades, y **1** Localidad puede tener **n** empleados trabajando en ella. Las relaciones también podrán ser **n a n** (Varios a Varios) o **1 a 1**, dependiendo del caso.

¿Qué es SQL?

El lenguaje de consulta estructurados o SQL (por sus siglas en inglés Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite ejecutar diversos tipos de operaciones en ellas.

Gracias a la utilización del álgebra y de cálculos relacionales, el SQL brinda la posibilidad de realizar consultas con el objetivo de recuperar información de las bases de datos de manera sencilla.

Sentencias SQL

Una sentencia SQL es una frase escrita en inglés con la que indicamos lo que queremos obtener y de donde obtenerlo.
Todas las sentencias empiezan con un verbo (palabra reservada que indica la acción a realizar), seguido del resto de cláusulas, algunas obligatorias y otras opcionales que completan la frase.

Las sentencias SQL pertenecen a dos categorías principales: Lenguaje de Definición de Datos (DDL) y Lenguaje de Manipulación de Datos (DML). Estos dos lenguajes no son lenguajes en sí mismos, sino que es una forma de clasificar las sentencias de lenguaje SQL en función de su cometido. La diferencia principal reside en que el DDL crea objetos en la base de datos y sus efectos se pueden ver en el diccionario de la base de datos; mientras que el DML es el que permite consultar, insertar, modificar y eliminar la información almacenada en los objetos de la base de datos.

En este manual veremos sólo sentencias DML, dado que para comenzar, la definición de tablas y su estructura puede hacerse con herramientas basadas en interfaz gráfica.

SELECT

La sentencia SELECT es la mas utilizada y sirve para seleccionar información de la base de datos.

```
1   SELECT * FROM provincias
```

Esta sentencia nos devuelve todos los registros que contiene la tabla *provincias*.
Si quisiéramos obtener únicamente los nombres, apellidos y teléfonos de los empleados podríamos utilizar:

```
1   SELECT nombre, apellido, telefono FROM empleados
```

Esta sentencia nos devolvería:

Nombre	Apellidos	Telefono
Jose	Blanco	656123456
María	Prado	695486632
Antonio	Leiva	645634981
Carmen	Alonso	946123656

FROM

Utilizando la cláusula FROM especificamos las tablas de las cuales queremos obtener los registros.

WHERE

La cláusula WHERE sirve para filtrar resultados.
Supongamos que queremos saber el ID y el número de teléfono de los empleados que llevan mas de 3 años en la empresa:

```
1   SELECT id_empleado, telefono FROM empleados
2   WHERE antiguedad > 3
```

Esto devolvería:

ID_Empleado	Telefono
0	656123456
1	695486632

AND y OR

Son operadores para agregar condiciones a la búsqueda.
Si queremos un listado de empleados que se llamen Carmen y tengan un sueldo mayor a €21.000:

```
1   SELECT * FROM empleados
2   WHERE nombre = "Carmen" AND sueldo > 21000
```

Devolvería:

ID_Empleado	Nombre	Apellidos	ID_Localidad	Telefono	Direccion	Sueldo	Antiguedad
3	Carmen	Alonso	3	946123656	Gran Via 15	24000	2

Para el caso de empleados que tengan apellido Leiva o nombre Jose:

```
1   SELECT * FROM empleados
2   WHERE apellido = "Leiva" OR nombre = "Jose"
```

Devuelve:

ID_Empleado	Nombre	Apellidos	ID_Localidad	Telefono	Direccion	Sueldo	Antiguedad
0	Jose	Blanco	3	656123456	Calle 1, 632	24000	3
2	Antonio	Leiva	13	645634981	Pradillo 324	22000	1

Debemos tener en cuenta que en el caso del operador AND, nos traerá todos los registros que cumplan con todas las condiciones.
Para el caso de OR, nos traerá los registros que cumplan con al menos una de las condiciones.

ORDER BY

La cláusula ORDER BY se utiliza para ordenar los resultados por uno de los campos devueltos.
Pedimos un listado de todos los empleados ordenados por apellido, de mayor a menor:

```
1  SELECT * FROM empleados
2  ORDER BY apellido DESC
```

Resultado:

ID_Empleado	Nombre	Apellidos	ID_Localidad	Telefono	Direccion	Sueldo	Antiguedad
1	María	Prado	5	695486632	Alcalá 23	26000	5
2	Antonio	Leiva	13	645634981	Pradillo 324	22000	1
0	Jose	Blanco	3	656123456	Calle 1, 632	24000	3
3	Carmen	Alonso	3	946123656	Gran Via 15	24000	2

Por defecto, la cláusula ORDER BY nos ordena los resultados de forma ascendente, es decir, de menor a mayor. En el ejemplo anterior pedimos de manera específica que el orden sea inverso agregando DESC luego del nombre de la columna.

Podemos ordenar por varias columnas. Para ello, escribimos los nombres separados por comas en el orden de prioridades en que queremos ordenarlas :

```
1  SELECT * FROM empleados
2  ORDER BY sueldo, antiguedad
```

INSERT INTO

La sentencia INSERT INTO se utiliza para almacenar registros en la tabla indicada.

Supongamos que queremos agregar un nuevo empleado a la tabla *empleados* de la base de datos. Hay dos maneras de hacerlo.
En la primera, especificamos que campos del registro queremos agregar, y sus valores:

```
1  INSERT INTO empleados (nombre, apellidos, id_localidad, telefono, direccion, sueldo, antiguedad)
2  VALUES ("Javier", "Díaz", 2, 924928738, "Una direccion 534", 25000, 2)
```

De esta forma, nos aseguramos que los valores que asignamos se corresponden con los campos correctos.

La otra manera de hacerlo es especificando directamente los valores en el mismo orden de las columnas de la tabla:

```
1   INSERT INTO empleados
2   VALUES (null, "Javier", "Díaz", 2, 924928738, "Una direccion 534", 25000, 2)
```

Cabe destacar que el campo id_empleado de esta tabla es autoincremental. Es decir, que cada vez que se inserta un nuevo registro, el motor de base de datos asigna como valor el número del último registro + 1.
Es por ello que en el primer ejemplo no especificamos valores para el campo id_empleado, y en el segundo pasamos "null" como valor. En este último caso debemos pasar un valor nulo para mantener la estructura de columnas de la tabla. En ambos casos el motor de base de datos se encargará de asignarle un valor automáticamente.
Dado el caso, si lo necesitamos podemos pasar un valor para id_empleado, siempre y cuando sea un número entero y no exista otro registro con el mismo número.

UPDATE

Se utiliza para actualizar registros existentes en la base de datos.
Supongamos el caso en que quisiéramos cambiar el apellido y la antigüedad de uno del empleado con id_empleado = 2:

```
1   UPDATE empleados
2   SET nombre = 'Alberto', antiguedad = 3
3   WHERE id_empleado = 2
```

Resultado:

ID_Empleado	Nombre	Apellidos	ID_Localidad	Telefono	Direccion	Sueldo	Antiguedad
0	Jose	Blanco	3	656123456	Calle 1, 632	24000	3
1	María	Prado	5	695486632	Alcalá 23	26000	5
2	Alberto	Leiva	13	645634981	Pradillo 324	22000	3
3	Carmen	Alonso	3	946123656	Gran Via 15	24000	2

Esto nos permite actualizar cuantos campos necesitemos de una sola vez, y de manera simple.

DELETE

Nos permite eliminar registros de la base de datos.
Al igual que las cláusulas SELECT o UPDATE, nos permite efectuar esta operación sobre un grupo de registros que cumplan con la condición WHERE.
En este caso queremos eliminar de la tabla a todos los empleados que lleven menos de 3 años trabajando en la empresa:

```
1   DELETE FROM empleados
2   WHERE antiguedad < 3
```

Resultado:

ID_Empleado	Nombre	Apellidos	ID_Localidad	Telefono	Direccion	Sueldo	Antiguedad
0	Jose	Blanco	3	656123456	Calle 1, 632	24000	3
1	María	Prado	5	695486632	Alcalá 23	26000	5

Hay que tener gran cuidado con la utilización de este comando porque no requiere confirmación y no hay manera de deshacer los cambios.

Consultas avanzadas

Para consultas mas puntuales, SQL nos provee varias herramientas que nos permiten obtener los datos que buscamos con mas exactitud.

Veamos algunas de ellas.

LIMIT

Se utiliza para limitar el número de resultados a un valor especificado. Es muy útil para consultas en las que solo queremos una cantidad acotada de registros. Esto permite que la consulta se ejecute mas rápidamente.

Supongamos que queremos sólo los 5 primeros registros de la tabla *localidades*:

```
1   SELECT *
2   FROM localidades
3   LIMIT 5
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
0	Benidorm	0
1	Calpe	0
2	Jávea	0
3	Barcelona	1
4	Sabadell	1

Naturalmente esto puede combinarse con otras cláusulas como ORDER BY o WHERE.

LIKE

Nos permite obtener registros que contengan un patrón de búsqueda definido. Para ello usamos los siguientes comodines:

- **%** ➔ Substituto de cero o varios caracteres
- **_** ➔ Substituto de un caracter
- **[ ]** ➔ Rango de caracteres a encontrar
- **[!]** ➔ Rango de caracteres a evitar

Veamos algunos ejemplos:

Substituto de cero o varios caracteres

La siguiente sentencia encuentra todas las localidades cuyo nombre comienza con 'Ba':

```
1   SELECT *
2   FROM localidades
3   WHERE localidad LIKE 'Ba%'
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
3	Barcelona	1
6	Baza	2

La siguiente sentencia encuentra todas las localidades cuyo nombre contiene 'ad':

```
1  SELECT *
2  FROM localidades
3  WHERE localidad LIKE '%ad%'
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
4	Sabadell	1
9	Fuenlabrada	3
10	Madrid	3

Substituto de un caracter

La siguiente sentencia encuentra todas las localidades cuyo nombre comienza con cualquier caracter seguido de 'alpe':

```
1  SELECT *
2  FROM localidades
3  WHERE localidad LIKE '_alpe'
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
1	Calpe	0

La siguiente sentencia encuentra todas las localidades cuyo nombre comienza con 'S', seguido de cualquier caracter, seguido de 't', seguido de cualquier caracter, seguido de 'es':

```
1  SELECT *
2  FROM localidades
3  WHERE localidad LIKE 'S_t_es'
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
5	Sitges	1

Rango de caracteres a encontrar

La siguiente sentencia encuentra todas las localidades cuyo nombre comienza con 'C', 'S' o 'T':

```
1  SELECT *
2  FROM localidades
3  WHERE localidad LIKE '[CST]%'
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
1	Calpe	0
4	Sabadell	1
5	Sitges	1
7	Santa Fe	2
8	Colmenar Viejo	3
13	Toledo	5

La siguiente sentencia encuentra todas las localidades cuyo nombre comienza con 'A', 'B', 'C' o 'D':

```
SELECT *
FROM localidades
WHERE localidad LIKE '[A-D]%'
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
0	Benidorm	0
1	Calpe	0
3	Barcelona	1
6	Baza	2
8	Colmenar Viejo	3
12	Béjar	4

Rango de caracteres a evitar

La siguiente sentencia encuentra todas las localidades cuyo nombre NO comienza con 'S', 'B' o 'J':

```
SELECT *
FROM localidades
WHERE localidad LIKE '[!SBJ]%'
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
1	Calpe	0
8	Colmenar Viejo	3
9	Fuenlabrada	3
10	Madrid	3
11	Las Rozas	3
13	Toledo	5

IN

Este operador nos permite especificar múltiples valores en la cláusula WHERE.

Si queremos obtener todas las localidades cuyo id_provincia sea 2 o 5:

```
SELECT *
FROM localidades
WHERE id_provincia IN (2, 5)
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
6	Baza	2
7	Santa Fe	2
13	Toledo	5

BETWEEN

Selecciona valores dentro de un rango. Estos valores pueden ser fechas, números o texto.

Si queremos obtener todas las localidades que tengan id_localidad entre 4 y 10:

```
1  SELECT *
2  FROM localidades
3  WHERE id_localidad BETWEEN 4 AND 10
```

Resultado:

ID_Localidad	Localidad	ID_Provincia
4	Sabadell	1
5	Sitges	1
6	Baza	2
7	Santa Fe	2
8	Colmenar Viejo	3
9	Fuenlabrada	3
10	Madrid	3

Intersecciones

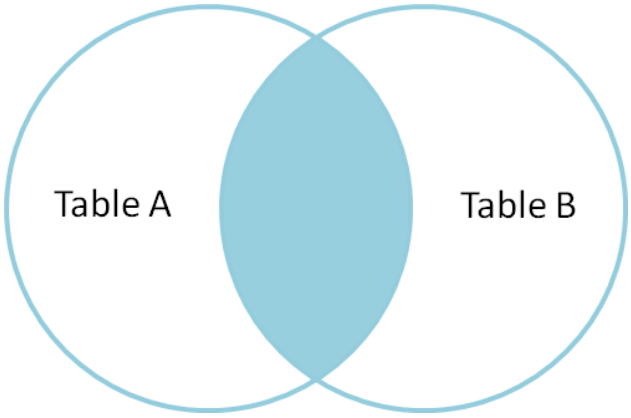
Cuando necesitamos obtener información que está distribuida en mas de una tabla, usamos intersecciones. La condición es que las tablas deben tener un campo en común para vincularlas. Para hacerlo utilizamos la cláusula JOIN en alguna de sus variantes.
Hay varios tipos de JOIN. Nosotros solo veremos los 3 mas importantes:

INNER JOIN

Es el tipo de JOIN mas común. Devuelve todos los registros de varias tablas donde el campo en común coincide.

Imaginemos las tablas de la base de datos como conjuntos. Cada tabla es un conjunto. Cada consulta devuelve un listado de registros que se almacenan en una tabla temporal, y por tanto, los resultados también son conjuntos.

En el caso del INNER JOIN, lo que conseguiremos es la intersección de registros entre los dos conjuntos (tablas), es decir, nos devolverá todos los registros que compartan el mismo valor en el campo especificado.



Supongamos que queremos saber el nombre, el apellido, el teléfono y el nombre de la localidad en que viven los empleados de nuestra empresa:

```
1  SELECT empleados.nombre, empleados.apellidos, empleados.telefono, localidades.localidad
2  FROM empleados
3  INNER JOIN localidades
4      ON empleados.id_localidad = localidades.id_localidad
```

Resultado:

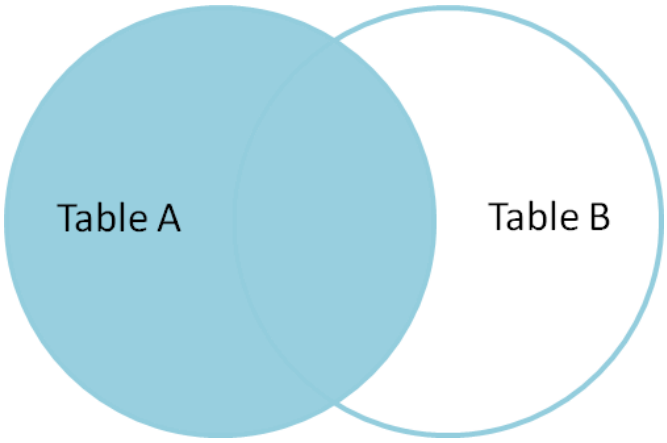
Nombre	Apellidos	Telefono	Localidad
Jose	Blanco	656123456	Barcelona
María	Prado	695486632	Sitges
Antonio	Leiva	645634981	Toledo
Carmen	Alonso	946123656	Barcelona

En la anterior consulta podemos notar varias cosas nuevas. A la hora de especificar los campos que queremos seleccionar, lo hacemos anteponiendo la tabla (tabla.campo). Esto se debe a que como ahora estamos trabajando con dos tablas distintas, tenemos que decirle al motor en cual de las tablas se encuentra el campo que queremos. Entonces en el caso de que las tablas tengan nombres iguales para algunas columnas, no habrá problemas porque el motor sabrá a que tabla nos referimos.
También podemos ver que para especificar cuál es el campo que comparten las dos tablas utilizamos la cláusula ON. Esto es lo que vincula nuestros registros.

Si hubiera algún empleado que no tenga asignado un id_localidad (es decir, que fuera NULL), no se mostraría en los resultados, dado que el INNER JOIN sólo nos devuelve los registros que se corresponden con otro en la segunda tabla.

LEFT JOIN

Nos devuelve todos los registros que existen en la tabla de la izquierda, y si tienen coincidencia en la tabla de la derecha la trae. De no tenerla, se rellenan los campos con valores nulos.



Siguiendo el ejemplo anterior, obtendremos todos los empleados, y si tienen asociada una localidad, la mostraremos:

```
1  SELECT empleados.nombre, empleados.apellidos, empleados.telefono, localidades.localidad
2  FROM empleados
3  LEFT JOIN localidades
4      ON empleados.id_localidad = localidades.id_localidad
```

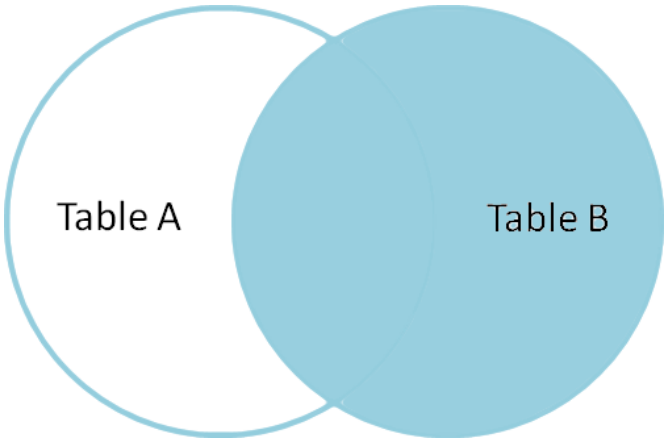
Resultado:

Nombre	Apellidos	Telefono	Localidad
Jose	Blanco	656123456	Barcelona
María	Prado	695486632	Sitges
Antonio	Leiva	645634981	Toledo
Carmen	Alonso	946123656	Barcelona

Notarán que el resultado es el mismo que haciendo INNER JOIN. Esto se debe a que todos los registros de la tabla de la izquierda (*empleados*) tienen asociado una localidad. En este caso, si hubiera un empleado que no tiene una localidad asociada, se mostraría igual, pero el campo 'localidad' sería NULL.
Podemos ver esto mejor representado en el siguiente caso:

RIGHT JOIN

Nos devuelve todos los registros que existen en la tabla de la derecha, y si tienen coincidencia en la tabla de la izquierda la trae. De no tenerla, se rellenan los campos con valores nulos.



Es exactamente igual que el LEFT JOIN, sólo que esta vez estamos interesados en la tabla de la derecha:

```
1  SELECT empleados.nombre, empleados.apellidos, empleados.telefono, localidades.localidad
2  FROM empleados
3  RIGHT JOIN localidades
4      ON empleados.id_localidad = localidades.id_localidad
```

Resultado:

Nombre	Apellidos	Telefono	Localidad
Jose	Blanco	656123456	Barcelona
María	Prado	695486632	Sitges
Antonio	Leiva	645634981	Toledo
Carmen	Alonso	946123656	Barcelona
NULL	NULL	NULL	Benidorm
NULL	NULL	NULL	Calpe
NULL	NULL	NULL	Jávea
NULL	NULL	NULL	Sabadell
NULL	NULL	NULL	Baza
NULL	NULL	NULL	Santa Fe
NULL	NULL	NULL	Colmenar Viejo
NULL	NULL	NULL	Fuenlabrada
NULL	NULL	NULL	Madrid
NULL	NULL	NULL	Las Rozas
NULL	NULL	NULL	Béjar

Esta vez los resultados sí que varían. Esto es porque nuestra consulta solicita todos los registros de la tabla de la derecha (*localidades*), y sus coincidencias con la tabla de la izquierda (*empleados*) en caso de haberlas.

Funciones

SQL tiene integradas algunas funciones que nos facilitan la obtención de los datos.
En este manual sólo veremos las mas utilizadas.

COUNT()

Devuelve la cantidad de registros que cumplen con las condiciones especificadas.

En el siguiente ejemplo queremos saber cuantos empleados tienen un sueldo igual o mayor que €24.000:

```
1  SELECT COUNT(*)
2  FROM empleados
3  WHERE sueldo >= 24000
```

Resultado:

COUNT(*)
3

AVG()

Devuelve el promedio de los valores de los campos numéricos seleccionados que cumplan con las condiciones especificadas.

En el siguiente ejemplo queremos saber el promedio de sueldos de la empresa:

```
1  SELECT AVG(sueldo)
2  FROM empleados
```

Resultado:

AVG(sueldo)
24000

SUM()

Devuelve la suma de los valores de los campos numéricos seleccionados que cumplan con las condiciones especificadas.

En el siguiente ejemplo queremos saber cuanto es lo que gasta la empresa en sueldos:

```
1  SELECT SUM(sueldo)
2  FROM empleados
```

Resultado:

SUM(sueldo)
96000

MAX()

Devuelve el registro que tenga el valor mas alto en la columna especificada.

En el siguiente ejemplo queremos saber el valor del sueldo mas alto pagado:

```
1  SELECT MAX(sueldo)
2  FROM empleados
```

Resultado:

MAX(sueldo)
26000

MIN()

Devuelve el registro que tenga el valor mas bajo en la columna especificada.

En el siguiente ejemplo queremos saber cual es la menor antigüedad entre los empleados:

```
1  SELECT MIN(antiguedad)
2  FROM empleados
```

Resultado:

MIN(antiguedad)
1

Ejercicios

1- Crear una base de datos que contenga las tres tablas citadas en este manual, e insertar en ellas los mismos valores.

2- Escribir una consulta que devuelva un listado de todos los empleados cuyo nombre conrenga la letra 'a'.

3- Escribir una consulta que devuelva un listado de todas las localidades que tengan un id_provincia mayor o igual que 4 o que tengan un id_localidad entre 2 y 5.

4- Escribir una consulta que devuelva el nombre y el apellido del empleado, el nombre de la localidad a la que pertenece, y el nombre de la provincia donde se encuentra, donde el sueldo del empleado sea superior a €23000. Ordenar los resultados por apellido.

5- Escribir una consulta que devuelva la cantidad de registros de la tabla *localidades*.