AWAKELAB

Módulo
Bases de Datos

Desarrollador de aplicaciones Full Stack

Python Trainee

Objetivo de la jornada

• Construye consultas utilizando sentencias SQL con condiciones de selección para resolver un problema planteado de selección condicional.
• Construye consultas utilizando sentencias SQL que requieren la consulta a varias tablas relacionadas a partir de un modelo de datos dado para resolver un
problema planteado de selección.
• Construye consultas utilizando sentencias SQL con funciones de agrupación para resolver un problema planteado que requiere la agrupación de datos

3.2.- Consultando información de una tabla

3.2.1.- El lenguaje estructurado de consultas SQL

SQL es un lenguaje de programación ampliamente utilizado que te permite definir y consultar bases de datos. Ya seas un analista de marketing, un periodista o un investigador que mapea neuronas en el cerebro de una mosca de la fruta, se beneficiará del uso de SQL para administrar objetos de bases de datos, así como para crear, modificar, explorar y resumir datos.

Dado que SQL es un lenguaje maduro que existe desde hace décadas, está profundamente arraigado en muchos sistemas modernos. Un par de investigadores de IBM describieron por primera vez la sintaxis de SQL (entonces llamado SEQUEL) en un artículo de 1974, basándose en el trabajo teórico del científico informático británico Edgar F. Codd. En 1979, una precursora de la empresa de bases de datos Oracle (entonces llamada Relational Software) se convirtió en la primera en utilizar el lenguaje en un producto comercial. Hoy en día, sigue siendo uno de los lenguajes informáticos más utilizados en el mundo y es poco probable que eso cambie pronto.

SQL viene en varias variantes, que generalmente están vinculadas a sistemas de bases de datos específicos. El Instituto Nacional Estadounidense de Estándares (ANSI) y la Organización Internacional de Normalización (ISO), que establecen estándares para productos y tecnologías, proporcionan estándares para el idioma y lo revisan. La buena noticia es que las variantes no se alejan mucho del estándar, por lo que una vez que aprenda las convenciones SQL para una base de datos, podrá transferir ese conocimiento a otros sistemas.

Entonces, ¿por qué debería usar SQL? Después de todo, SQL no suele ser la primera herramienta que las personas eligen cuando están aprendiendo a analizar datos. De hecho, muchas personas comienzan con hojas de cálculo de Microsoft Excel y su variedad de funciones analíticas. Después de trabajar con Excel, es posible que se pasen a Access, el sistema de base de datos integrado en Microsoft Office, que tiene una interfaz gráfica de consulta que facilita el trabajo, lo que hace que las habilidades de SQL sean opcionales.

Pero, como ya explicamos anteriormente en esta clase, Excel y Access tienen sus límites. Excel actualmente permite un máximo de 1.048.576 filas por hoja de trabajo, y Access limita el tamaño de la base de datos a dos gigabytes y limita las columnas a 255 por tabla. No es infrecuente que los conjuntos de datos superen esos límites, especialmente cuando se trabaja con datos descargados de sistemas gubernamentales. El último obstáculo que desea descubrir al enfrentarse a una fecha límite es que su sistema de base de datos no tiene la capacidad para realizar el trabajo.

El uso de un sólido sistema de base de datos SQL le permite trabajar con terabytes de datos, múltiples tablas relacionadas y miles de columnas. Le brinda un control programático mejorado sobre la estructura de sus datos, lo que genera eficiencia, velocidad y, lo más importante, precisión.

SQL también es un excelente complemento de los lenguajes de programación utilizados en ciencias de datos, como R y Python. Si usa cualquiera de esos lenguajes, por ejemplo, puede conectarse a bases de datos SQL y, en algunos casos, incluso incorporar la sintaxis SQL directamente en el lenguaje. Para las personas sin experiencia en lenguajes de programación, SQL a menudo sirve como una introducción fácil de entender a los conceptos relacionados con las estructuras de datos y la lógica de programación.

Además, conocer SQL puede ayudarle más allá del análisis de datos. Si profundiza en la creación de aplicaciones en línea, encontrará que las bases

de datos brindan el poder de backend para muchos frameworks web comunes, mapas interactivos y sistemas de administración de contenido. Cuando necesite excavar debajo de la superficie de estas aplicaciones, la capacidad de SQL para manipular datos y bases de datos será muy útil.

3.2.2. Recuperando información de una tabla

Antes de consultar datos desde una base, es la acción de crearla. Para esto sigue estos pasos:

- Ingresa a MySQL Workbench
- Haz clic en la conexión de usuario root, e ingresa la clave determinada anteriormente.
- Una vez dentro de la sesión, busca la pestaña “Schemas” en el menú del lado izquierdo.
- Presiona el botón secundario del mouse, y selecciona la opción “Create Schema”.
- En la ventana que aparecerá, ingresa el nombre “escuela” en el campo name, y selecciona un conjunto de caracteres (“charset”); para el idioma español te recomendamos el tipo “UTF-8”, el que contiene tildes y caracteres propios del idioma.
- En la misma ventana anterior, en la elección del subtipo (“collation”), debes seleccionar un tipo de juego de caracteres que se adapte a tus necesidades. Podrías escoger, por ejemplo, el tipo UTF8_general_ci.
- Realizado lo anterior, presiona el botón “Apply”, con lo que se creará el esquema. Este concepto de esquema se encuentra y es equivalente en la mayoría de los SGBDR (MySQL, Oracle, SQL Server, Postgres...). Por supuesto, el concepto de base de datos también, pero puede presentar diferencias significativas según el SGBD. En lo que respecta a MySQL, el término esquema es completamente intercambiable con el término base de datos y, a partir de ahora, se utilizará el término esquema para evitar la ambigüedad con el software MySQL bautizado a menudo como la ”Base de datos MySQL”.

El siguiente paso recomendado es crear un usuario que tenga acceso completo solo al esquema recién creado. Con esto se asegura mantener la integridad de los datos, y evitar pérdida de información sensible. Para esto se deben seguir estos pasos:

- Dentro de MySQL Workbench como root, debes seleccionar la pestaña “Administration”.
- Haz clic en la opción “Users and privileges”.
- En la ventana que se despliega a la derecha, presiona el botón “Add Account”.
- Por cada sección ingresa los siguientes datos:
    - Login: Ingresa nombre de usuario y clave (se pide dos veces)
    - Accounts limits: puedes limitar la acción del usuario. Por lo pronto no haremos cambios acá.
    - Administrative roles: puedes configurar roles de administración al usuario. Dado que queremos limitar los permisos del usuario, no alteramos esta sección.
    - Schema privileges: presiona el botón “Add Entry …”, y selecciona la base “escuela” creada anteriormente, y presiona “Ok”.
    - Se agregará un nuevo registro al listado; presiona el registro, y en la parte de abajo selecciona todos los permisos de la parte izquierda y central (permisos de objeto y de DDL). Una vez realizado presiona el botón “Apply”.
- Con esto el usuario estará creado en sistema, y podrás realizar acciones con este usuario sobre la base recién creada.

Para los ejemplos continuaremos utilizando la base de datos creada anteriormente; recuerda que debes ingresar desde ahora con el usuario creado recientemente, y seleccionar el esquema “escuela” como esquema por defecto (botón derecho sobre el esquema, y selecciona la opción “Set as Default Schema”). La sintaxis para crear una tabla en nuestra base de datos es la siguiente:

```
CREATE TABLE profesores (
   id int primary key auto_increment,
   nombre varchar(25),
   apellido varchar(50),
   escuela varchar(50),
   fecha_de_contratacion date,
   sueldo int
);
```

Ahora bien, para cada columna (id, nombre, apellido) utilizamos un tipo de datos específico. Dado que ya tenemos la tabla profesores, vamos a llenarla con algunos datos; para tal propósito utilizaremos la instrucción INSERT de SQL:

```
INSERT INTO profesores (
nombre, apellido, escuela, fecha_de_contratacion, sueldo)
VALUES
('Juanita', 'Perez', 'Gabriela Mistral', '2011-10-30', 234000),
('Bruce', 'Lee', 'Republica Popular China', '1993-05-22', 780945),
('Juan Alberto', 'Valdivieso', 'Sagrada Concepcion', '2005-08-01', 3400000),
('Pablo', 'Rojas', 'E-34', '2011-10-30', 300000),
('Nicolas', 'Echenique', 'Bendito Corazón de María', '2005-08-30', 8900000),
('Jericho', 'Jorquera', 'A-18 Abrazo de Maipu', '2010-10-22', 67500);
```

También, podemos insertar filas individuales o una a una:

```
INSERT INTO profesores (
nombre, apellido, escuela, fecha_de_contratacion, sueldo)
VALUES
('Caupolicán','Catrileo','Santiago de la extremadura',
'2000-10-26', 780000);
```

Observa que ciertos valores que estamos insertando están entre comillas simples, pero otros no. Este es un requisito estándar de SQL. El texto y las fechas requieren comillas; los números, incluidos los números enteros y decimales, no requieren comillas.
Note que no se utilizó cifra alguna para la columna id, que es la primera columna de la tabla. Esto sucede porque cuando creó la tabla, su secuencia de comandos especificó que esa columna fuera del tipo autoincremental. Entonces, cuando se inserta cada fila, automáticamente se llena la columna id con un número entero que se incrementa automáticamente como veremos a continuación cuando recuperemos (o leamos) la información de nuestra tabla profesores utilizando la sentencia SELECT de SQL. Una instrucción SELECT puede ser simple, recuperando todo en una sola tabla, o puede ser lo suficientemente compleja como para vincular docenas de tablas mientras maneja múltiples cálculos y filtra por criterios exactos.

Aquí hay una declaración SELECT que recupera cada fila y columna de la tabla llamada profesores:

```
SELECT * FROM profesores;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
4	Pablo	Rojas	E-34	2011-10-30	300000
5	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
6	Jericho	Jorquera	A-18 Abrazo de Maipu	2010-10-22	67500
7	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
NULL	NULL	NULL	NULL	NULL	NULL

Esta única línea de código muestra la forma más básica de una consulta SQL. El asterisco que sigue a la palabra clave SELECT es un comodín. Un comodín es como un sustituto de un valor: no representa nada en particular y, en cambio, representa todo lo que ese valor podría ser. Aquí, es la abreviatura de "seleccionar todas las columnas". Si hubiera dado un nombre de columna en lugar del comodín, este comando seleccionaría los valores en esa columna. La palabra clave FROM indica que desea que la consulta devuelva datos de una tabla en particular. El punto y coma después del nombre de la tabla le dice a PostgreSQL que es el final de la declaración de consulta.

También podemos consultar un subconjunto de columnas:

```
SELECT nombre, apellido FROM profesores;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

nombre	apellido
Juanita	Perez
Bruce	Lee
Juan Alberto	Valdivieso
Pablo	Rojas
Nicolas	Echenique
Jericho	Jorquera
Caupolicán	Catrileo

3.2.3.- Consultas utilizando la llave primaria

Una llave primaria o clave principal (primary key) es una columna o colección de columnas cuyos valores identifican de forma única cada fila de una tabla. Una columna de clave primaria válida impone ciertas restricciones:

- La columna o colección de columnas debe tener un valor único para cada fila.
- La columna o colección de columnas no puede tener valores faltantes.

En el caso que hemos visto hasta ahora con la tabla profesores, el campo id es una especie de llave primaria por defecto. Fíjate que cumple con las restricciones, pero podemos definir nuestras propias llaves primarias.

La llave primaria se puede definir al momento de creación de la tabla. En el primer campo de la tabla se acompaña con el término “primary key”.

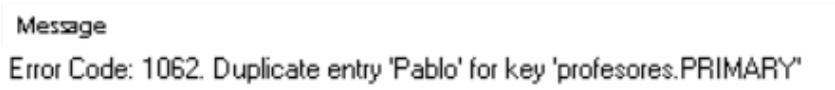
```
CREATE TABLE profesores (
   id int,
   nombre varchar(25),
   apellido varchar(50),
   escuela varchar(50),
   fecha_de_contratacion date,
   sueldo int,
   constraint profesores_pk primary key (nombre)
);
```

También la podemos agregar una vez que la tabla ya ha sido creada; para este caso definiremos el campo “nombre” como llave primaria:

```
alter table profesores add primary key (nombre);
```

Ahora bien, dado que nombre es nuestra llave primaria debe cumplir con las restricciones anteriormente descritas, es decir no podemos, por ejemplo, en este caso duplicar un nombre:

```
INSERT INTO profesores (nombre, apellido, escuela, fecha_de_contratacion, sueldo) VALUES ('Pablo', 'Lizarraga', 'Sagrado Grial de Montegrande', '2004-08-21', 6800000);
```



Naturalmente nuestro ejemplo es solo ilustrativo, los nombres de personas se repiten frecuentemente en una base de datos; no obstante, no hay dos personas que tengan el mismo número de cédula de identidad, tal condición es ideal para crear una llave primaria si su diseño y tabla lo requieren.

3.2.4.- Consultas utilizando condiciones de selección

Las consultas que requieren criterios de selección son naturalmente una práctica absolutamente necesaria y frecuente. Veremos algunas típicas, pero existen muchos criterios de selección a la hora de ejecutar estas consultas.

Anteriormente vimos la sintaxis básica para la instrucción SELECT

```
SELECT * FROM profesores;
```

Podemos elegir solo campos que nos interesan descartando los que no nos sirven para una consulta en particular. La regla general es

```
SELECT una_columna, otra_columna, super_columna FROM tabla;
```

Ejemplo 1:

```
SELECT apellido, escuela, sueldo FROM profesores;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

	apellido	escuela	sueldo
▶	Lee	Republica Popular China	780945
	Catrileo	Santiago de la extremadura	780000
	Jorquera	A-18 Abrazo de Maipu	67500
	Valdivieso	Sagrada Concepcion	3400000
	Perez	Gabriela Mistral	234000
	Echenique	Bendito Corazón de María	8900000
	Rojas	E-34	300000

Con esa sintaxis, la consulta recupera todas las filas de solo esas tres columnas. El orden lo podemos elegir de acuerdo a nuestras necesidades, por ejemplo:

```
SELECT sueldo, apellido, escuela FROM profesores;
```

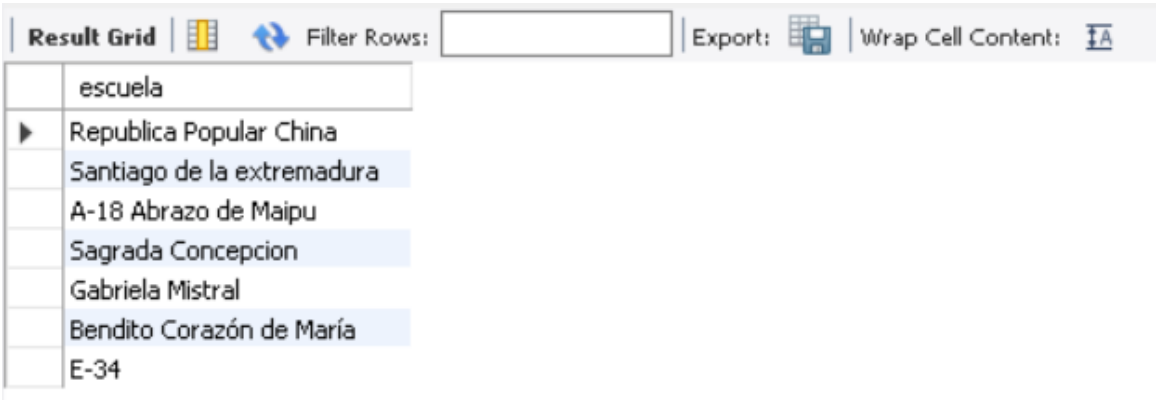
Aunque estos ejemplos son básicos, ilustran una buena estrategia para comenzar a consultar un conjunto de datos. Generalmente, es aconsejable comenzar su análisis verificando si sus datos están presentes y en el formato que espera.

Para este ejemplo primero insertemos una nueva fila con un apellido y una escuela repetida:

```
INSERT INTO profesores (nombre, apellido, escuela, fecha_de_contratacion, sueldo) VALUES ('Wong', 'Lee', 'Santiago de la extremadura', '2000-10-26', 780000);
```

Supongamos que necesitamos saber solo las escuelas que existen en nuestros registros, esto lo podemos lograr con **DISTINCT**:

```
SELECT DISTINCT escuela FROM profesores;
```

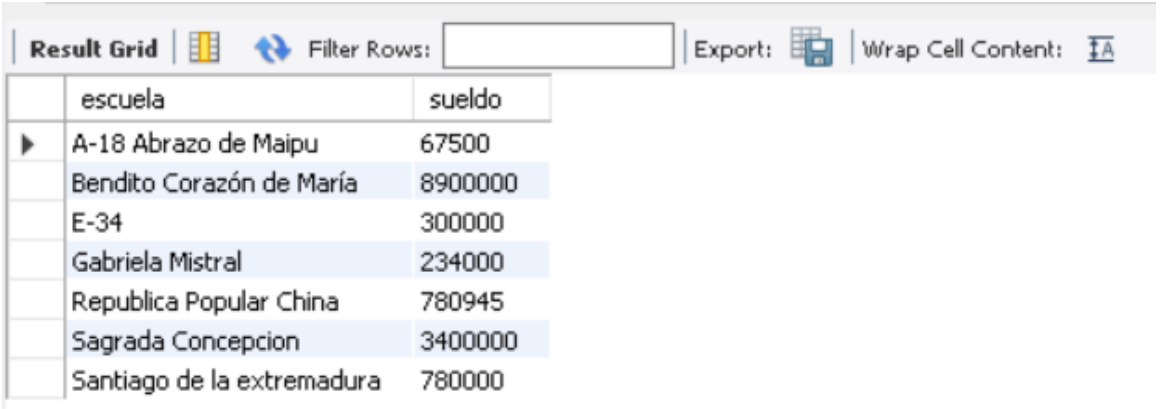


escuela
Republica Popular China
Santiago de la extremadura
A-18 Abrazo de Maipu
Sagrada Concepcion
Gabriela Mistral
Bendito Corazón de María
E-34

Pese a que los profesores Catrileo y Wong Lee trabajan en la misma escuela, solo necesitamos las escuelas y obviamente dejar de lado repeticiones innecesarias.

Podríamos ordenar la información para que nos resultará más legible, por ejemplo, alfabéticamente usando **ORDER BY** (y ascendente **ASC** o decreciente **DESC**):

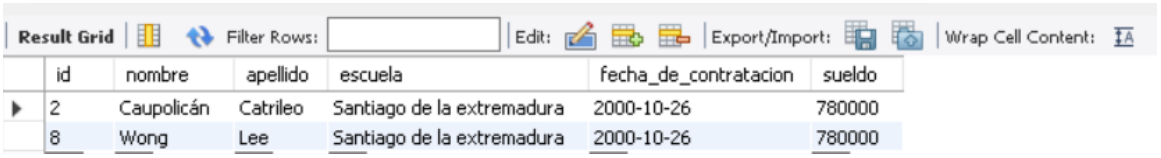
```
SELECT DISTINCT escuela, sueldo FROM profesores ORDER by escuela ASC;
```



escuela	sueldo
A-18 Abrazo de Maipu	67500
Bendito Corazón de María	8900000
E-34	300000
Gabriela Mistral	234000
Republica Popular China	780945
Sagrada Concepcion	3400000
Santiago de la extremadura	780000

Una construcción común en las consultas, son aquellas que usan la palabra clave WHERE (donde). Por ejemplo, si solo queremos saber los profesores que trabajan en cierta escuela podríamos ejecutar:

```
SELECT * FROM profesores WHERE escuela = 'Santiago de la extremadura';
```



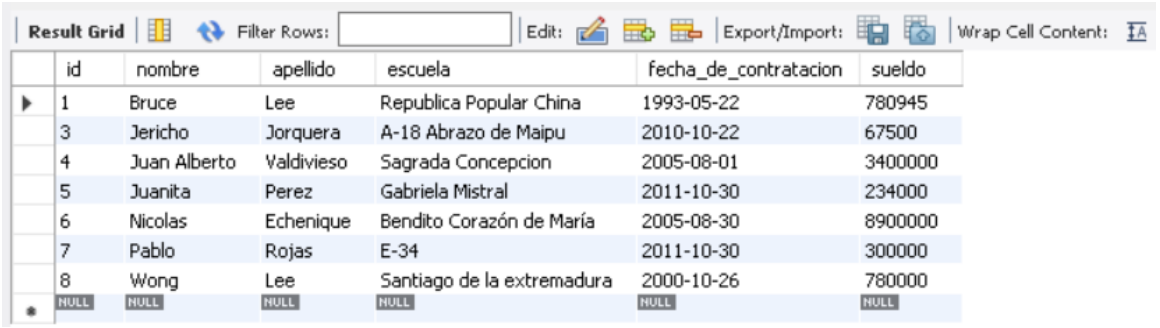
id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
2	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
8	Wong	Lee	Santiago de la extremadura	2000-10-26	780000

Existen muchos Operadores que se pueden usar en conjunto con WHERE

- = Igual a - ej. **WHERE escuela = 'E-34'**
- <> o != Distinto de - ej. **WHERE escuela <> 'E-34'**
- > Mayor a - ej. **WHERE sueldo > 100000**
- < Menor que - ej. **WHERE sueldo < 1000000**
- >= Mayor o igual que - ej. **WHERE sueldo >= 100000**
- <= Menor o igual que - ej. **WHERE sueldo <= 1000000**
- BETWEEN Dentro de un rango - ej. **WHERE sueldo BETWEEN 100000 AND 600000**
- IN Coincidir con un grupo de valores - ej. **WHERE apellido IN ('Catrileo', 'Perez')**
- LIKE Coincidir con un patrón (distingue mayúsculas y minúsculas) - ej. **WHERE apellido LIKE 'Catri%'**
- NOT Niega una condición - ej. **WHERE apellido NOT LIKE 'catri%'**

Para ilustrar vamos a ejecutar una de estas instrucciones, el resto se pueden ratificar de forma similar.

```
SELECT * FROM profesores WHERE apellido NOT LIKE 'catri%';
```



id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Bruce	Lee	Republica Popular China	1993-05-22	780945
3	Jericho	Jorquera	A-18 Abrazo de Maipu	2010-10-22	67500
4	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
5	Juanita	Perez	Gabriela Mistral	2011-10-30	234000
6	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
7	Pablo	Rojas	E-34	2011-10-30	300000
8	Wong	Lee	Santiago de la extremadura	2000-10-26	780000
NULL	NULL	NULL	NULL	NULL	NULL

3.2.5.- Utilización de funciones en las consultas

Al igual que con el número de operadores, el número de funciones integradas de MySQL es enorme. También vamos a explorar solo algunas de las más comunes, la documentación en línea tiene todo lo que puedas necesitar para tareas más específicas.

- **current_date:** Devuelve la fecha de hoy.
- **current_time:** Devuelve la hora actual (no se devuelve información de fecha).
- **current_timestamp:** Devuelve una marca de tiempo (fecha y hora) de la hora actual.
- **extract(tipo from campo):** Devuelve una parte de un campo especificado en el atributo "tipo".
- **now():** Devuelve un campo especificado en el texto de una marca de tiempo determinada.

Las funciones antes descritas se pueden probar fácilmente usando SELECT

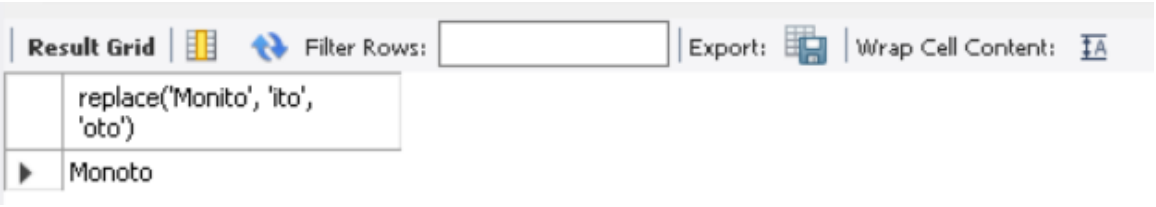
```
SELECT current_time;
```

Las funciones de cadena (string) se pueden usar para manipular valores de cadena de varias formas. Para estas funciones, cualquier entrada de cadena, incluidas las cadenas de texto, varchar y char, se considerará una entrada válida para la función.

- **lower(string):** Devuelve la cadena en minúsculas.
- **position(substring in string):** Devuelve la posición entera de una subcadena dentro de la cadena.
- **substring(string,from,[for]):** Extrae una subcadena de la cadena a partir de los dígitos especificados.
- **replace(string,from,to):** Reemplaza texto por texto en una cadena determinada.
- **upper(string):** Devuelve la cadena en mayúsculas.

Ejemplo 2:

```
SELECT replace('Monito', 'ito', 'oto');
```



3.2.6.- Consultas de selección con funciones de agrupación

SQL proporciona una serie de funciones que le permiten realizar recuentos, promedios y otras operaciones agregadas. Algunas de las funciones agregadas más comunes son:

- **avg(expression):** El promedio de todos los valores de entrada. Los valores de entrada deben ser uno de los tipos enteros.
- **count(*):** El número de valores de entrada.
- **count(expression):** El número de valores de entrada no nulos.
- **max(expression):** El valor máximo de expresión para todos los valores de entrada.
- **min(expression):** Los valores mínimos de expresión para todos los valores de entrada.
- **sum(expression):** La suma de la expresión para todos los valores de entrada no nulos.

Ejemplo 3:

Utilizando nuestra base datos de ejemplo de apartados anteriores, busquemos la cantidad de profesores que fueron contratados a partir del año 2010.

```
select
   count(DISTINCT escuela)
from profesores
WHERE fecha_de_contratacion > '2010-01-01';
```

Ejemplo 4:

O sumemos la cantidad de dinero mensual gastada en todos los profesores registrados

```
SELECT sum(sueldo) AS total FROM profesores;
```

3.2.7.- Consultando información relacionada en varias tablas

Continuando con nuestro camino para dominar los fundamentos de SQL, en secciones anteriores vimos lo básico de crear consultas con la instrucción SELECT. A continuación vamos a complicar un poco la cosa aprendiendo a realizar consultas en varias tablas de la base de datos al mismo tiempo.

Es habitual que queramos acceder a datos que se encuentran en más de una tabla y mostrar información mezclada de todas ellas como resultado de una consulta. Para ello tendremos que hacer combinaciones de columnas de tablas diferentes. En SQL es posible hacer esto especificando más de una tabla en la cláusula FROM de la instrucción SELECT.

Tenemos varias formas de obtener esta información. Una de ellas consiste en crear combinaciones que permiten mostrar columnas de diferentes tablas como si fuese una sola tabla, haciendo coincidir los valores de las columnas relacionadas.

Este último punto es muy importante, ya que si seleccionamos varias tablas y no hacemos coincidir los valores de las columnas relacionadas, obtendremos

una gran duplicidad de filas, realizándose el producto cartesiano entre las filas de las diferentes tablas seleccionadas.

Pero antes de entrar de lleno en este tema, analizaremos los modelos de datos, y su función en la creación de consultas entre varias tablas.

3.2.8.- Qué es un modelo de datos y cómo leerlo

Modelos de Datos

Durante las décadas de 1960 y 1970, los desarrolladores crearon bases de datos que resolvieron el problema de los grupos repetidos de varias formas diferentes. Estos métodos dan como resultado lo que se denominan modelos para sistemas de bases de datos. La investigación realizada en IBM proporcionó gran parte de la base para estos modelos, que todavía se utilizan en la actualidad.

Un factor principal en los primeros diseños de sistemas de bases de datos fue la eficiencia. Una de las formas comunes de hacer que los sistemas sean más eficientes era imponer una longitud fija para los registros de la base de datos, o al menos tener un número fijo de elementos por registro (columnas por fila). Esto esencialmente evita el problema del grupo que se repite. Si es un programador en casi cualquier lenguaje de procedimiento, verá fácilmente que, en este caso, puede leer cada registro de una base de datos en una estructura C simple. La vida real rara vez es tan complaciente, por lo que debemos encontrar formas de lidiar con datos estructurados de manera inconveniente. Los diseñadores de sistemas de bases de datos hicieron esto mediante la introducción de diferentes tipos de bases de datos.

- Modelo de base de datos jerárquica.
- Modelo de base de datos de red.
- Modelo de base de datos relacional.

Los primeros dos escapan al alcance de este curso y requerirían probablemente cursos aparte especializados. Nuestro foco será, como hasta acá, el modelo relacional.

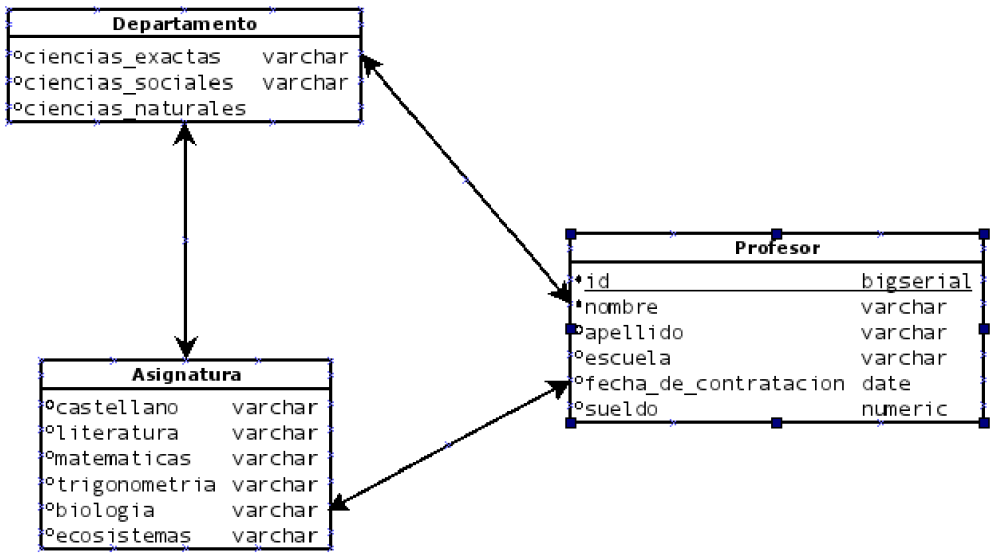
No es ningún misterio que una base de datos sea algo que almacena datos. Sin embargo, los modernos sistemas de administración de bases de datos relacionales (RDBMS) de hoy, como MySQL, PostgreSQL, SQL Server, Oracle, DB2 y otros, han ampliado esta función básica al agregar la capacidad de

almacenar y administrar datos relacionales. Este es un concepto que merece cierta atención.

Entonces, ¿qué significan los datos relacionales? Es fácil ver que cada dato escrito en una base de datos del mundo real está relacionado de alguna manera con información ya existente. Los productos están relacionados con categorías y departamentos; los pedidos están relacionados con productos y clientes, etc. Una base de datos relacional mantiene su información almacenada en tablas de datos, pero también es consciente de las relaciones entre las tablas.

Estas tablas relacionadas forman la base de datos relacional, que se convierte en un objeto con un significado propio, en lugar de ser simplemente un grupo de tablas de datos no relacionadas. Se dice que los datos se convierten en información solo cuando les damos significado, y establecer relaciones con otros datos es un medio ideal para hacerlo.

La figura muestra una representación simple de tres tablas de datos.



Cuando se dice que dos tablas están relacionadas, esto significa más específicamente que los registros de esas tablas están relacionados. Entonces, si la tabla de productos está relacionada con la tabla de asignatura, esto se traduce en que cada registro de profesores está relacionado de alguna manera con uno de los registros de la tabla de asignatura.

Los diagramas como este se utilizan para decidir qué se debe almacenar en la base de datos. Una vez que sepa qué almacenar, el siguiente paso es decidir

cómo se relacionan los datos enumerados, lo que conduce a la estructura física de la base de datos. El diagrama de la figura es solo un boceto que sirve para ir estructurando un modelo (que por supuesto puede sufrir modificaciones).

Entonces, ahora que conoce los datos que desea almacenar, pensemos en cómo se relacionan las tres partes entre sí. Además de saber que los registros de dos tablas están relacionados de alguna manera, también necesita saber el tipo de relación entre ellos. Veamos ahora más de cerca las diferentes formas en que se pueden relacionar dos tablas.

Datos relacionales y relaciones de tablas

Para continuar explorando el mundo de las bases de datos relacionales, analicemos más a fondo las tres tablas lógicas que hemos estado viendo hasta ahora. Para hacer la vida más fácil, démosles nombres ahora: la tabla que contiene profesores es profesor (necesitamos renombrarla); la tabla que contiene departamentos es departamento; y el último es asignatura. ¡No hay sorpresas aquí! Afortunadamente, estas tablas implementan los tipos más comunes de relaciones que existen entre tablas, las relaciones de uno a muchos y de muchos a muchos, por lo que tienes la oportunidad de aprender sobre ellas.

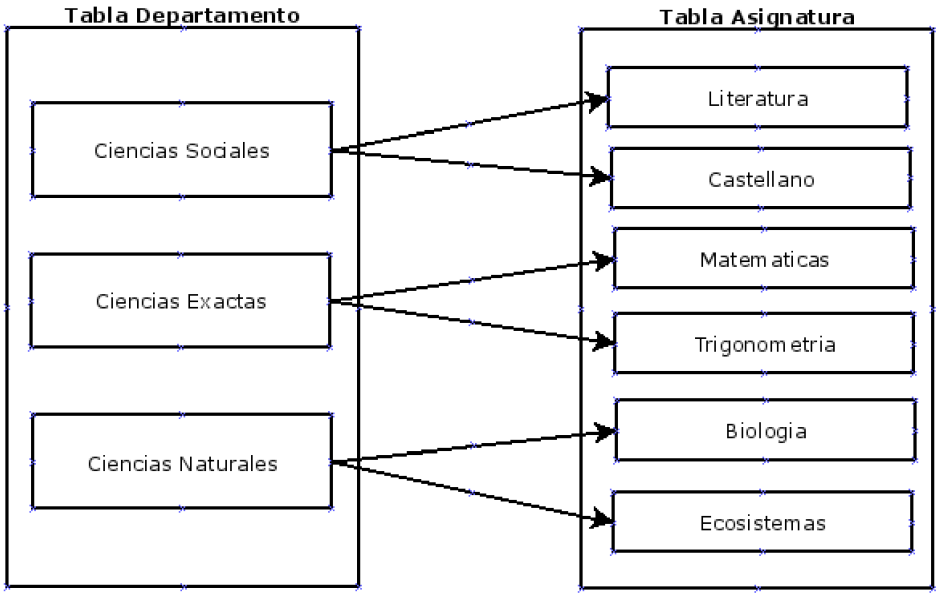
Renombremos nuestra tabla **profesores** a **profesor** para seguir las convenciones.

```
ALTER TABLE profesores RENAME TO profesor;
```

Nota: Existen algunas variaciones de estos dos tipos de relación que explicaremos, así como la relación uno a uno que es menos popular. En la relación uno a uno, cada fila de una tabla coincide exactamente con una fila de la otra. Por ejemplo, en una base de datos que permitiera que los pacientes fueran asignados a camas, ¡sería de esperar que hubiera una relación uno a uno entre los pacientes y las camas! Los sistemas de bases de datos no admiten la aplicación de este tipo de relación, porque tendría que agregar registros coincidentes en ambas tablas al mismo tiempo. Además, se pueden unir dos tablas con una relación uno a uno para formar una sola tabla.

Relaciones uno a muchos (one-to-many)

La relación de uno a varios ocurre cuando un registro de una tabla se puede asociar con varios registros de la tabla relacionada, pero no al revés. En nuestro caso, esto sucede para la relación departamento-asignatura. Un departamento específico puede contener cualquier número de categorías, pero cada categoría pertenece exactamente a un departamento. La figura siguiente representa mejor la relación de uno a varios entre departamentos y categorías.



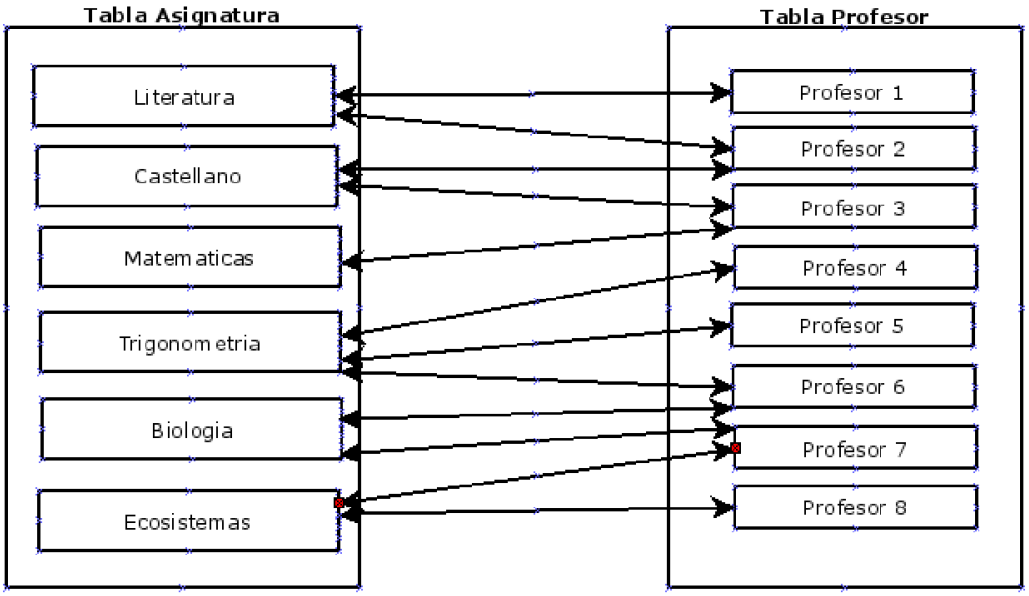
La relación de uno a muchos se implementa en la base de datos agregando una columna adicional en la tabla en el "lado muchos" de la relación, que hace referencia a la columna de ID de la tabla en el lado de la relación. En pocas palabras, en la tabla de asignaturas, tendrá una columna adicional (llamada **departamento_id**) que contendrá el ID del departamento al que pertenece la asignatura.

Relaciones de muchos a muchos (many-to-many)

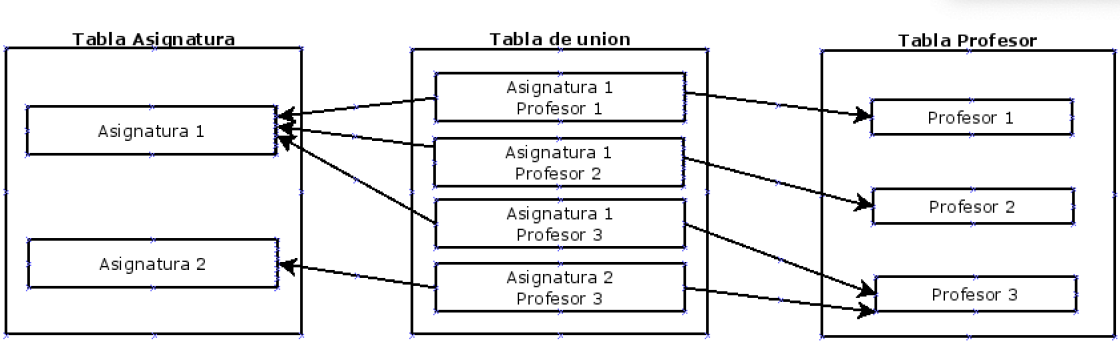
El otro tipo común de relación es la relación de muchos a muchos o varios a varios. Este tipo de relación se implementa cuando los registros en ambas tablas de la relación pueden tener múltiples registros coincidentes en la otra. En nuestro escenario, esto sucede para las tablas de profesores y asignaturas, porque sabemos que un profesor puede existir en más de una asignatura (un profesor con muchas asignaturas) y una asignatura puede tener más de un profesor (una asignatura con muchos profesores).

Esto sucede porque decidimos antes que un profesor podría estar en más de una asignatura. Si un profesor solo pudiera pertenecer a una asignatura, tendría otra relación de uno a varios, como la que existe entre departamentos y asignaturas (donde una asignatura no puede pertenecer a más de un departamento).

Representaremos esta relación con una imagen, al igual que como lo hicimos para la relación anterior:



Aunque lógicamente la relación de muchos a muchos ocurre entre dos tablas, las bases de datos no tienen los medios para implementar físicamente este tipo de relación usando solo dos tablas, por lo que hacemos trampa agregando una tercera tabla a la mezcla. Esta tercera tabla, llamada tabla de unión (también conocida como tabla de vinculación o tabla asociada) y dos relaciones de uno a muchos ayudarán a lograr la relación de muchos a muchos. La tabla de unión se utiliza para asociar profesores y asignaturas, sin restricciones sobre cuántos profesores pueden existir para una asignatura o cuántas asignaturas se pueden agregar a un profesor. La figura muestra el papel de la tabla de unión.



Tenga en cuenta que cada registro de la tabla de unión vincula una categoría con un profesor. Puede tener tantos registros como desee en la tabla de unión, vinculando cualquier asignatura a cualquier profesor. La tabla de unión contiene dos campos, cada uno de los cuales hace referencia a la llave primaria de una de las dos tablas vinculadas. En nuestro caso, la tabla de unión contendrá dos campos: un campo **asignatura_id** y un campo **profesor_id**.

Cada registro de la tabla de unión constará de un par de ID de profesor y asignatura (**profesor_id**, **asignatura_id**), que se utilizará para asociar un profesor en particular con una asignatura en particular. Al agregar más registros a la tabla **profesor_asignatura**, puede asociar un profesor con más asignaturas o una asignatura con más profesores, implementando eficazmente la relación de muchos a muchos.

Debido a que la relación de muchos a muchos se implementa utilizando una tercera tabla que hace la conexión entre las tablas vinculadas, no es necesario agregar campos adicionales a las tablas relacionadas de la forma en que agregamos el **departamento_id** a la tabla de asignatura para implementar la relación de uno a muchos.

No existe una convención de nomenclatura definitiva para usar en la tabla de unión. La mayoría de las veces está bien unir los nombres de las dos tablas vinculadas; en este caso, la tabla de unión se llama asignatura_profesor.

Aplicación de relaciones de tabla mediante llaves externas

Las relaciones entre tablas se pueden hacer cumplir físicamente en la base de datos utilizando restricciones **FOREIGN KEY**, o simplemente llaves externas o claves foráneas.

Ya aprendimos acerca de la restricción **PRIMARY KEY** o llave principal. Las llaves externas, por otro lado, ocurren entre dos tablas: la tabla en la que se define la llave externa (la tabla de referencia) y la tabla a la que hace referencia la clave externa (la tabla referenciada).

Una llave externa es una columna o combinación de columnas que se utiliza para imponer un vínculo entre los datos de dos tablas (generalmente representa una relación de uno a varios). Las llaves externas se utilizan como método para garantizar la integridad de los datos y para establecer una relación entre tablas.
Para hacer cumplir la integridad de la base de datos, las llaves externas, aplican ciertas restricciones. A diferencia de las restricciones **PRIMARY KEY** y **UNIQUE** que aplican restricciones a una sola tabla, la restricción **FOREIGN KEY** aplica restricciones tanto en las tablas de referencia como en las referenciadas. Por ejemplo, si aplica la relación de uno a varios entre las tablas de departamento y de categoría mediante una restricción FOREIGN KEY, la base de datos incluirá esta relación como parte de su integridad. No le permitirá agregar una categoría a un departamento inexistente, ni le permitirá eliminar un departamento si hay categorías que pertenecen a él.

Creación y llenado de nuevas tablas de datos

Ya renombramos la tabla profesores como **profesor** para seguir las convenciones (no usar plurales). Es hora de completar el resto de nuestras tablas.

```
CREATE TABLE departamento (
departamento_id int,
nombre VARCHAR(100),
descripcion VARCHAR(1000),
PRIMARY KEY (departamento_id)
);

INSERT INTO departamento (departamento_id, nombre, descripcion)
VALUES
(1, 'Ciencias Sociales', 'Ramas de la ciencia relacionadas con la sociedad y el comportamiento humano.'),
(2, 'Ciencias Exactas', 'Disciplinas que se basan en la observación y experimentación para crear conocimientos y cuyos contenidos pueden sistematizarse a partir del lenguaje matemático.'),
```

```
(3, 'Ciencias Naturales', 'Ciencias que tienen por objeto el estudio de la naturaleza, siguiendo la modalidad del método científico conocida como método empírico-analítico.');

CREATE TABLE asignatura
(
asignatura_id int NOT NULL,
departamento_id int NOT NULL,
nombre VARCHAR(100) NOT NULL,
descripcion VARCHAR(1000),
PRIMARY KEY (asignatura_id)
);
```

Insertamos registros en la tabla de asignaturas:

```
INSERT INTO asignatura (
asignatura_id, departamento_id, nombre, descripcion)
VALUES
(1, 1, 'Literatura', 'Arte de la expresión verbal'),
(2, 1, 'Castellano', 'Lengua romance procedente del latín hablado'),
(3, 2, 'Matemáticas', 'Ciencia formal que, partiendo de axiomas y siguiendo el razonamiento lógico, estudia las propiedades y relaciones entre entidades abstractas como números, figuras geométricas, iconos, glifos, o símbolos en general'),
(4, 2, 'Trigonometría', 'Rama de la matemática, cuyo significado etimológico es la medición de los triángulos'),
(5, 3, 'Biología', 'Rama de la ciencia que estudia los procesos naturales de los organismos vivos, considerando su anatomía, fisiología, evolución, desarrollo, distribución y relaciones'),
(6, 3, 'Ecosistema', 'Sistema biológico constituido por una comunidad de organismos vivos (biocenosis) y el medio físico donde se relacionan (biotopo)');
```

Si por alguna razón desea borrar alguna tabla, por ejemplo, con el fin de crearla de nuevo para experimentar, puede ejecutar:

```
DROP TABLE <nombre de la tabla>;
```

Ahora llenaremos algunas filas de nuestra tabla asignatura:

```
CREATE TABLE profesor_asignatura (
profesor_id INT NOT NULL,
asignatura_id INT NOT NULL,
PRIMARY KEY (profesor_id, asignatura_id)
);

INSERT INTO profesor_asignatura (profesor_id, asignatura_id) VALUES (1, 1), (1, 2), (2, 3), (2, 4), (3, 5), (3, 6), (4, 6), (5, 4), (6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (6, 6), (7, 3), (7, 4), (8, 1);
```

Ahora realicemos cambios en nuestra tabla profesor para continuar con las convenciones de nombres que hemos adoptado.

```
ALTER TABLE profesor RENAME COLUMN id TO profesor_id;
```

Agregamos la llave primaria

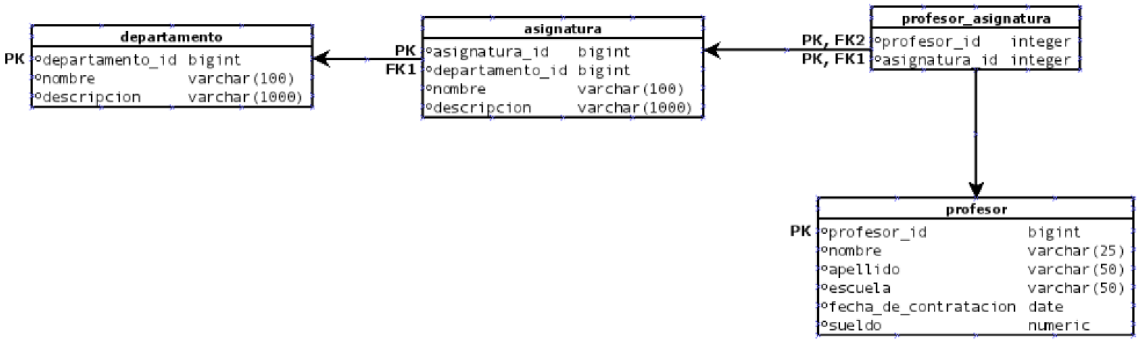
```
ALTER TABLE profesor ADD PRIMARY KEY (profesor_id);
```

3.2.9.- Consultas de selección con tablas relacionadas

Ahora tenemos un esquema de tablas que podemos relacionar y consultar según nuestras necesidades, por ejemplo, si queremos saber a qué departamento pertenece la asignatura 'Castellano'.

```
SELECT departamento.nombre FROM departamento
INNER JOIN asignatura
ON asignatura.departamento_id = departamento.departamento_id
WHERE asignatura.nombre = 'Castellano';
```

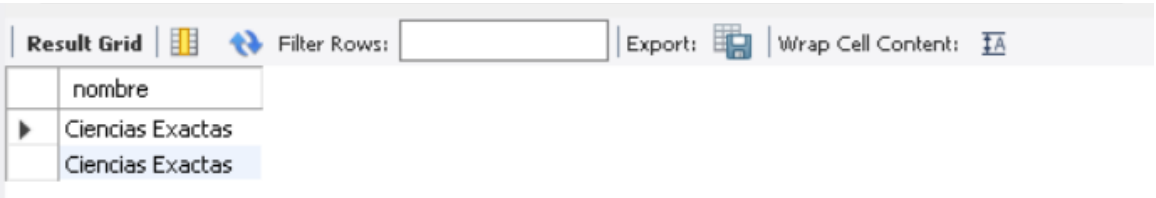
Esta consulta entrega lo siguiente:



O si queremos saber en qué departamento(s) trabaja el profesor Pablo Rojas

```
SELECT d.nombre
FROM departamento d
INNER JOIN asignatura a
ON a.departamento_id = d.departamento_id
INNER JOIN profesor_asignatura pa
ON pa.asignatura_id = a.asignatura_id
INNER JOIN profesor p
ON p.profesor_id = pa.profesor_id
WHERE p.nombre = 'Pablo' AND p.apellido = 'Rojas';
```

El resultado obtenido será:



Puedes comprobar el resultado siguiendo las queries y las tablas de forma visual/manual.

3.2.10.- Integridad referencial

La integridad referencial se refiere a la precisión y consistencia de los datos dentro de una relación.

En las relaciones, los datos están vinculados entre dos o más tablas. Esto se logra haciendo que la llave externa (en la tabla asociada) haga referencia a un valor de llave principal (en la tabla principal). Debido a esto, debemos asegurarnos de que los datos de ambos lados de la relación permanezcan intactos.

Por lo tanto, la integridad referencial requiere que, siempre que se use un valor de llave externa, debe hacer referencia a una llave primaria válida existente en la tabla principal.

En nuestras tablas no hemos observado que, por ejemplo, exista una asignatura que haga referencia al departamento_id = 4, solo tenemos 3 departamentos: Ciencias Sociales, Ciencias Exactas y Ciencias Naturales. Referenciar un departamento con id = 4 generaría un récord huérfano.

La falta de integridad referencial en una base de datos puede llevar a que se devuelvan datos incompletos, normalmente sin indicación de error. Esto podría resultar en la "pérdida" de registros en la base de datos, porque nunca se devuelven en consultas o informes.

También podría dar lugar a que aparezcan resultados extraños en informes (como productos sin una empresa asociada). O peor aún, podría resultar en que los clientes no reciban los productos por los que pagaron o que un paciente del hospital no reciba el tratamiento correcto, o un equipo de socorro en casos de desastre que no reciba los suministros o la información correctos.

3.2.11.- Queries anidadas

Una subconsulta a veces se anida dentro de otra consulta. Normalmente, se utiliza para un cálculo o una prueba lógica que proporciona un valor o un conjunto de datos que se pasarán a la parte principal de la consulta. Su sintaxis no es inusual: simplemente encerramos la subconsulta entre paréntesis y la usamos donde sea necesario. Por ejemplo, podemos escribir una subconsulta que devuelve varias filas y trata los resultados como una tabla en la cláusula FROM de la consulta principal. O podemos crear una subconsulta escalar que devuelva un solo valor y usarlo como parte de una

expresión para filtrar filas a través de las cláusulas WHERE, IN y HAVING. Esos son los usos más comunes de las subconsultas.

Como ejemplo una vez más buscaremos en qué departamento(s) trabaja el profesor Pablo Rojas:

```
SELECT d.nombre
FROM departamento d
WHERE d.departamento_id IN (
SELECT departamento_id FROM asignatura a
WHERE a.asignatura_id IN (
SELECT asignatura_id FROM profesor_asignatura pa
WHERE pa.profesor_id IN (
SELECT p.profesor_id FROM profesor p
WHERE p.nombre = 'Pablo' AND p.apellido = 'Rojas'
)
)
);
```

El resultado obtenido es:

3.2.12.- Queries con distintos tipos de JOIN (INNER, LEFT, RIGHT)

En lo que se refiere a **JOIN(s)**, hasta ahora hemos utilizado **INNER JOIN** en nuestras consultas. Ahora veremos las distintas opciones de esta cláusula.

Hay más de una forma de unir tablas en SQL, y el tipo de unión que usará depende de cómo desee recuperar los datos. La siguiente lista describe los diferentes tipos de combinaciones. Al revisar cada uno, es útil pensar en dos tablas una al lado de la otra, una a la izquierda de la palabra clave JOIN y la otra a la derecha. A continuación de la lista, se muestra un ejemplo basado en datos de cada combinación:

- **JOIN** Devuelve filas de ambas tablas donde se encuentran valores coincidentes en las columnas unidas de ambas tablas. La sintaxis alternativa es **INNER JOIN**.
- **LEFT JOIN** Devuelve todas las filas de la tabla de la izquierda más las filas que coinciden con los valores de la columna unida de la tabla de la derecha. Cuando una fila de la tabla de la izquierda no tiene una coincidencia en la tabla de la derecha, el resultado no muestra valores de la tabla de la derecha.
- **RIGHT JOIN** Devuelve todas las filas de la tabla derecha más las filas que coinciden con los valores clave en la columna clave de la tabla izquierda. Cuando una fila de la tabla de la derecha no tiene una coincidencia en la tabla de la izquierda, el resultado no muestra valores de la tabla de la izquierda.
- **FULL JOIN** Devuelve cada fila de ambas tablas y coincide con las filas; luego une las filas donde coinciden los valores de las columnas unidas. Si no hay ninguna coincidencia para un valor en la tabla izquierda o derecha, el resultado de la consulta contiene una fila vacía para la otra tabla. En MySQL no existe este comando, no así en otros motores de bases de datos.

Para explicar de forma más clara vamos a "ensuciar" un poco nuestra base de datos e insertar una fila adicional y sin relaciones en la tabla departamento y otra en la tabla asignatura (violando la integridad referencial solo para ejemplificar).

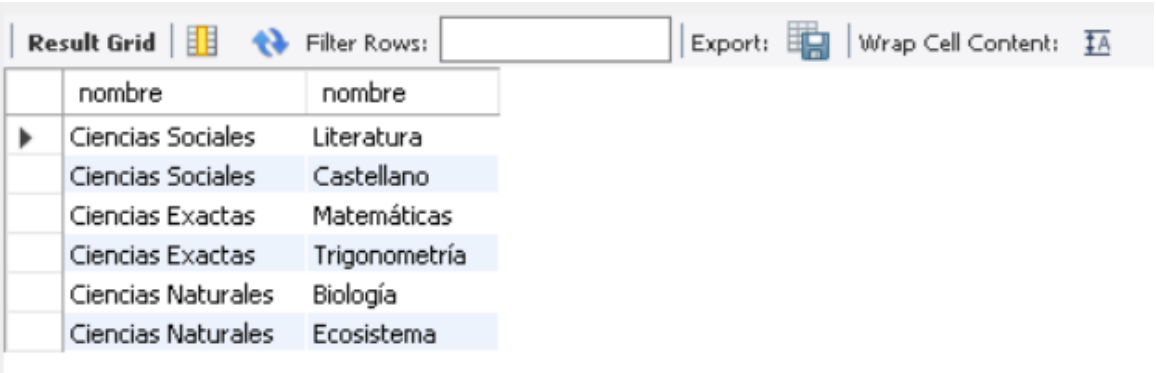
```
INSERT INTO departamento (departamento_id, nombre, descripcion)
VALUES (4, 'Ciencias Especiales', 'Ramas de la ciencia que son especiales.');

INSERT INTO asignatura (asignatura_id, departamento_id, nombre, descripcion) VALUES (7, 10, 'Especial', 'Asignatura Especial');
```

Usaremos un par de tablas de nuestra base de datos para demostrar alternativas.

Hasta ahora hemos usado **JOIN (INNER JOIN)**; este tipo puede ser pensado como la "Intersección" de ambas tablas. Ejemplo para recordar que asignaturas corresponden a que departamento podemos ejecutar:

```
SELECT d.nombre, a.nombre FROM
```

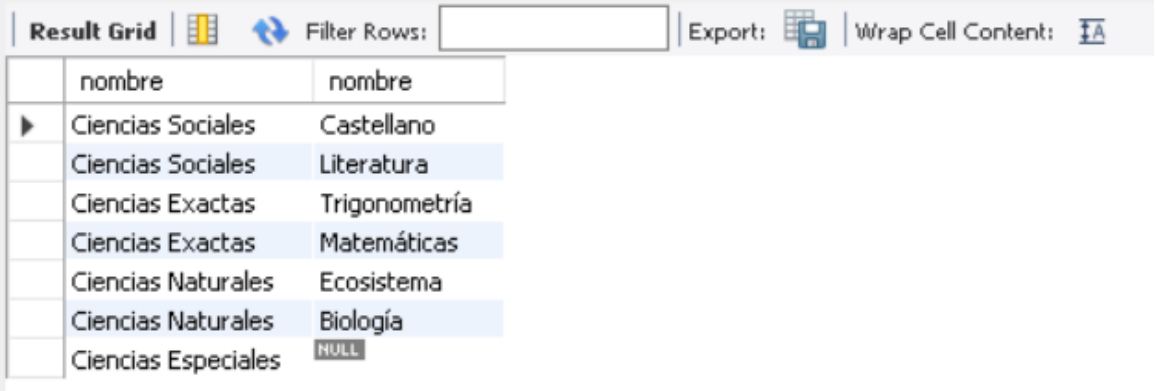
```
departamento d JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```



nombre	nombre
Ciencias Sociales	Literatura
Ciencias Sociales	Castellano
Ciencias Exactas	Matemáticas
Ciencias Exactas	Trigonometría
Ciencias Naturales	Biología
Ciencias Naturales	Ecosistema

LEFT JOIN y RIGHT JOIN: A diferencia de **JOIN**, las palabras clave **LEFT JOIN** y **RIGHT JOIN** devuelven todas las filas de una tabla y muestran filas en blanco de la otra tabla si no se encuentran valores coincidentes en las columnas unidas. Primero veamos **LEFT JOIN** en acción, esta puede ser pensada como la "Intersección más lo que está a la izquierda".

```
SELECT d.nombre, a.nombre FROM
departamento d LEFT JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```



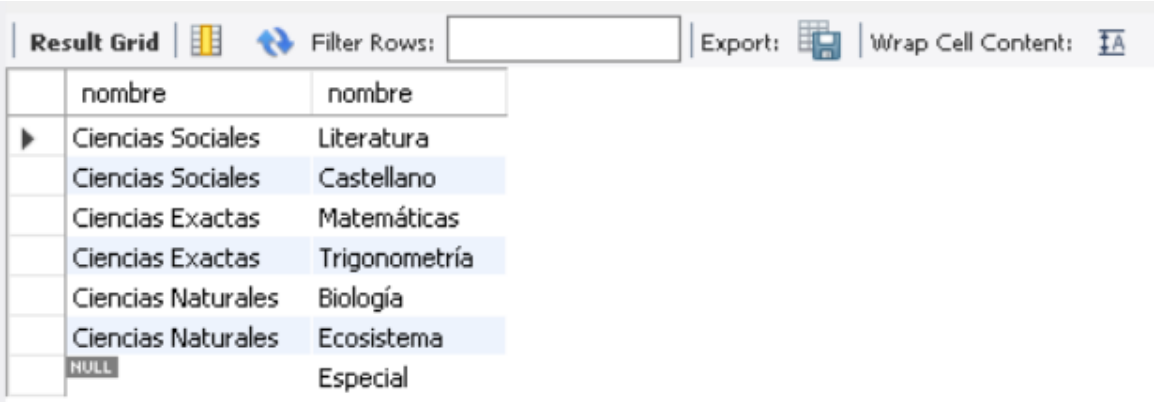
nombre	nombre
Ciencias Sociales	Castellano
Ciencias Sociales	Literatura
Ciencias Exactas	Trigonometría
Ciencias Exactas	Matemáticas
Ciencias Naturales	Ecosistema
Ciencias Naturales	Biología
Ciencias Especiales	NULL

Nota: Vemos que el departamento dummy Ciencias Especiales también aparece en la consulta.

RIGHT JOIN es similar solo que este puede ser pensado como la "Intersección más lo que está a la derecha".

```
SELECT d.nombre, a.nombre FROM
departamento d RIGHT JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```

Obtenemos:

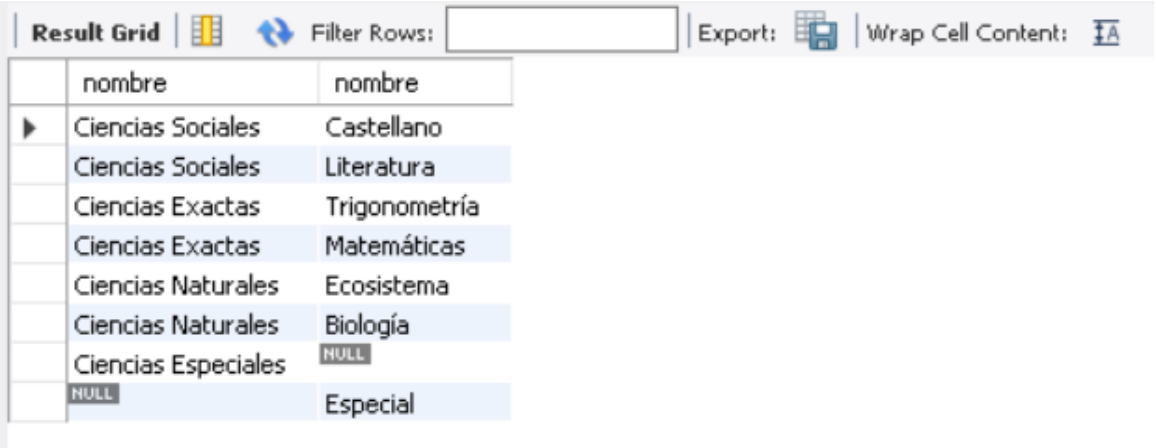


nombre	nombre
Ciencias Sociales	Literatura
Ciencias Sociales	Castellano
Ciencias Exactas	Matemáticas
Ciencias Exactas	Trigonometría
Ciencias Naturales	Biología
Ciencias Naturales	Ecosistema
NULL	Especial

Finalmente, tal como se dijo antes, FULL OUTER JOIN no existe en el motor de bases de datos en estudio. Si se desea simular en una consulta, se puede usar un comando UNION que junte el resultado de un LEFT JOIN con el resultado de RIGHT JOIN.

```
SELECT d.nombre, a.nombre FROM
departamento d LEFT JOIN asignatura a
ON d.departamento_id = a.departamento_id
UNION
SELECT d.nombre, a.nombre FROM
departamento d RIGHT JOIN asignatura a
ON d.departamento_id = a.departamento_id;
```

Con la consulta anterior se obtiene lo siguiente:



nombre	nombre
Ciencias Sociales	Castellano
Ciencias Sociales	Literatura
Ciencias Exactas	Trigonometría
Ciencias Exactas	Matemáticas
Ciencias Naturales	Ecosistema
Ciencias Naturales	Biología
Ciencias Especiales	NULL
NULL	Especial

3.2.13.- Procedimientos almacenados

Un procedimiento almacenado MySQL no es más que una porción de código que puedes guardar y reutilizar. Es útil cuando repites la misma tarea repetidas veces, siendo un buen método para encapsular el código. Al igual que ocurre con las funciones, también puede aceptar datos como parámetros, de modo que actúa en base a éstos.

Para poder crear un procedimiento almacenado es necesario que tengas permisos INSERT y DELETE sobre la base de datos.

Los procedimientos almacenados suelen ser confundidos con las funciones almacenadas, pero son dos conceptos distintos. Por ejemplo, los procedimientos deben ser invocados con la sentencia EXEC, mientras es posible utilizar cualquier función almacenada directamente en una sentencia SQL.

Los procedimientos pueden ser utilizados para una infinidad de tareas, permitiendo organizar mejor el código y preservar una integridad de datos óptima. El uso de procedimientos también suele suponer una mejora de rendimiento en tareas relativamente complejas. Al igual que ocurre con las funcionales almacenadas, no nos debemos olvidar de que el uso de procedimientos almacenados mejorará también la legibilidad del código.

Sintaxis

La sintaxis de un procedimiento almacenado es la siguiente:

```
CREATE PROCEDURE nombre_procedimiento
AS
sentencias_sql
GO;
```

Para ejecutar un procedimiento almacenado lo invocamos así:

```
EXEC nombre_procedimiento (param1, param2, ....);
```

Parámetros

Como has visto, los parámetros se definen separados por una coma. Los parámetros de los procedimientos almacenados de MySQL pueden ser de tres tipos:

- **IN:** Es el tipo de parámetro que se usa por defecto. La aplicación o código que invoque al procedimiento tendrá que pasar un argumento para este parámetro. El procedimiento trabajará con una copia de su valor, teniendo el parámetro su valor original al terminar la ejecución del procedimiento.
- **OUT:** El valor de este parámetro puede ser cambiado en el procedimiento, y además su valor modificado será enviado de vuelta al código o programa que invoca el procedimiento.
- **INOUT:** Es una mezcla de los dos conceptos anteriores. La aplicación o código que invoca al procedimiento puede pasarle un valor a éste, devolviendo el valor modificado al terminar la ejecución.

Como toda tecnología, el uso de procedimientos almacenados tiene sus pros y sus contras.

- Ventajas
    - Al reducir la carga en las capas superiores de la aplicación, se reduce el tráfico de red y, si el uso de los procedimientos almacenados es el correcto, puede mejorar el rendimiento.

- Al encapsular las operaciones en el propio código SQL, nos aseguramos de que el acceso a los datos es consistente entre todas las aplicaciones que hagan uso de ellos.
- En cuanto a seguridad, es posible limitar los permisos de acceso de usuario a los procedimientos almacenados y no a las tablas directamente. De este modo evitamos problemas derivados de una aplicación mal programada que haga un mal uso de las tablas.

- Inconvenientes
  - Al igual que ocurre con toda tecnología, tenemos que formarnos para aprender a crear procedimientos, por lo que existe cierta curva de aprendizaje.
  - Otro posible problema puede ocurrir con las migraciones. No todos los sistemas gestores de bases de datos usan los procedimientos del mismo modo, por lo que se reduce la portabilidad del código.

¿Cómo crear un procedimiento almacenado?

Ahora que hemos configurado una base de datos y una tabla, vamos a crear tres procedimientos almacenados para mostrar el uso y las diferencias de cada tipo de parámetro.

Al definir los procedimientos, tendremos que usar delimitadores para indicar a MySQL que se trata de un bloque independiente. En los siguientes ejemplos, DELIMITER $$ frena la ejecución de MySQL, que se retomará de nuevo en la sentencia DELIMITER del final.

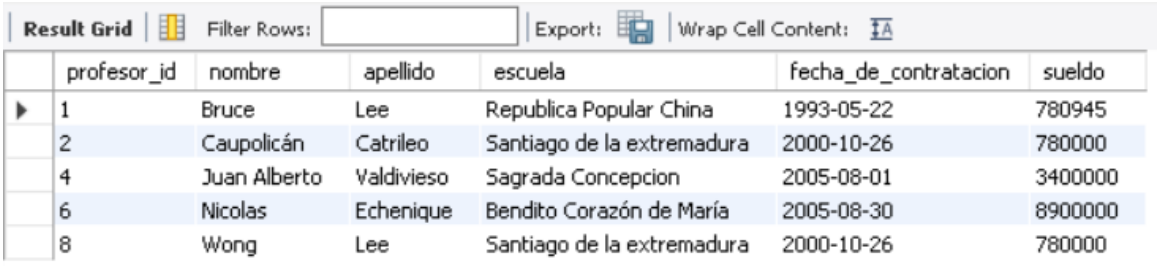
Siguiendo con el ejemplo original, vamos a obtener los profesores que tienen un determinado salario. Para crear el procedimiento, ejecuta estas sentencias SQL:

```
DELIMITER $$
CREATE PROCEDURE obtenerProfesores (IN vsueldo INT)
BEGIN
   SELECT *
   FROM profesor
   WHERE sueldo > vsueldo;
END$$
```

El sueldo a evaluar está contenido en el parámetro **vsueldo** que hemos definido como IN. Suponiendo que quieras obtener los profesores con un sueldo mayor a 700000, tendrías que invocar al procedimiento de este modo:

```
CALL obtenerProfesores(700000);
```

Este será el resultado:



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

profesor_id	nombre	apellido	escuela	fecha_de_contratacion	sueldo
1	Bruce	Lee	Republica Popular China	1993-05-22	780945
2	Caupolicán	Catrileo	Santiago de la extremadura	2000-10-26	780000
4	Juan Alberto	Valdivieso	Sagrada Concepcion	2005-08-01	3400000
6	Nicolas	Echenique	Bendito Corazón de María	2005-08-30	8900000
8	Wong	Lee	Santiago de la extremadura	2000-10-26	780000

Ahora vamos a obtener el número de profesores que cumplen la condición indicada anteriormente. Para crear el procedimiento, ejecuta estas sentencias SQL:

```
DELIMITER $$
CREATE PROCEDURE contarProfesores(
   IN vsueldo INT,
   OUT vcantidad INT)
BEGIN
   SELECT count(profesor_id)
   INTO vcantidad
   FROM profesor
   WHERE sueldo > vsueldo;
END$$
```

Al igual que antes, pasamos el sueldo a evaluar como vsueldo, definido como IN. También definimos numero como parámetro OUT. Suponiendo que quieras obtener la cantidad de profesores que cumple la condición, debes llamar al procedimiento de este modo:

```
CALL contarProfesores(700000, @numero);
```

```
SELECT @numero AS CANTIDAD;
```

Este será el resultado:

3.2.14.- Manejo de errores

Una excepción es el aviso de un error que se está produciendo durante la ejecución de un trozo de código.

En el siguiente ejemplo, si ya hay un profesor con el mismo ID en la base de datos se producirá un error (clave primaria duplicada) e interrumpirá la ejecución.

```
BEGIN
...
INSERT INTO PROFESOR VALUES ('1', ..., ...);
...
END;
```

Si no queremos que se interrumpa la ejecución deberemos capturar el error para decidir qué hacer en ese caso. Para ello deberemos declarar un handler (que podemos traducir literalmente como "manejador" de excepciones) de la siguiente manera:

```
DECLARE tipo_handler HANDLER FOR condición[,...]
lista_sentencias
```

donde tipo_handler toma valor de [CONTINUE|EXIT]

- **CONTINUE:** la ejecución del programa continua
- **EXIT:** la ejecución termina

y condición toma valor de error_code | SQLSTATE [VALUE]

error_code y **SQLSTATE** son valores que los encontramos en el mensaje de error que queremos controlar. Basta con ejecutar el código hasta esperar a que aparezca el mensaje de error, y entonces apuntar dicho código para introducirlo en el manejador, o bien, consultar en la ayuda de MySQL el listado de todos los mensajes de error susceptibles de ser capturados y controlados por los manejadores: https://dev.mysql.com/doc/refman/8.0/en/

```
Error: 1051 SQLSTATE: 42S02 (ER_BAD_TABLE_ERROR)
Message: Unknown table '%s'
```

El siguiente ejemplo capturaría el error de tabla desconocida continuando con la ejecución permitiendo que el resto de instrucciones puedan finalizar:

```
DECLARE CONTINUE HANDLER FOR 1051
BEGIN
-- cuerpo handler
END;
```

El siguiente ejemplo capturaría ese mismo error pero forzando la finalización de la ejecución:

```
DECLARE EXIT HANDLER FOR SQLSTATE '42S02'
BEGIN
 -- cuerpo handler
END;
```

3.2.15.- Cursores y condicionales

Los cursores son estructuras temporales de almacenamiento auxiliar muy útiles cuando se construyen procedimientos sobre bases de datos. Para crear un cursor es necesario declararlo y definir la consulta select que lo poblará de valores:

```
DECLARE nombre_cursor CURSOR FOR sentencia_select
```

- La sentencia SELECT no puede contener INTO.
- Los cursores no son actualizables.
- Se recorren en un sentido, sin saltos.
- Se deben declarar antes de los handlers y después de la declaración de variables

Ejemplo:

```
DECLARE cur1 CURSOR FOR select nombre from profesor;
```

Para abrir el cursor que se haya definido

```
OPEN nombre_cursor
```

Para desplazarnos por el cursor

```
FETCH nombre_cursor INTO nombre_variable
```

Si no existen más registros disponibles, ocurrirá una condición de Sin Datos con el valor SQLSTATE 02000. Se debe definir una excepción para detectar esta condición.

Para cerrar el cursor:

```
CLOSE nombre_cursor
```

Anexo: Referencias

1.- ¿Qué es una base de datos?
Referencia: https://www.hn.cl/blog/para-que-sirven-la-bases-de-datos/

2.- MySQL: Consulta SELECT
Referencia:
https://www.anerbarrena.com/mysql-select-consultas-base-datos-5426/

3.- Funciones en MySQL
Referencia:
https://dev.mysql.com/doc/refman/8.0/en/date-and-time-functions.html

4.- Funciones de agregación en MySQL
Referencia:
https://www.campusmvp.es/recursos/post/Fundamentos-de-SQL-Agrupaciones-y-funciones-de-agregacion.aspx

5.- ¿Qué es un modelo de datos?
Referencia:
https://sites.google.com/site/modelamientodebasesdedatos/mysql-workbench

6.- Querys anidadas en MySQL
Referencia: https://www.javatpoint.com/mysql-join

7.- Procedimientos almacenados en MySQL
Referencia:
https://www.neoguias.com/procedimientos-almacenados-mysql/

8.- Procedimientos, excepciones y cursores
Referencia:
http://tisbddocs.dlsi.ua.es/inicio/sql/practicas-sql-espanol/procedimientos