

Introducción a MySQL

MySQL Overview

MySQL es la Base de datos de código abierto más popular del mundo.

Actualmente las estimaciones realizadas indican que hay mas de 15 millones de instalaciones activas.

Cuando utilizamos LAMP, WAMP, XAMPP es la M

- Linux-Apache-Mysql-PHP
- Windows-Apache-Mysql-PHP
- Cualquiera--Apache-Mysql-PHP-Perl



Incorporado por más de 3.000 ISVs (Vendedores independientes de SW) y OEMs (original equipment manufacturer)

Muy populares en las redes sociales (Facebook, Twitter, etc.)

MySQL Overview

MySQL es un sistema de gestión de Base de Datos Relacional de código abierto.

Es desarrollado, distribuido y apoyado por Oracle.



¿Porqué Oracle?

- MySQL completa la oferta de Productos de Oracle.
- MySQL representa una solución de BBDD mejor de su clase para aplicaciones basadas en la web.
- MySQL está preparada para web de próxima generación, móviles y aplicaciones embebidas.
- Disponible en la Nube (iCloud)



MySQL Overview





Características Generales

- Sistema Gestor de Base de Datos Relacional
- Sistema Open Source (Licencia GPL)
 - Cualquier persona puede descargarlo desde Internet sin pagar nada
 - El Software puede ser usado y modificado
- SGBDR rápido, fiable, escalable y fácil de usar.
- MySQL Server trabaja en entornos cliente / servidor o sistemas embebidos
- La base de datos número uno para aplicaciones basados en internet, impulsando sitios web líderes en todo el mundo, incluyendo Facebook, Twitter y YouTube.



Productos MySQL

MySQL dispone de varios productos para cubrir todas las necesidades del mercado.

MySQL Community Edition <ul style="list-style-type: none"> Versión GRATUITA Está disponible bajo licencia GPL 	MySQL Classic Edition Versión DE PAGO Ideal para ser incluida en ISV y OEM	
MySQL Cluster Community Edition <ul style="list-style-type: none"> Versión GRATUITA Está disponible bajo licencia GPL Cluster 	MySQL Standard Edition Versión DE PAGO Aplicaciones de Lectura y OLTP	
	MySQL Enterprise Edition Versión DE PAGO Aplicaciones de Lectura y OLTP. HA , Backup	
	MySQL Cluster Carrier Grade edición Versión DE PAGO Alta disponibilidad, Tolerancia a Fallos Cluster de alto Rendimiento	

<http://www.mysql.com/products/>

Productos MySQL: MySQL Enterprise Edition

MySQL Enterprise Edition es la versión más avanzada de MySQL.

Incluye el conjunto más completo de características avanzadas junto con un conjunto de herramientas de gestión y apoyo técnico.

Características:

- Escalabilidad de MySQL
- Securitización, Auditoria, Integración con otros productos Oracle
- Posibilidad de monitorización y Análisis de Consultas SQL
- Backup y Herramientas gráficas



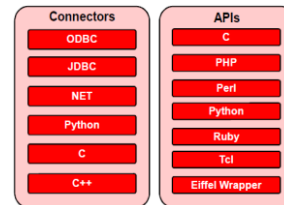
Productos MySQL: Conectores y API's

Como cualquier Base de Datos, no está aislada del sistema y dispone de una serie de Conectores y API's que lo facilitan:

- Conectores
 - Proporcionan conectividad del Servidor MySQL con las aplicaciones externas
- API's
 - Proporcionan acceso a bajo nivel para los protocolos y recursos de MySQL.

Conectores soportados por Mysql:

- PHP
- Perl
- Python
- Ruby
- C++ Wrapper



Para introducir **mysql** dentro de aplicaciones utilizamos la biblioteca del servidor **libmysql**

MySQL: Sistemas Operativos

MySQL está disponible en multitud de Sistemas Operativos.

En todas las plataformas se proporciona control y flexibilidad para realizar las instalaciones, gestiones y mantenimiento.

Los Sistemas Operativos mas habituales son:

- Windows (x86, x86_64)
- Linux (x86, x86_64)
- Oracle Solaris (SPARC, x86_64, x86)
- Mac OS X (x86, x86_64)
- Si queremos conocer todos los Sistemas Soportados por MySQL podemos consultarlos en el siguiente enlace

<http://www.mysql.com/support/supportedplatforms/database.html>

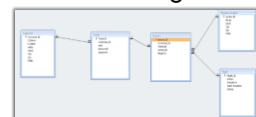
Base de Datos

Describir el modelado de bases de datos

- Modelado de bases de datos es el proceso de definición de la estructura lógica de una base de datos.
- Esta estructura determina cómo se almacenan los datos, como se organizan y como se manipulan.
- El modelo entidad-relación es el modelo más común utilizado para describir una Base de Datos Relacional:
 - Metodología TOP-DOWN
 - Comienza con visión global y va bajando añadiendo detalles
- El modelo entidad-relación finaliza con la creación del diagrama entidad-relación (ERD)



Hay varios tipos de ERD. Utiliza diferentes tipos en diferentes etapas del proceso y por diferentes razones.

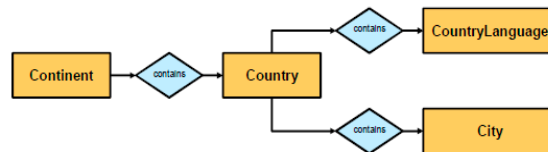


ERD: Diagrama de Estructura

- El modelo entidad-relación finaliza con la creación del diagrama entidad-relación (ERD).
- Mediante este diagrama podemos visualizar el contenido y las relaciones de una base de datos
 - Organiza los datos en grupos de entidades similares (que se colocará en las tablas)

Ejemplo:

- Diagrama de alto nivel de la base de datos *world_innodb*:



ERD: Diagrama de Cardinalidad

- Cuando definimos las relaciones entre las Entidades, deberemos indicar la Cardinalidad de las mismas.
 - Uno a Uno Uno a mucho
 - Muchos a Uno Muchos a muchos, etc
- El tipo de notación, habitual, se llama notación "pata de gallo". (Utiliza las líneas y símbolos específicos (notación) para mostrar estas relaciones).

○ Cero a uno

|| Exactamente uno (uno y sólo uno)

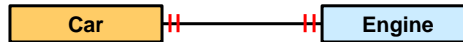
○ Cero o mas

⌞ Uno o mas

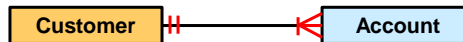
ERD: Diagrama de Cardinalidad

Ejemplos

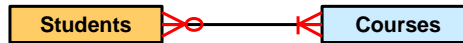
Uno-a-uno: Un coche puede tener sólo un motor.



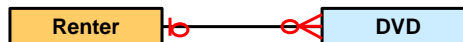
Uno-a-uno (o muchos): Un cliente de un banco puede tener una o más cuentas.



Cero (o mucho) a-uno (o muchos): Cero o muchos estudiantes pueden tomar uno o muchos cursos.



Cero (o uno) a-cero (o muchos): Cero o una persona puede alquilar cero o muchos DVDs.



Claves

- Una tabla consta de varias filas, cada una de las cuales corresponde a un registro.
- Una clave es un identificador único para cada registro.
- La base de datos utiliza claves para identificar registros y soportar la integridad referencial.
- Disponemos de diferentes tipos de Claves, pueden aparecer o no:
 - Claves candidatas
 - Claves Primarias
 - Claves Ajenas o Foráneas
- Las claves pueden estar compuesta de una o varias columnas.

Claves

Claves candidatas

- Todas las claves que pueden existir en un tabla

Claves Primarias

- Columna o columnas que permiten distinguir unívocamente una fila de otra en la misma tabla

Claves Ajenas o Foráneas

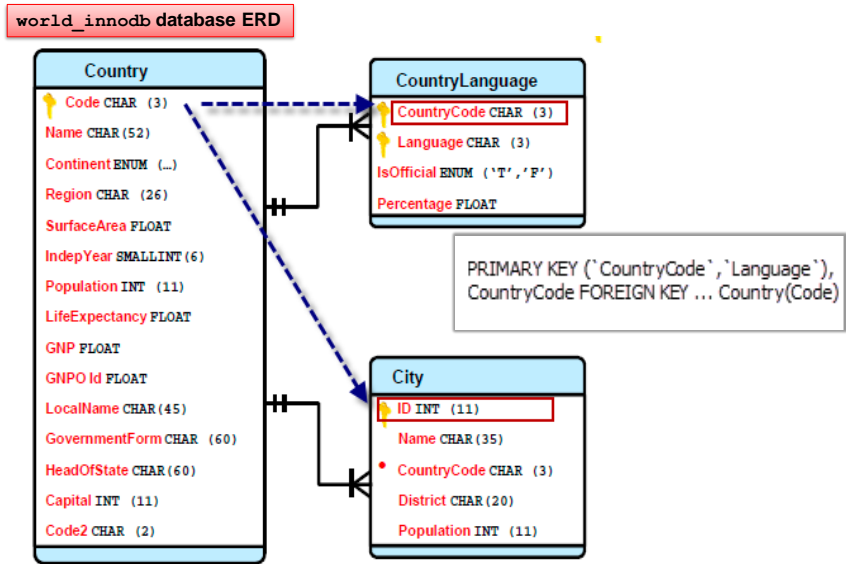
- Representa la relación entre tablas
- En otra tabla es una clave primaria
 - Los valores que tome en nuestra tabla tienen que existir previamente en la tabla en la que es clave primaria
- Mediante Claves Ajenas damos forma a la Integridad Referencial



INTEGRIDAD REFERENCIAL

- La integridad referencial exige que la clave externa de cualquier tabla de referencia debe referirse siempre a una fila válida de la tabla referenciada.
- La integridad referencial asegura que los datos de tablas relacionadas permanece constante durante las actualizaciones y eliminaciones.

Claves



Normalización

- Proceso que realiza una optimización del diseño de la B.D. para construir una B.D.Relacional
- Proceso que permite:
 - Asegura que cada dato individual se almacena sólo una vez
 - Evita que los datos no válidos o incoherentes después de la modificación
 - "Se descompone" base de datos en tablas más pequeñas
 - Evita que una columna contenga valores múltiples
 - Evita los grupos repetitivos ("tablas dentro de tablas").
 - Ayuda con un buen diseño de base de datos
- Un modelo de datos puede encontrarse en diferentes estados de normalización

Ventajas de la Normalización

Una **mejor organización** de base de datos:

- Más tablas con filas más pequeñas.
- El acceso de datos eficiente.
- Mayor flexibilidad para las consultas.
- Capacidad para encontrar rápidamente la información que necesita.
- Más fácil de implementar la seguridad.
- Más fácil de mantener a medida que cambian sus necesidades.

Reducción de los datos redundantes y duplicado:

- Base de datos más compacta.
- Más fácil de asegurar Datos consistentes después de la modificación

Des-Ventajas de la Normalización

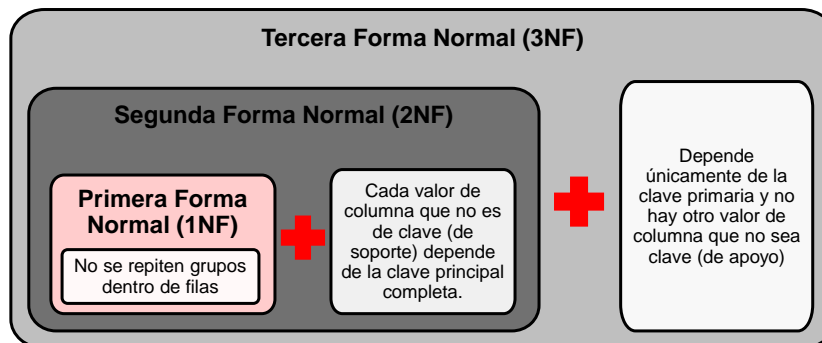
Las Desventajas mas directas son:

- Consultas simples, a menudo, necesitan tener acceso a varias tablas.
- Tablas menudo contienen códigos en lugar de los datos reales.
 - Necesidad de buscar los códigos en tablas relacionadas
 - Optimizado para aplicaciones, no consulta en general
- Las operaciones JOIN pueden reducir el rendimiento de base de datos.
- El proceso de normalización es complejo y consume mucho tiempo.

Normal Forms

Hay muchos niveles sucesivos de normalización. Estos se llaman **formas normales**.

- Cada forma normal consecutiva depende de la anterior.
- Las tres primeras formas normales suelen ser adecuadas..



Formas Normales

FN1 (1971)

- Las entidades tienen Identificador Único Principal
- No existen grupos repetitivos de datos (conjunto de valores con el mismo significado que se repiten a lo largo de la entidad)

FN2 (1973)

- Está en FN1
- Todos los atributos que no formen parte del Identificador Único Primario dependen de todo el IUP

FN3 (1973)

- Está en FN2
- No existen dependencias transitivas entre el IUP y ningún otro atributo que no forme parte del Identificador Único Primario

Cuando todas las entidades están en Tercera Forma Normal se puede decir que el modelo está en FN3

***“Ninguna aplicación de cierta importancia
funcionará en la 3 FN” (George Koch)***

Estrategias de Normalización. Ejemplo 1

FN2 (1973)

- Está en FN1
- Cuando un atributo NO clave, dependa de manera completa de toda la clave.

DNI	Codigo	Nombre	Apellido	Nota
1	34	Juan	García	9
1	35	Juan	García	7
2	34	Pedro	Reyes	3

- Sólo la nota depende de la PK.
- Nombre + Apellido dependen del DNI

DNI	Código	NOTA
1	34	9
2	35	7
3	34	3

DNI	Nombre	Apellido
1	Juan	García
2	Pedro	Reyes

Estrategias de Normalización. Ejemplo 1

FN3 (1973)


- Está en FN2
- Cuando Ningún atributo NO CLAVE, no depende de otro atributo no clave de la tabla. Eliminación de dependencias transitivas.

DNI	Nombre	CP	Provincia
1	Juan	28	Madrid
2	Pedro	08	Barcelona
3	Ana	28	Madrid

- La provincia depende del Código Postal

DNI	Nombre	CP
1	Juan	28
2	Pedro	08
3	Ana	28

CP	Provincia
28	Madrid
08	Barcelona



Normalization Process: Example

Base de datos de inventario para una cadena de tiendas de muebles

- Convierta la hoja de cálculo del inventario en una base de datos, para que pueda:
 - Ejecutar búsquedas inteligentes
 - Hacer modificaciones
 - Prepararse para el crecimiento futuro de su negocio
- Hoja de inventario original:

Inventory									
sID	sLoc	sPostal	pID1	pName1	pQty1	pID2	pID	pName2	pQty2
1	Holtsville	00501	1	bed	15	2	2	chair	4
2	Waukesha	53146	1	bed	4	3	3	table	6
3	Waukesha	53146	2	chair	8	4	4	sofa	4
4	Ketchikan	99950	2	chair	24	4	4	sofa	10

Primera Forma Normal: 1NF

Inventory								
sID	sLoc	sPostal	pID1	pName	pQty1	pID2	pName2	pQty2
1	Holtsville	00501	1	bed	15	2	chair	4
2	Waukesha	53146	1	bed	4	3	table	6
3	Waukesha	53146	2	chair	8	4	sofa	4
4	Ketchikan	99950	2	chair	24	4	sofa	10

1NF

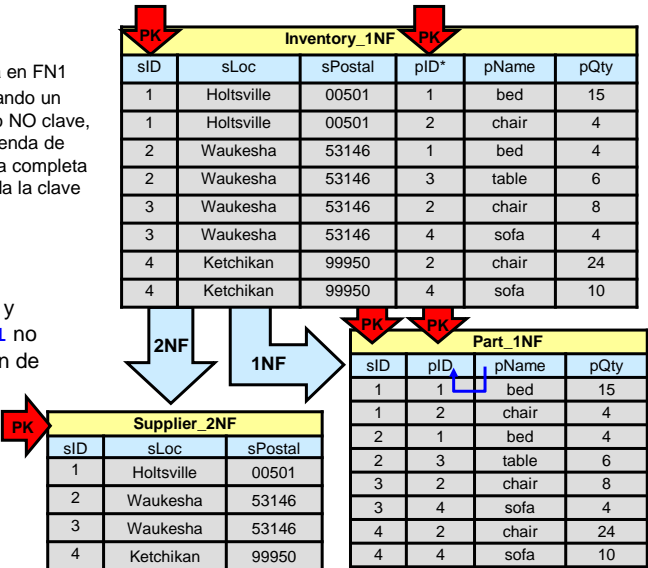
Inventory_1NF					
sID	sLoc	sPostal	pID*	pName	pQty
1	Holtsville	00501	1	bed	15
1	Holtsville	00501	2	chair	4
2	Waukesha	53146	1	bed	4
2	Waukesha	53146	3	table	6
3	Waukesha	53146	2	chair	8
3	Waukesha	53146	4	sofa	4
4	Ketchikan	99950	2	chair	24
4	Ketchikan	99950	4	sofa	10

* Esta columna no suele aparecer en una primera forma normal; sin embargo, Está aquí para demostrar una conexión con la segunda forma normal.

Segunda Forma Normal: 2NF

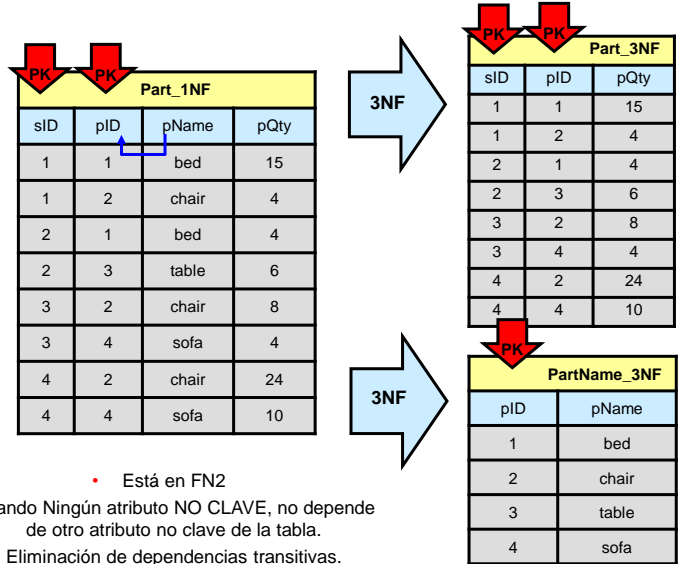
- Está en FN1
- Cuando un atributo NO clave, dependa de manera completa de toda la clave

sLoc y sPostal no dependen de pID.

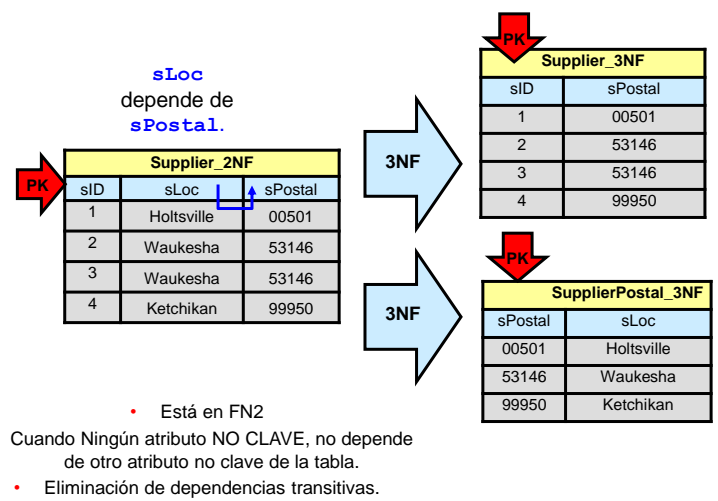


pName sólo depende de pID.

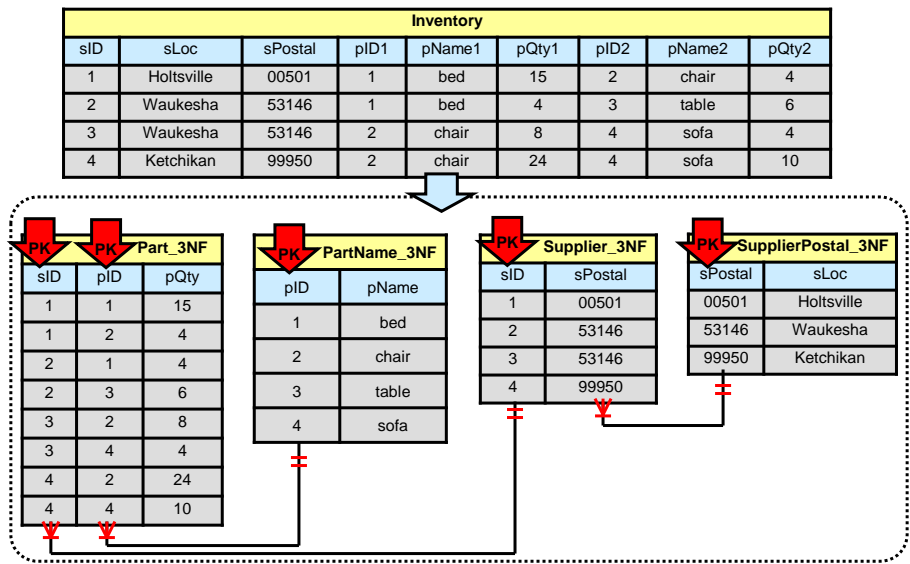
Tercera Forma Normal: 3NF



Tercera Forma Normal: 3NF



Base de datos normalizada de "Tiendas de muebles"



Consideraciones de Diseño de Base de Datos.

- Antes de realizar las operaciones de CREATE DATABASE o CREATE TABLE es necesario realizar un diseño apropiado del mismo.
- Siempre hay que diseñar la base de datos antes de crear, incluso si no utiliza métodos formales.

Recomendaciones:

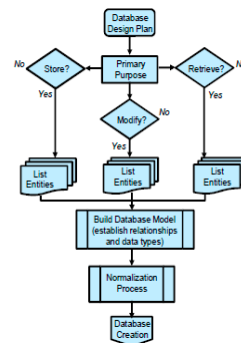
- Dar nombres sensibles a Tablas y Columnas
- Utilice la normalización en BBDD nuevas o ya creadas.
- Normalizar sólo hasta el punto en el que sea útil

Preparar un plan de diseño

- Para diseñar un plan para la Base de Datos necesitamos responder a una serie de preguntas:
 - ¿Cuál es el propósito principal de esta base de datos?
 - ¿Qué entidades necesitan ser almacenados, recuperados, y / o modificado?
 - ¿Cómo esas entidades se relacionan entre sí?

Empezaremos desarrollando un Diagrama para Visualizar dicho plan.

- Identificar las entidades clave y dibujarlas
- Anotar los atributos de cada una
- Anotar las Claves existentes (PK, FK, Unique, etc)
- Requisitos de almacenamiento (tipos de datos, etc)
- Elección del Motor de BBDD mas óptimo.
- Normalizar hasta 2FN o 3FN



Ver y Evaluar Diseño de base de datos

MySQL dispone de varias sentencias para poder ver la estructura de la Base de Datos como las siguientes:

- **SHOW DATABASES**
 - Visualizar el nombre de todas las BBDD / Esquemas del Gestor
- **USE <database_name>**
 - Cambiar la BBDD / Esquema por defecto de utilización
- **SHOW TABLES**
 - Visualizar todas las tablas del Esquema / BBDD actual
- **DESCRIBE <table_name>**
 - Visualiza la estructura de una determinada tabla en la BBDD utilizada
- **SELECT * FROM <table_name>**
 - Visualiza los datos de una determinada tabla

Ver y Evaluar Diseño de base de datos

- Para poder evaluar el Diseño definido en una Base de Datos/Esquema, deberemos utilizar los comandos anteriores.
- Algunos comandos sólo se utilizarán una vez (`show databases`), mientras que otros lo deberemos repetir N veces (`describe <table>`).
- La evaluación del Diseño consiste en asegurarnos que todo lo decidido en el plan/estudio se ha llevado a la práctica.
- Habitualmente, con el tiempo, el plan de diseño y la implementación de BBDD suelen no coincidir.

Evaluar un diseño de base de datos

PASOS

1. Conectarse con el Servidor MySQL

```
shell> mysql -uroot -p
Enter password: <password>
```

2. Visualizar todas las Bases de Datos / Esquemas existentes

```
mysql> SHOW DATABASES;
+-----+
| Database           |
+-----+
| information_schema |
| mysql               |
| performance_schema |
| test                |
| world_innodb        |
+-----+
```

3. Cambiar el Esquema / BBDD por defecto

```
mysql> USE world_innodb
Database changed
```

Evaluar un diseño de base de datos

PASOS

4. Visualizar todas las tablas existentes en el Esquema/BBDD usada

```
mysql> SHOW TABLES;
+-----+
| Tables_in_world_innodb |
+-----+
| city                    |
| country                 |
| countrylanguage         |
+-----+
```

5. Visualizar todas las Columnas de una tabla específica (Estructura)

```
mysql> DESCRIBE City;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ID    | int(11) | NO | PRI | NULL | auto_increment |
| Name  | char(35) | NO | | | |
| District | char(20) | NO | | | |
| Population | int(11) | NO | | 0 | |
+-----+-----+-----+-----+-----+-----+
```

Evaluar un diseño de base de datos

PASOS

6. Visualizar todos los datos de una tabla específica

```
mysql> SHOW TABLES;
mysql> SELECT * FROM City;
+-----+-----+-----+-----+-----+
| ID | Name | CountryCode | District | Population |
+-----+-----+-----+-----+-----+
| 1 | Kabul | AFG | Kabul | 1780000 |
| 2 | Qandahar | AFG | Qandahar | 237500 |
| 3 | Herat | AFG | Herat | 186800 |
| 4 | Mazar-e-Sharif | AFG | Balkh | 127800 |
| 5 | Amsterdam | NLD | Noord-Holland | 731200 |
| 6 | Rotterdam | NLD | Zuid-Holland | 593321 |
...
| 4079 | Rafah | PSE | Rafah | 92020 |
+-----+-----+-----+-----+-----+
```

Repetir las etapas 5 y 6 para cada tabla existente

Sistema Gestor de Base de Datos

Concepto de **Esquema**:

- Dependiendo del SGBDR el concepto de puede variar:
- **ORACLE**
 - Esquema conjunto de objetos de Base de Datos que pertenecen a un usuario.
 - Hay una relación 1 a 1 entre Esquema y Usuario.
 - Pueden existir muchos en una Base de Datos
- **SQL Server**
 - Esquema conjunto de objetos de Base de Datos que pueden ser compartidos por los diferentes usuarios.
 - Hay una relación 1 a N entre Esquema y Usuarios.
 - Pueden existir muchos en una Base de Datos
- **MySQL**
 - Para MySQL el concepto de Esquema es equivalente a Base de Datos.
 - En el Gestor MySQL puede encontrarse con muchas bases de Datos = Esquemas.
 - Hay una relación 1 a 1 entre Esquema y BBDD.

Identificadores

- Un *identificador* es un nombre dado a uno de los siguientes objetos de base de datos:
 - Alias
 - Database
 - Table
 - Column
 - Index
 - Stored routine
 - Trigger
- Puede asignar un identificador con o sin comillas.

Reglas de Identificadores: Con y sin Comillas

- Identificadores sin comillas:
 - Puede contener todos los caracteres alfanuméricos, el subrayado (_), y el signo de dólar (\$))
 - Puede comenzar con cualquiera de los caracteres legales (incluso un dígito)
 - No puede ser enteramente en dígitos
- Identificadores con comillas:
 - Puede contener caracteres como espacios o guiones que no son legales
 - Debe estar encerrado entre comillas invertidas (`)
 - Puede ser incluido entre comillas dobles (") cuando el modo **ANSI_QUOTES** de SQL ha sido habilitado

Identificadores. ALIAS

- Permite la utilización de palabras reservadas (como SELECT o DESC,etc), siempre que vayan entre comillas.
- Podemos utilizar comillas simples ('), comillas invertidas (`) o comillas dobles (").
- Los **ALIAS** puede incluir cualquier carácter.
- Sino estamos seguro de que el identificador sea correcto → comillas

```
mysql> SELECT 1 AS INTEGER;  
ERROR 1064 (42000): You have an error in your SQL syntax. Check  
the manual that corresponds to your MySQL server version for  
the right syntax to use near 'INTEGER' at line 1
```

```
SELECT 1 AS 'INTEGER';  
SELECT 1 AS "INTEGER";  
SELECT 1 AS `INTEGER`;
```

Nombres Cualificados

- Se denomina nombre cualificado cuando nos referimos a un objeto mediante `<nombre_BBDD>.<nombre_objeto>`
- Podemos definir nombre cualificados para tablas y columnas
`<Nombre_BBDD>.<nombre_objeto>`
`<Nombre_BBDD>.<nombre_objeto>.<Columna>`

```
SELECT * FROM world_innodb.Country;  
SELECT Country.Name FROM Country;  
SELECT world_innodb.Country.Name  
FROM world_innodb.Country;
```

- Los nombre cualificados son utilizados para resolver la ambigüedad que se pueden producir cuando disponemos de tablas / columnas con el mismo nombre.

Mayúsculas y Minúsculas

- Cuando trabajamos con MySQL, deberemos tener en cuenta que, dependiendo del Sistema Operativo, algunos objetos **DISTINGUEN** entre mayúsculas y minúsculas.
- **case-sensitive**
 - **Base de Datos, Tablas y Disparadores**
 - Son creadas como directorios y Ficheros en el Sistema Operativo, por lo que si el SO distingue entre MAY/MIN en MySQL también.
 - Si el SO no distingue entre mayúsculas y minúsculas, estos identificadores tampoco distinguen entre mayúsculas y minúsculas.
- **NO case-sensitive**
 - **Columnas, Índices, Rutinas almacenadas.**
 - Son creados dentro del propio Gestor por lo que no existe la posibilidad anterior.

```
VARIABLE : lower_case_table_names = 1
```

Los nombres de tablas se almacenan en minúsculas en disco y las comparaciones de nombre no distinguen entre mayúsculas y minúsculas.

Creación de Base de Datos

- Tras llegar a nuestras manos el cuaderno de carga, deberemos de crear las estructuras básicas para nuestros datos:
 - Base de Datos
 - Tablas

Creación de Bases de Datos

- La sintaxis general de creación es la siguiente:

```
CREATE DATABASE <database_name> [<options>]
```

- Si obtiene un error como:

```
- ERROR 1044 (42000): Access denied for user 'monty'@'localhost' to database
```

Significa que la cuenta de usuario utilizada no tiene los privilegios necesarios para hacer esta operación.
(CREATE / CREATE SCHEMA)

ESQUEMA = BASE DE DATOS

Creación de Base de Datos

- La operación de `CREATE DATABASE`, se realiza una sola vez.
- Si la base de Datos ya existe, se producirá un error de creación; podemos añadir la opción `IF NOT EXISTS` para que dicho error no se produzca.

```
CREATE DATABASE IF NOT EXISTS mydatabase
```

- También podemos definir una serie de características de creación para la BBDD como el Character SET o Collate

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name  
    [DEFAULT] CHARACTER SET [=] charset_name  
    | [DEFAULT] COLLATE [=] collation_name
```

Creación de BBDD. Conversión de Nombres

- En MySQL las BBDD y las Tablas se crean como directorios y ficheros del Sistema Operativo.
- Debido a esto, debemos de tener en cuenta que hay SO que son **case-sensitive** (distinguen entre mayúsculas y minúsculas)
 - Nombres en Windows **NO** distinguen entre mayúsculas y minúsculas
 - Servidores MySQL en UNIX **SI** distinguen

CARACTERISTICAS

- Los nombres no pueden tener mas de 64 caracteres.
- No se pueden utilizar ciertos caracteres como: **ASCII (0), ASCII (255), /, \, y .**
- Se pueden utilizar palabras reservadas y caracteres especiales, siempre y cuando estén entre comillas invertidas

```
CREATE TABLE `my table` (ID INT);  
SELECT `STATUS` FROM this_table;
```

Modificar una BBDD

- Para poder modificar las opciones de una Base de Datos disponemos de la opción:

ALTER DATABASE [Nombre_BBDD] opciones

Sino indicamos el nombre de la BBDD, MySQL lo realizará en la Base de Datos que está usando actualmente, sino ERROR.

- Podemos modificar diferentes opciones de la BBDD excepto su nombre.

```
ALTER DATABASE mydb COLLATE utf8_polish_ci;  
ALTER DATABASE mydb CHARACTER SET latin1 COLLATE  
latin1_swedish_ci;
```

Los cambios afectan sólo a las tablas que se crean después de realizar ALTER DATABASE.

Eliminar (o borrar) una base de datos

- Al igual que la orden **CREATE DATABASE**, MySQL dispone de la orden inversa: **DROP DATABASE**.
- Ambas ordenes se engloban dentro de una categoría de SQL denominada **DDL** (Lenguaje de Definición de Datos)
 - **CREATE, ALTER, DROP, RENAME, TRUNCATE**
- Esta categoría se caracteriza por realizar operaciones atómicas y que son validadas si se han ejecutado correctamente.
- La órdenes **DDL** afectan a la estructura/Metadatos de MySQL .
- Los usuarios, normalmente, disponen de **privilegios de ejecución** de este tipo de orden dentro de su sesión y no a nivel **GLOBAL**.

Eliminar (o borrar) una base de datos

DROP DATABASE <nombre>

- La ejecución de esta orden, implica que MySQL elimina todos los objetos que contiene dicha BASE DE DATOS.
 - Tablas, Procedimientos almacenados y disparadores, Índices, etc
- La ejecución exitosa de la orden, devuelve un número = número de tablas borradas.
 - (En realidad es el número de archivos **.frm** borrados)
- La orden **SHOW DATABASES** **no** mostrará la Base de Datos anterior.
- Para poder ejecutar dicha orden requiere privilegios de **DROP** en la Base de Datos.

```
DROP DATABASE mydb ;
```

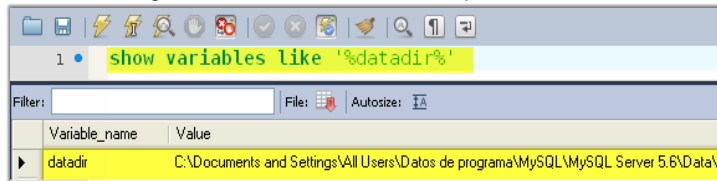
- Si la Base de Datos NO EXISTE → Error

```
DROP DATABASE IF EXISTS mydb ;
```


Eliminar (o borrar) una base de datos

DROP DATABASE <nombre>

- Esta es una operación que **no se puede revertir**.
- MySQL elimina los directorios y ficheros donde la Base de Datos está almacenada.
 - Base de Datos → Directorio
 - Tablas → Ficheros
- Las BBDD se guardan en el directorio indicado por la variable **DATADIR**



- Mediante el comando **SHOW ERRORS**, podremos ver si se han producido errores en la orden de borrado.

Tipos de Datos

Tipos de datos en el diseño de bases de datos

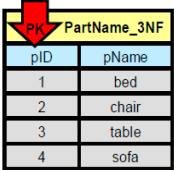
- Una base de datos bien diseñada utiliza el tipo de datos más adecuada para cada elemento de datos.
- Debemos de seleccionar el tipo de datos mas adecuado para poder obtener un rendimiento óptimo en el acceso a ellos.
- Si tenemos un buen «Cuaderno de Carga», en él se nos indicará el tipo GENERICO a utilizar y es cuestión nuestra definir el mas correcto.

– **pID**

- Es una columna numérica (Genérico)

– **pName**

- Es una columna con caracteres alfabéticos (Genérico)



PartName_3NF	
pID	pName
1	bed
2	chair
3	table
4	sofa

Tipos de datos en el diseño de bases de datos

Para seleccionar el tipo de dato mas adecuado, habitualmente se sigue la regla ABC (Apropiado-Breve-Completo)

- **Apropiado**

- Elija el tipo de datos, de los que tenemos disponibles que mas se adecue a la columna de la tabla que queramos definir.
- Nos basaremos en el Modelo De Datos – Cuaderno de Carga

- **Breve**

- Elija el tipo de datos que utiliza la menor cantidad de espacio de almacenamiento.
- Esto permitirá ahorrar recursos y mejora el rendimiento.

- **Completa**

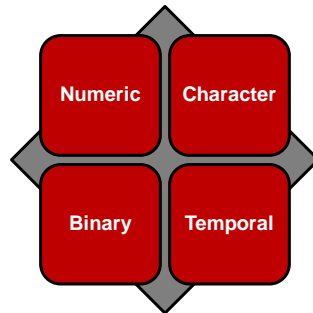
- Asegúrese de que el tipo de datos puede almacenar el mayor valor posible para cada columna y sin pérdida de datos.

- Disponemos de numerosos subtipos que utilizan distintas cantidades de memoria y espacio en disco y afectan el rendimiento

Categorías de datos en MySQL

MySQL dispone de 4 categorías de datos:

- **Numeric:** Valores Numéricos
- **Temporal:** Valor de Fechas y Tiempo
- **Carácter:** Cadenas de texto
- **Binario:** Cadenas de datos Binarios



Categorías de datos en MySQL

NUMERIC

- Antes de definir el tipo de dato numérico, debemos de considerar unos factores para poder definir cual es el mejor SUBTIPO.
 - Gama de valores a almacenar
 - Cantidad de espacio de almacenamiento necesario
 - Columna de precisión y escala (flotante-punto y punto fijo)
- Disponemos de:
 - **ENTEROS** Valores SIN Decimales
 - **COMA FLOTANTE** (Floating-point) Valor APROXIMADO para numero fraccionarios
 - **DECIMAL FIJO** (Fixed-point) Valor EXACTO para numero fraccionarios
 - **BIT** Valores en binario

Categorías de datos en MySQL

NUMERIC. ENTEROS

- Tipos Disponibles
 - TINYINT Tipo de datos entero muy pequeño. 1 BYTE
 - SMALLINT: Tipo de datos entero Pequeño. 2 BYTES
 - MEDIUMINT: Tipo de datos entero Mediano. 3 BYTES
 - INT, INTEGER: Tipo de datos Normal (promedio) 4 BYTES
 - BIGINT: Tipo de datos grande. 8 BYTES
- Atributos
 - UNSIGNED Sin signo. Todos números positivos
 - ZEROFILL Reemplaza los espacios en blanco por 0

INT [(M)] M se utiliza con la opción UNSIGNED o ZEROFILL e indica los dígitos a visualizar rellenos de 0. Máximo (255)

NO INDICA tamaño de almacenamiento, ni número de dígitos máximos.

INT(5) ZEROFILL → con valor 4 → 00004.

Categorías de datos en MySQL

NUMERIC

Type	Storage Required	Signed Range	Unsigned Range
TINYINT	1 byte	−128 to 127	0 to 255
SMALLINT	2 bytes	−32,768 to 32,767	0 to 65,535
MEDIUMINT	3 bytes	−8,388,608 to 8,388,607	0 to 16,777,215
INT	4 bytes	−2,147,683,648 to 2,147,483,647	0 to 4,294,967,295
BIGINT	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0 to 18,446,744,073,709,551,615

Categorías de datos en MySQL

NUMERIC. COMA FLOTANTE

- Son utilizados para número ENTEROS o FRACCIONARIOS. Disponemos de los Tipos FLOAT y DOUBLE

Float

- Un valor de coma flotante de precisión simple (FLOAT) tiene una precisión de aproximadamente 7 cifras decimales. (4 bytes)
- **Float (M , D)**
 - **M** Número totales de Dígitos (El número de dígitos ENTEROS = M-D)
 - **D** Número de dígitos Decimales a guardar

Float (4,2) Se almacenan, como máximo 2 decimales y MAXIMO 2 dígitos ENTEROS

EJEMPLO:

```
mysql> Create table prueba1 ( c1 FLOAT( 4, 2 ));
mysql> insert into prueba1 values (12.20);    OK
mysql> Insert into prueba1 values (123.0);    Error
mysql> insert into prueba1 values (12.2021);  OK
```

Categorías de datos en MySQL

NUMERIC. COMA FLOTANTE

DOUBLE

- Un valor de coma flotante de precisión compleja (DOUBLE) tiene una precisión de aproximadamente 15 cifras decimales. (8 bytes)
- **DOUBLE (M , D)**
 - **M** Número totales de Dígitos (El número de dígitos ENTEROS = M-D)
 - **D** Número de dígitos Decimales a guardar

DOUBLE (4,2) Se almacenan, como máximo 2 decimales y MAXIMO 2 dígitos ENTEROS

EJEMPLO:

```
mysql> Create table prueba2 ( c1 DOUBLE( 4, 2 ));
mysql> insert into prueba2 values (12.20);    OK
mysql> Insert into prueba2 values (123.0);    Error
mysql> insert into prueba2 values (12.2021);  OK
```

Categorías de datos en MySQL

NUMERIC. COMA FLOTANTE

Type	Storage Required	Signed Range	Unsigned Range
FLOAT	4 bytes	-3.402823466E+38 to -1.175494351E-38 to 1.175494351E-38 to 3.402823466E+38	0 and 1.175494351E-38 to 3.402823466E+38
DOUBLE	8 bytes	-1.7976931348623157E+308 to -2.2250738585072014E-308 and 2.2250738585072014E-308 to 1.7976931348623157E+308	0 and 2.2250738585072014E-308 to 1.7976931348623157E+308

Categorías de datos en MySQL

NUMERIC. DECIMAL FIJO

- Todos los valores de una columna **DECIMAL** tienen el mismo número de decimales.
- Características:
 - **DECIMAL** se almacena exactamente como aparecen, cuando sea posible.
 - NO son tan eficientes como **FLOAT** y **DOUBLE**.
 - SI sufre REDONDEAN (0-4, 5-9)
- Sintaxis
 - DECIMAL (M, D)**
 - M Número totales de Dígitos (65) (El número de dígitos ENTEROS = M-D)
 - D Número de dígitos Decimales a guardar (30)

DECIMAL (4, 2) Se almacenan, como máximo 2 decimales y MAXIMO 2 dígitos ENTEROS

EJEMPLO:

```
mysql> Create table prueba3 ( c1 DECIMAL ( 4, 2 ));
mysql> insert into prueba3 values (12.20); OK
mysql> Insert into prueba3 values (123.0); Error
mysql> insert into prueba3 values (12.2091); OK
```

Categorías de datos en MySQL

NUMERIC. BIT

- Se utiliza para almacenar un número determinado de **BITS**.
- Para una columna **BIT (n)** la columna, el rango de valores es de 0 a 2^n-1
- Puede asignar valores de columna **BIT** usando expresiones numéricas con la siguiente expresión (**b'xxxx'**)

EJEMPLO:

Almacenar 4 bits y otra de 20 bits

```
bit_coll1    BIT(4)  
bit_coll2    BIT(20);
```

```
b'1111'  -> 15
```

```
b'1000000' -> 64
```

Categorías de datos en MySQL

TEMPORALES

- MySQL ofrece varios tipos de datos diferentes para la fecha y tiempo.
- Aquí, AAAA, MM, DD, HH, MM y SS representan año, mes, día del mes, hora, minuto y segundo, respectivamente.

Categorías de datos en MySQL

TEMPORALES

- MySQL ofrece varios tipos de datos diferentes para la fecha y tiempo.
- TIME:**
 - **HH:MM:SS** 12:59:02
- YEAR:**
 - 4 dígitos (**YYYY**) 2012
- DATE:**
 - **YYYY-MM-DD** 2012-06-04
- DATETIME:**
 - **YYYY-MM-DD HH:MM:SS** 2012-06-04 12:59:02
- TIMESTAMP:**
 - Fecha y hora actuales 2012-06-04 12:59:02
 - **DEFAULT CURRENT_TIMESTAMP**
- TIME, DATETIME y TIMESTAMP**
 - Los valores pueden aceptar segundos fraccionarios con precisión de hasta seis dígitos.

NOTA:

- Si introducimos un valor temporal incorrecto, se inserta todo a ceros (0000-00-00)

Categorías de datos en MySQL

TEMPORALES

- TIME**
 - Puede expresarse hora del día o de una duración determinada
 - Se muestra como **HH: MM: SS** y se almacenan en 3 bytes
 - Los valores de tiempo pueden ser asignados por el uso de cadenas o números.
 - Tiene un rango de 00:00:00.000000 a 23:59:59.999999.) Permitir valores fuera de ese rango es una extensión de MySQL
- YEAR**
 - Permite guardar el valor año con 4 cifras completas
 - Sintaxis: **YEAR [(4)]** .
 - El rango de valores permitido es de 1901 a 2155, y 0000.
 - Los valores se guardan en **1 bytes** (256 valores posibles) → SMALLINT
- DATE**
 - Permite guardar fechas completas, en el formato **YYYY-MM-DD**
 - El intervalo permitido es de **01-01-1000** a **31-12-9999** y se almacenan en **3 bytes**
 - Valores inferiores se rellenan a 0 al principio
 - Los valores pueden ser asignados como cadenas o números.

Categorías de datos en MySQL

TEMPORALES

- **DATETIME**
 - Es una combinación de fecha, hora y fracciones de Segundos
 - El rango de valores permitidos es de **01-01-1000-01-01 00:00:00** a **31-12-9999 23:59:59**
 - Valores inferiores se rellenan a 0 al principio
 - El requisito de almacenamiento es de **8 bytes**.
 - Puede incluir fracciones de segundos: **2011-12-2014 23:59:59.542545**
- **TIMESTAMP**
 - Es un valor especial de DATETIME para almacenar la fecha y hora actuales
 - El rango de valores permitidos es **01-01-1970 00:00:01,000000** a **19-01-2038 03:14:07,999999**.
 - Valores inferiores se rellenan a 0 al principio
 - Los valores TIMESTAMP se convierten de la zona horaria actual a UTC para su almacenamiento, y se convierten de nuevo a la UTC a la zona horaria actual para su recuperación.
 - El requisito de almacenamiento es de 4 bytes.

```
Mysql> set sql_mode='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION';
```

Categorías de datos en MySQL

TEMPORALES

Type	Storage Required	Range
DATE	3 bytes	1000-01-01 to 9999-12-31
TIME*	3 bytes + fractional seconds storage	-838:59:59 to 838:59:59
DATETIME*	5 bytes + fractional seconds storage	1000-01-01 00:00:00 to 9999-12-31 23:59:59
TIMESTAMP*	4 bytes + fractional seconds storage	1970-01-01 00:00:00 to 2038-01-19 03:14:07
YEAR	1 byte	1901 to 2155

Categorías de datos en MySQL

CARACTER

- Los tipos de datos **CARÁCTER** representa una secuencia de caracteres alfanuméricos que pertenecen a un conjunto de caracteres dado.
- Pueden contener datos de texto o binarios.
- Están soportados por casi cualquier lenguaje de programación.
- Soportan diferentes juegos de caracteres y colaciones (**COLLATION**)
- Estos tipos de datos se diferencian en:
 - Almacenamiento en un formato FIJO o VARIABLE
 - Longitud MAXIMA que puede almacenar
 - Tipo de Datos TEXTO o BINARIO
- Los tipos de datos:
 - **CHAR**, **VARCHAR**, **TINYTEXT**
 - **TEXT**, **MEDIUMTEXT**, **LONGTEXT**

Categorías de datos en MySQL

CARACTER

CHAR

- Los valores se almacenan utilizando un número FIJO de bytes
- Sintaxis
 - **CHAR (M)**, donde M es la longitud de la cadena de caracteres (0 a 255)**Lenguaje CHAR(30);**
- Los caracteres no utilizados son rellenos con espacios.
- Espacios finales se borran cuando se obtienen los valores CHAR.

VARCHAR: de longitud variable (dinámico)

- Los valores se almacenan utilizando un número VARIABLE de bytes.
- Sintaxis:
 - **VARCHAR (M)**, donde M es el número máximo de caracteres
 - El valor máximo de **M** depende del conjunto de caracteres utilizado.
- El valor de cadena y la longitud de la cadena almacenada juntos

Categorías de datos en MySQL

CARACTER

- TINYTEXT**, **TEXT**, **MEDIUMTEXT** y **LONGTEXT**.
- son alternativas no estándar a VARCHAR con capacidades de almacenamiento predefinidos y diferentes.

Type	Description	Maximum Length
CHAR	M characters	255 characters
VARCHAR	L characters plus 1 or 2 bytes	65,535 characters (subject to limitations)
TINYTEXT	L + 1 bytes, where L < 2 ⁸	255 bytes
TEXT	L + 2 bytes, where L < 2 ¹⁶	65,535 bytes
MEDIUMTEXT	L + 3 bytes, where L < 2 ²⁴	16,777,215 bytes
LONGTEXT	L + 4 bytes, where L < 2 ³²	4,294,967,295 bytes

M representa la longitud de la columna declarada en caracteres.
L representa la longitud real en bytes de un valor de cadena.

Categorías de datos en MySQL

CARÁCTER. ESPECIALES

- MySQL dispone de 2 tipos de Valores CARÁCTER ESPECIALES que se utilizan en campos de valores muy concretos.
- Ambos tipos sirven para determinar los valores que puede tener un campo de una tabla
- ENUM**
 - Lista de valores de cadena enumeradas. La columna **SOLO** puede contener 1 valor
 - Hasta 65535 valores posibles (Se permite NULL pero debe de estar entre los valores)
 - Cada valor de la lista es numerado con un índice (1,2,3,4.....)
 - Sino se indica nada la columna toma el primer valor de la lista

Continent ENUM('Asia', 'Europe', 'North America'...

SET	Decimal	Bytes
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000
- SET**
 - Lista de valores de cadena no ordenadas . La columna puede contener 0, 1 o N valores.
 - Hasta 64 valores diferentes.
 - Cada valor es asignado con un valor de bit (1,2,4,8,....)

Symptom SET('sneezing', 'runny nose', 'stuffy head', 'red eyes')

Los tipos de datos ENUM y SET se representan internamente como enteros.

Character Set y Collation Support

- Cuando los sistemas procesan caracteres, utilizan los **códigos numéricos** de los caracteres en lugar de su representación gráfica.
- El proceso de asignar estos códigos numéricos se llama **codificación**.

Por ejemplo:

- cuando MySQL almacena la letra A, en realidad almacena un código numérico (65)

CONJUNTO DE CARACTERES: (Character Set)

- Un grupo de caracteres que son codificados para ser tratados (alfabéticos, ideogramas, símbolos, signos de puntuación y caracteres de control)
- El conjunto de caracteres se puede especificar en la definición de la columna mediante el atributo **CHARSET**.

Character Set y Collation Support

COLLATION:

- Es una secuencia de **CLASIFICACION** de un conjunto de caracteres.
- Esta **COLLATION**:
 - Define el orden de los carácter
 - Afecta a comparación de caracteres y cadenas.

Disponemos del atributo **COLLATE** para especificar una colación de un conjunto de caracteres para el conjunto de caracteres.

```
CREATE TABLE t
( c1 VARCHAR(20) CHARACTER SET utf8,
  c2 TEXT CHARACTER SET latin1
  COLLATE latin1_general_cs );
```

- Conjunto de Caracteres: **latin1 (UTF8)**
- Colación **latin1_general_cs**

El atributo CHARACTER SET se aplica a los tipos CHAR y VARCHAR, ENUM y SET.

Juegos de caracteres disponibles

- MySQL ofrece varios juegos de caracteres.
- Elija un conjunto adecuado para mejorar el rendimiento.
- Utilice **SHOW CHARACTER SET** para ver la lista:

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central Europe	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3
...			

Colaciones disponibles

- Un conjunto de caracteres puede tener varias agrupaciones.
 - Puede seleccionar diferentes órdenes de clasificación para el mismo conjunto de caracteres.
- Utilice **SHOW COLLATION** para ver la lista:

```
mysql> SHOW COLLATION LIKE 'latin1%';
```

Collation	Charset	Id	Default	Compiled	Sortlen
latin1_german1_ci	latin1	5			0
latin1_swedish_ci	latin1	8	Yes	Yes	1
latin1_danish_ci	latin1	15			0
latin1_german2_ci	latin1	31		Yes	2
latin1_bin	latin1	47		Yes	1
latin1_general_ci	latin1	48			0
latin1_general_cs	latin1	49			0
latin1_spanish_ci	latin1	94			0

Categorías de datos en MySQL

BINARIO

- Los tipos de datos BINARIO guardan información en formato BINARIO (bits) agrupados en ochos [octetos].
- Su representación en formato texto no tiene nada que ver con su significado real. Por lo tanto, las cadenas binarias **no tienen juegos de caracteres o colaciones**.
- Tipos de datos binarios son adecuados para almacenamiento de imágenes, multimedia, audio y código veces ejecutable.
- Disponemos de varios tipos para almacenar datos BINARIOS:
 - **BINARY** Longitud fija (estática)
 - **VARBINARY** Longitud variable (dinámico)
 - **TINYBLOB** Longitud variable
 - **BLOB** Longitud variable
 - **MEDIUMBLOB** Longitud variable
 - **LONGBLOB** Longitud variable

Categorías de datos en MySQL

BINARIO

Type	Description	Maximum Length
BINARY	M bytes	255 characters
VARBINARY	L bytes plus 1 or 2 bytes	65,535 characters
TINYBLOB	L + 1 bytes	255 bytes
BLOB	L + 2 bytes	65,535 bytes
MEDIUMBLOB	L + 3 bytes	16,777,215 bytes
LONGBLOB	L + 4 bytes	4,294,967,295 bytes

Elección correcta de tipo de datos

- El uso correcto de los tipos de datos es esencial para la integridad de la base de datos y el rendimiento.
- Considere las siguientes pautas a la hora de elegir un tipo de datos:
 - Utilizar el tipo de dato adecuado
 - No es lo mismo almacenar 9 que 09, son iguales en Numero pero no en Cadenas
 - Tenga en cuenta el tamaño máximo de los datos
 - Utilizar número con o sin SIGNO
 - Adecuar el valor máximo a almacenar mediante el tipo de datos
`EDAD → TINYINT UNSIGNED (0-255)`
 - En caracteres alfabéticos tener en cuenta datos variables o fijos
 - Podemos elegir un tipo FIJO para valores que varíen poco en su tamaño (DNI) `CHAR(10)`
 - Podemos elegir un tipo VARIABLE cuando conocemos exclusivamente su tamaño máximo (NOMBRE y APELLIDO) `VARCHAR(64)`

Uso de NULL

- `NULL` es una palabra clave de SQL se utiliza para definir los tipos de datos que permiten a los valores perdidos.
- Un valor nulo es aquel que no se puede calcular o no se conoce; los NULOS no son ni valor en blanco ni 0.

Esto significa una de dos cosas:

- **Valor Desconocido:**
 - No es un valor, pero el valor exacto no se conoce en este momento.
- **No Aplicable**
 - No se pueden hacer comparaciones con nada → Resultado `NULL`

Utilice `NULL`

- En vez de un valor real cuando:
 - Tenga que poner «ningún valor», "valor desconocido", "valor perdido", "Fuera de rango", "no aplicable", etc

Cuando usar NULL

- Durante el diseño de base de datos, en los casos en que la información de la columna no está disponible, determinar si se permiten valores nulos.
 - Por ejemplo, la tabla Country de la base de datos contiene `world_innodb`, hay países que no tienen datos de esperanza de vida y debe permitir valores nulos.
- Para indicar si una columna permite `NULL` por defecto o no.
 - Utilice `NOT NULL` si una columna no debe permitir que los nulos, para asegurar la integridad de los datos.
- Puede aplicar `NULL` y `NOT NULL` a las definiciones de columnas existentes en el momento de su creación o posteriormente al modificarlas.
- Una clave principal no debe contener valores nulos.

Extensiones de tipo Spatial Data

MySQL admite extensiones de tipo de datos espaciales para permitir la generación, almacenamiento y análisis de **características geográficas**.

- Las extensiones siguen las especificaciones del [Open Geospatial Consortium \(OGC\)](#).
 - MySQL implementa un subconjunto del entorno "`SQL con Geometry Types`" propuesto por OGC.
- Una característica geográfica (o geoespacial) es cualquier cosa en el mundo que tiene una ubicación.
 - **Entity:** Una montaña, un estanque o una ciudad
 - **Space:** Un distrito de la ciudad o el trópico
 - **Definable location:** Un cruce (un lugar particular donde dos calles se cruzan)

Extensiones de tipo Spatial Data: Motores de almacenamiento

- Puede utilizar Spatial Data con tablas basadas en los motores de almacenamiento InnoDB, MyISAM, NDB y ARCHIVE.
 - MyISAM es compatible con índices SPATIAL y non-SPATIAL.
 - Otros motores de almacenamiento admiten índices non-SPATIAL.
- Con MyISAM, puede crear índices espaciales utilizando las siguientes sentencias:

– CREATE TABLE :

```
CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL  
INDEX(g)) ENGINE=MyISAM;
```

– CREATE INDEX :

```
CREATE SPATIAL INDEX sp_index ON geom (g) ;
```

– ALTER TABLE :

```
ALTER TABLE geom ADD SPATIAL INDEX(g) ;
```

Extensiones de tipo Spatial Data: Tipos de Datos

Tipos de datos espaciales de MySQL que corresponden a clases de [OpenGIS](#):

- Tipos de datos utilizados para valores de geometría única
 - **GEOMETRY**: Clase raíz de la jerarquía; Valores de cualquier tipo
 - **POINT**: Ubicación única en el espacio de coordenadas
 - **CURVE**: Geometría unidimensional; Una secuencia de puntos
 - **LINESTRING**: Curva con interpolación lineal entre puntos
 - **SURFACE**: Geometría bidimensional
 - **POLYGON**: Plano que representa una geometría multisided
- Tipos de datos utilizados para contener colecciones de valores de geometría
 - **MULTIPOINT**: Elementos de punto
 - **MULTICURVE**: Elementos de la curva
 - **MULTILINESTRING**: Elementos LineString
 - **MULTISURFACE**: Elementos de superficie
 - **MULTIPOLYGON**: Elementos poligonales
 - **GEOMETRYCOLLECTION**: Geometrías de cualquier clase

Extensiones de tipo Spatial Data: Relleno de columnas espaciales

- Inserte valores de geometría en una tabla convirtiendo Well-Known Text (**WKT**) en formato de geometría interna.
- Realice la conversión directamente en la instrucción **INSERT** :

```
INSERT INTO geom VALUES  
  (GeomFromText('POINT(1 1)'));  
  
SET @g = 'POINT(1 1)';  
  
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Realice la conversión antes de la instrucción **INSERT** :

```
SET @g = GeomFromText('POINT(1 1)');  
  
INSERT INTO geom VALUES (@g);
```

- MySQL proporciona varias funciones que toman como argumentos un **BLOB** que contiene una representación **WKB** y un **SRID** opcional y devuelven la geometría correspondiente.

Tablas

Introducción

- Una de las operaciones habituales es la utilización de una Base de Datos para almacenar/crear nuestras tablas.
- Si el administrador crea la base de datos podremos utilizarla para tal fin, sino, tendremos que crearla nosotros mismos (siempre que tengamos privilegios para ello)
- En este capítulo se intenta dar una visión global de la creación de las estructuras básicas para poder trabajar con ellas.
- En capítulos posteriores se tratará mas en profundidad estos comandos de MySQL para ver todas las opciones disponibles.

Creación de Tablas

- Otras de las operaciones básicas que haremos habitualmente, es consultar las tablas que tengamos en nuestro sistema.
- Para ello, deberemos disponer de dichas tablas, creadas por nosotros mismos o por el Administrador.
- Para poder crear tablas en MySQL deberemos:
 - Disponer de los privilegios apropiados para ello.
 - Privilegio de **CREATE**
 - Realizar la creación dentro de una Base de Datos existente en el Gestor.
 - Mediante la orden **mysql> USE Base_de_datos**
 - Definir el MOTOR de MySQL a utilizar para la creación
 - MySQL dispone de diferentes Gestores de Almacenamiento.
 - Sino definimos nada utiliza el Motor por defecto (InnoDB)

Creación de Tablas

- A continuación os mostramos la sintaxis general de creación de tablas en MySQL.

```
CREATE TABLE <table_name> (  
    <column_name> <column_type> [<column_options>]  
    ,...  
    [, <primary key definition>]  
) [<table_options>;
```

- `CREATE TABLE Nombre_de_la_tabla (`
 - Tras el paréntesis, debemos definir las columnas y restricciones de la misma
- `<columna_nombre> <tipo> [restricciones]`
 - Definimos el nombre de las columnas y el tipo de las misma
 - A este nivel podemos poner las restricciones de la columna
 - También podemos poner Claves y Check
- `[PK, FK, ...]`
 - Claves de que consta la tabla, tanto Primarias, Foraneas, etc.

Recordar:
Hay que elegir la BBDD
antes de ejecutar la orden
anterior, sino no se creará

Creación de Tablas

```
CREATE TABLE CountryLanguage (  
    CountryCode CHAR(3) NOT NULL,  
    Language CHAR(30) NOT NULL,  
    IsOfficial ENUM('True','False') NOT NULL DEFAULT  
    'False',  
    Percentage FLOAT(3,1) NOT NULL,  
    PRIMARY KEY(CountryCode, Language)  
    ENGINE = InnoDB COMMENT='Lists Language Spoken';
```

- Nombre de la Tabla
 - `CountryLanguage`
- Columnas
 - `CountryCode` Tipo: CHAR (3) No permite Nulos
 - `Language` Tipo: CHAR (30) No permite Nulos
 - `IsOfficial` Tipo: ENUM No permite Nulos Default: TRUE
 - `Percentage` Tipo: FLOAT No Permite Nulos
- Claves
 - `Clave Primary columnas (CountryCode y Language)`

Recordar:
Hay que elegir la BBDD
antes de ejecutar la orden
anterior, sino no se creará

Propiedades de las Tablas

- Podemos definir una serie de opciones en la creación de la Tabla.
- Las opciones de la tabla deberemos definirlas después del paréntesis de cierre derecho de la creación de la tabla.
- Las opciones tienen la siguiente sintaxis:
`<Nombre_Opcion> [=] < Valor >`

El formato y el rango del valor dependen de la opción específica.

Puede agregar varias opciones de la tabla

```
CREATE TABLE CountryLanguage (  
    ...  
    )ENGINE = InnoDB COMMENT 'Lists Language  
Spoken' CHARSET utf8 COLLATE  
utf8_unicode_ci;
```

Propiedades de las Tablas

Opciones:

- **ENGINE [=] <engine_name>**
 - Indica el motor de Almacenamiento donde se crea la tabla
 - INNODB, MYISAM, MEMORY, MERGE, etc
- **COMMENT [=] '<comment>'**
 - Comentario asociado a la Tabla
- **[DEFAULT] [CHARACTER SET | CHARSET] [=] <character set>**
 - Indica el Character Set de cualquier columna String que no lo tenga definido de forma directa.
- **[DEFAULT] COLLATE [=] <collation_name>**
 - Indica la Colación de cualquier columna String que no lo tenga definido de forma directa.

```
CREATE TABLE CountryLanguage (  
    ...  
    )ENGINE = InnoDB COMMENT 'Lists Language  
Spoken' CHARSET utf8 COLLATE  
utf8_unicode_ci;
```

Opciones de la Columnas

- Toda tabla debe de tener, al menos, una columna.
- Estas columnas pueden ser definidas con diferentes opciones para ser tratadas de forma especial:
 - **NULL**
 - Es la opción por defecto y permite el almacenamiento de valores NULL.
 - **NOT NULL**
 - No permite almacenar valores NULOS
 - Estas columnas permiten ahorrar tiempo de respuesta y en algunos casos espacio de Disco
 - **DEFAULT <Valor>**
 - Si no se proporciona un valor diferente, el valor DAFAULT es almacenado.
 - **AUTO_INCREMENT**
 - Define una columna auto_numerica, por lo que sólo es válida en columnas INTEGER.
 - El uso de NULL o no indicar valor → provoca una inserción automática del siguiente número disponible en secuencia.
 - Sólo se permite una sola columna de este tipo por tabla.

Opciones de la Columnas

Example:

```
CREATE TABLE City (  
  ID int(11) NOT NULL AUTO_INCREMENT,  
  Name char(35) NOT NULL DEFAULT '',  
  CountryCode char(3) NOT NULL DEFAULT '',  
  District char(20) NOT NULL DEFAULT '',  
  Population int(11) NOT NULL DEFAULT '0',  
  PRIMARY KEY (ID)  
);
```

Describir las restricciones de tabla

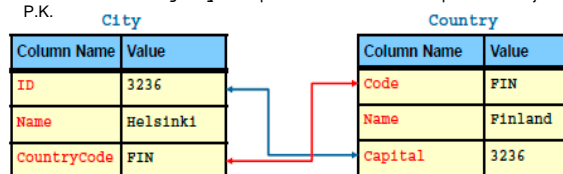
- Una restricción es una condición impuesta en uno o más valores de las columnas de una tabla para hacer cumplir las reglas de integridad.
- Habitualmente se implementan mediante cláusulas como:
 - **PRIMARY KEY**
 - **UNIQUE**
 - **FOREIGN KEY**
- Con las limitaciones establecidas, el servidor devuelve un error si se intenta modificar o eliminar datos que vayan en contra de estas restricciones.

Nota:

Sólo el motor de almacenamiento InnoDB soporta restricciones FOREIGN KEY. Otros motores de almacenamiento ignoran estas limitaciones en definiciones de tabla.

Foreign Key

- Las claves externas (**Foreign Key - FK**) se utilizan como claves que se refieren a Claves Primarias en otras Tablas.
- Utilizaremos las **Foreign Key** para implementar:
 - **Relaciones entre Filas de Datos**
 - Una **Foreign Key** es un conjunto de una o más columnas que tiene unos valores en común con otro conjunto de columnas (1 o N), por lo general en otra tabla.
 - Una **Foreign Key** crea una relación simbólica que utiliza estos valores como una referencia a la fila correspondiente.
 - **Relaciones de Integridad**
 - Toda clave **Foreign Key** solo puede contener un valor que esté reflejado en la correspondiente P.K.



Foreign Key: Sintaxis Completa

- Para definir una relación **Primary Key Foreign Key** debemos:

```
[CONSTRAINT [name]]
FOREIGN KEY [name] (referencing_coll[,...,
referencing_colN])
REFERENCES referenced_tab (referenced_coll[,...,
referenced_colN])
[ON DELETE {CASCADE | NO ACTION | RESTRICT | SET NULL}]
[ON UPDATE {CASCADE | NO ACTION | RESTRICT | SET NULL}]
```

- **Columnas Referencia-Referenciadas (Primary Key)**
 - Para cada columna de referencia, debe proporcionar exactamente una columna coincidente de la tabla referenciada.
- **Tipos de datos idénticos**
 - Los tipos de datos de cada par de columnas referencia-referenciadas (PK) deben ser idénticos, incluyendo cualquier UNSIGNED, conjunto de caracteres y atributos COLLATE.
- **Columnas Referenciadas – Primary Key**
 - En la mayoría de las implementaciones, las columnas referenciadas forman una restricción **PRIMARY KEY**.
 - Esto no es obligatorio, pero es habitual

Foreign Key: Sintaxis Completa - Opciones

Opciones dentro de las Foreign Key.

- Opciones de **ON DELETE / ON UPDATE**
 - **CASCADE**
 - Eliminar o Actualiza la fila de la tabla padre, y automáticamente eliminar o actualizar las filas coincidentes en la tabla secundaria
 - **NO ACTION**
 - El servidor MySQL rechaza la operación de eliminación o actualización para la tabla primaria si hay un valor de clave externa relacionado en la tabla referenciada.
 - **RESTRICT**
 - Similar a **NO ACTION**.
 - Rechaza la operación de eliminación o actualización de la tabla principal. Es idéntico a omitir la clausula **ON DELETE / ON UPDATE**
 - **SET NULL**
 - Eliminar o actualizar la fila de la tabla padre, y establecer la columna de clave externa o columnas de la tabla secundaria en NULL

```
CREATE TABLE parent (
  id INT NOT NULL,
  PRIMARY KEY (id)
) ENGINE=INNODB;
```

```
CREATE TABLE child (
  id INT,
  parent_id INT,
  INDEX par_ind (parent_id),
  FOREIGN KEY (parent_id)
  REFERENCES parent (id)
  ON DELETE CASCADE
) ENGINE=INNODB;
```


Utilizando el comando SHOW CREATE TABLE

Ver la declaración exacta que se puede utilizar para crear una tabla.

Ejemplo:

```
mysql> SHOW CREATE TABLE City\G
***** 1. row *****
      Table: City
Create Table: CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Opciones Creación de Tablas. Tabla Existentes

- Anteriormente hemos visto como crear una tabla utilizando la orden **CREATE TABLE**.
- Hemos teniendo que definir sus: columnas, tipos de datos, tamaños, etc.
- ¿Qué ocurre si queremos hacer una copia exacta de una Tabla con diferente nombre pero con las mismas características de sus columnas?
- Podemos utilizar la **CREATE TABLE** con un **SELECT** sobre una tabla (s) existente.

new table name
↓
CREATE TABLE CityCopy1 SELECT * FROM City;

- Esta tabla se crea con las mismas columnas que la tabla existente

Opciones Creación de Tablas. Tabla Existentes

- EJEMPLO

```
mysql> CREATE TABLE EU_Countries
-> SELECT Name,
-> Population * 1.5 AS NewPopulation
-> FROM Country
-> WHERE Continent = 'Europe';
Query OK, 46 rows affected (0.42 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

new table name

column alias

Características:

- La Sentencia **SELECT** puede hacer referencia a 1 o mas TABLAS.
- La Tabla **EU_Countries** es creada con los valores resultantes de la **SELECT**.
- La nueva Tabla **NO INCLUYE**: **INDICES**, **RESTRICCIONES**, etc
- Podemos cambiar el nombre de las columnas y su tipo, indicándolos antes de la **SELECT**
 - `CREATE TABLE bar (m INT) SELECT n FROM foo;`

Opciones Creación de Tablas. Tabla Existentes

- Los ejemplos de tablas Creación de la tabla city

- Una tabla que contiene todas las filas:

```
CREATE TABLE CityCopy1 SELECT * FROM City;
```

new table name

- Una tabla que contiene todas las ciudades con una población de más de 2.000.000:

```
CREATE TABLE CityCopy2 SELECT * FROM City
WHERE Population > 2000000;
```

- Una copia vacía de la tabla:

```
CREATE TABLE CityCopy3
SELECT * FROM City LIMIT 0;
```

Opciones Creación de Tablas. Tabla Existentes

Ejemplos adicionales de la tabla City

- Una nueva tabla con columnas única seleccionados de todas las filas:

```
mysql> CREATE TABLE CityCopy4
-> SELECT Name, CountryCode FROM City;
Query OK, 4079 rows affected (0.17 sec)
Records: 4079 Duplicates: 0 Warnings: 0
```

- El contenido de la nueva tabla:

```
mysql> SELECT * FROM CityCopy4;
+-----+-----+
| Name          | CountryCode |
+-----+-----+
| Kabul         | AFG         |
| Qandahar      | AFG         |
| Herat         | AFG         |
| Mazar-e-Sharif | AFG         |
| Amsterdam     | NLD         |
...
```

Opciones Creación de Tablas. Copia Estructura

- También podemos copiar EXCLUSIVAMENTE la estructura de una Tabla en otra nueva.
- Esta operación copia todo EXCEPTO DATOS.
 - Estructura, Índices, Restricciones, etc
- Utilice **CREATE TABLE** con la palabra clave **LIKE**

```
mysql> CREATE TABLE NewCity
-> LIKE City;
Query OK, 0 rows affected (0.05 sec)
```

Comparando SELECT Versus LIKE

- Estructura de la tabla **CityCopy3** con la sentencia **SELECT** :

Vacia

```
mysql> SHOW CREATE TABLE CityCopy3\G
...
Create Table: CREATE TABLE `citycopy3` (
  `ID` int(11) NOT NULL DEFAULT '0',
  ...
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

- Estructura de la tabla **NewCity** con la sentencia **LIKE**:

Vacia

```
mysql> SHOW CREATE TABLE NewCity\G
...
Create Table: CREATE TABLE `newcity` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  ...
  PRIMARY KEY (`ID`),
  KEY `CountryCode` (`CountryCode`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Opciones Creación de Tablas. Tabla Temporal

- Las tablas temporales son un conjunto de Tablas especiales, utilizadas para usos concretos.
- Su misión es guardar **DATOS TEMPORALES** que no son necesarios mantenerlos en el Sistema:

Características:

- Existe sólo durante la sesión del cliente. Desaparecen los DATOS + ESTRUCTURA
- Sólo son visibles por el cliente que la creó.
- No afecta a otros clientes que utilizan los mismos datos.
- Pueden ser utilizadas para almacenar los datos de resumen.

```
mysql> CREATE TEMPORARY TABLE EU_CountriesTemp
-> SELECT Name, Population * 1.5
-> AS NewPopulation
-> FROM Country
-> WHERE Continent = 'Europe';
Query OK, 46 rows affected (0.42 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

Mostrar información de las Tablas

Una vez creada nuestra primera tabla podemos obtener información sobre:

- Tablas existentes en la Base de Datos que estamos usando

`Mysql> SHOW TABLES;`

- Información sobre la estructura de la Tabla (columnas y tipos)

`Mysql> DESCRIBE Nombre_Tabla;`

- Información sobre COMO se ha creado la tabla

`Mysql> SHOW CREATE TABLE Nombre_tabla;`

```
mysql> show tables;
+-----+
| Tables_in_world_innodb |
+-----+
| city                    |
| city1                   |
| country                 |
+-----+
```

```
mysql> describe city;
+-----+-----+-----+
| Field | Type | Null |
+-----+-----+-----+
| ID    | int(11) | NO   |
| Name  | char(35) | NO   |
| CountryCode | char(3) | NO   |
| District | char(20) | NO   |
| Population | int(11) | NO   |
+-----+-----+-----+
```

```
mysql> SHOW CREATE TABLE City\G
***** 1. row *****
Table: `City`
Create Table: CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY (`ID`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Información de las Columnas de una Tabla

- Como hemos podido ver en el comando anterior de Visualización de creación, las columnas de las tablas tienen diferentes opciones.
- Estas opciones modifican, cómo MySQL trata la columna asociada.

Opciones	Descripción
NULL	- Permiten Valores NULOS
NOT NULL	- No permite valores NULOS - Normalmente aumenta el rendimiento
DEFAULT	- Si el usuario no proporciona un valor, el valor especificado se almacena.
AUTO_INCREMENT	- Se utiliza para INTEGER sólo las columnas de tipos de datos - Si no se especifica un valor o asignar un valor nulo a esta columna, el siguiente número disponible en la secuencia se inserta automáticamente. - Uno sólo por tabla
ENGINE	- Motor del Gestor usado para crear la Tabla
CHARACTER SET	- Conjunto de Caracteres por Defecto de la TABLA

Eliminar una tabla

- Para eliminar una tabla dentro del Gestor MySQL utilizaremos la Orden **DROP TABLE**.

Sintaxis

```
DROP TABLE table1, table2, table3
```

- En una misma orden pueden ser eliminadas de 1 a N tablas.

Características:

- La tabla puede estar vacía o contener datos.
- Es una operación que no tiene «marcha atrás», es decir, no se puede anular.
- Necesitamos privilegios de **DROP** para ejecutarla.
- Si al tabla no existe, nos produciría un error (**Clausula ... IF EXISTS...**)

```
DROP TABLE IF EXISTS table1
```

Eliminar una tabla Temporal

- También podemos borrar **tablas temporales**

Sintaxis

```
DROP TEMPORARY TABLE table1_temp
```

- En una misma orden pueden ser eliminadas de 1 a N tablas temporales

Características:

- Sólo realiza el borrado de **TABLAS TEMPORALES**, no es posible el borrado de tablas **NORMALES**.
- No finaliza transacciones en curso en dichas tablas.
- No comprueba derechos de usuario, pues sólo son visibles para quien la creó.

Agregar y eliminar columnas a la tabla

- En los desarrollos actuales, el Modelo Relacional de Datos es la base de la implementación de las Bases de Datos.
- Habitualmente no se realizan cambios en el mismo, pero el cliente «cambia de parecer».
- Necesitamos tener la posibilidad de agregar o eliminar columnas a las tablas que ya están creadas.
- Utilizaremos la orden **ALTER TABLE** para realizar operaciones con tablas ya creadas:
 - Agregar o quitar una columna
 - Agregar o quitar un índice
 - Cambiar una definición de columna existente

Agregar columnas a la tabla

- Utilice **ALTER TABLE Nombre ADD COLUMN [propiedades]** para añadir la columnas con las características requeridas.
- Utilizaremos la misma sintaxis que cuando definimos las Tablas y las Columnas por primera vez

Columna Tipo (tamaño) [restricciones]

```
mysql> ALTER TABLE EU_Countries
-> ADD COLUMN ID INT NOT NULL;
Query OK, 46 rows affected (0.11 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

- Los nombres de columna dentro de una tabla deben ser únicos y no se distingue entre mayúsculas y minúsculas

```
mysql> DESC EU_Countries;
```

Field	Type	Null	Key	Default	Extra
Name	char(52)	NO			
NewPopulation	decimal(12,1)	NO		0.0	
ID	int(11)	NO		NULL	

```
3 rows in set (0.00 sec)
```

Agregar columnas a la tabla

- Siga las siguientes indicaciones cuando quiera añadir una columna a una Tabla:
 - Si la tabla NO tiene datos, la nueva columna puede permitir o no **NULLS**
 - Si la tabla ya contiene datos:
 - Sino indicamos nada → **NULLS**
 - Si la columna no permite **NULLS** → Valor Implícito por Defecto del tipo de dato (blanco o 0)
 - Si definimos un valor **DEFAULT**, se rellena con dicho valor

```
mysql> SELECT * FROM EU_Countries;
```

Name	NewPopulation	Id
Albania	5101800.0	0
Andorra	117000.0	0
Austria	12137700.0	0
Belgium	15358500.0	0
Bulgaria	12286350.0	0
Bosnia and Herzegovina	5958000.0	0
Belarus	15354000.0	0
...		

Agregar columnas a la tabla. Ejemplos

Ejemplo:

```
ALTER TABLE NewCity ADD COLUMN LocalName VARCHAR(35)
CHARACTER SET utf8 NOT NULL DEFAULT '' COMMENT 'The
local name of this City';
```

La estructura se cambia de la siguiente:

```
mysql> DESC NewCity;
```

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	auto_increment
Name	char(35)	NO			
...					
Population	int(11)	NO		0	
LocalName	varchar(35)	NO			

LocalName se añade como la última columna.

Agregar columnas a la tabla

Ejemplos

```
mysql> alter table t add column c2 varchar(20);
Query OK, 0 rows affected (2.63 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> select * from t;
+----+-----+
| c  | c2    |
+----+-----+
| Hello | NULL  |
| hello | NULL  |
| Good  | NULL  |
| good  | NULL  |
+----+-----+
```

```
mysql> alter table t add column c4 varchar(20) not null;
Query OK, 0 rows affected (0.55 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> select * from t;
+----+-----+-----+-----+
| c  | c2    | c3    | c4    |
+----+-----+-----+-----+
| Hello | NULL  | HOLA  |      |
| hello | NULL  | HOLA  |      |
| Good  | NULL  | HOLA  |      |
| good  | NULL  | HOLA  |      |
+----+-----+-----+-----+
```

```
mysql> alter table t add column c3 varchar(20) not null default 'HOLA';
Query OK, 0 rows affected (1.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> select * from t;
+----+-----+-----+
| c  | c2    | c3    |
+----+-----+-----+
| Hello | NULL  | HOLA  |
| hello | NULL  | HOLA  |
| Good  | NULL  | HOLA  |
| good  | NULL  | HOLA  |
+----+-----+-----+
```

Eliminar columnas de la tabla

- Utilice **ALTER TABLE Nombre DROP COLUMN [nombre]** para eliminar la columna indicada.

```
mysql> ALTER TABLE EU_Countries
-> DROP COLUMN ID;
Query OK, 46 rows affected (0.11 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

- Si una tabla contiene sólo una columna, no se puede eliminar la columna, sería lo equivalente a borrar la Tabla → **DROP TABLE**.
 - No borre una columna de una tabla si es una clave principal (*Se puede hacer*)
 - No se puede eliminar una columna que es una clave externa que hace referencia otra tabla.

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- Se pueden borrar varias columnas a la vez en la misma orden

Eliminar columnas de la tabla. Ejemplo

Ejemplo:

```
ALTER TABLE NewCity DROP COLUMN LocalName;
```

La estructura se cambia de la siguiente manera:

```
mysql> DESC NewCity;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| ID         | int(11)   | NO   | PRI | NULL    | auto_increment |
| Name      | char(35)  | NO   |     |         |                 |
| ...       |           |      |     |         |                 |
| Population | int(11)   | NO   |     | 0       |                 |
+-----+-----+-----+-----+-----+-----+
```

LocalName ha sido eliminada

Modificar las columnas de tabla

- MySQL también permite modificar las características de las columnas que tengamos definidas.
- Podemos realizar modificaciones como:
 - Añadir / Quitar Restricciones de columna
 - Modificar el tipo de dato asignado a una Columna
 - Modificar el Tamaño del dato de una Columna

Utilice

```
ALTER TABLE Nombre MODIFY COLUMN nombre [ propiedades ]
```

```
mysql> ALTER TABLE EU_Countries
-> MODIFY COLUMN NewPopulation
-> INT UNSIGNED NOT NULL;
Query OK, 46 rows affected (0.11 sec)
Records: 46 Duplicates: 0 Warnings: 0
```

Modificar las columnas de tabla

Ejemplo:

```
ALTER TABLE NewCity MODIFY COLUMN
ID BIGINT NOT NULL AUTO_INCREMENT;
```

El tipo de datos de columna se cambia de la siguiente:

```
mysql> DESC NewCity;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| ID    | bigint(20) | NO   | PRI | NULL    | auto_increment |
| Name  | char(35)   | NO   |     |         |              |
| CountryCode | char(3)   | NO   | MUL |         |              |
| District | char(20)  | NO   |     |         |              |
| Population | int(11)  | NO   |     | 0       |              |
+-----+-----+-----+-----+-----+-----+
```

ID ha sido cambiado por el tipo **BIGINT**

Cambiando Columnas

La segunda manera de modificar una definición de columna es utilizar una cláusula **CHANGE COLUMN**.

```
ALTER TABLE NewCity CHANGE COLUMN
Name Name CHAR(40) NOT NULL;
```

El tipo de datos de columna se cambia de la siguiente :

```
mysql> DESC NewCity;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| ID    | bigint(20) | NO   | PRI | NULL    | auto_increment |
| Name  | char(40)   | NO   |     |         |              |
| CountryCode | char(3)   | NO   | MUL |         |              |
| District | char(20)  | NO   |     |         |              |
| Population | int(11)  | NO   |     | 0       |              |
+-----+-----+-----+-----+-----+-----+
```

Name ha sido cambiado al tipo de dato **CHAR (40)**

Modificar las columnas de tabla

- Al modificar una columna de tabla deberemos ponerle las nuevas características a las ya existentes.
 - Si la columna *NewPopulation* esta definida como **DECIMAL NOT NULL** DEBEREMOS modificarla como
..... NewPopulation INT UNSIGNED NOT NULL
- No se puede modificar una columna si es **Primary Key** o **Foreing Key**
- Deberemos tener cuidado con el **truncado** de las columnas cuya redefinición disminuya el tamaño original

Añadir / Eliminar índices y restricciones

- MySQL dispone de la posibilidad de añadir índices y restricciones a las tablas ya creadas.
- Utilice **ALTER TABLE Nombre ADD [opciones]**
 - Podemos añadir:
 - Índices
 - Índices únicos
 - Restricciones de Integridad

```
ALTER TABLE table_name ADD INDEX [index_name]
(index_columns)

ALTER TABLE table_name ADD UNIQUE [index_name]
(index_columns)

ALTER TABLE table_name ADD PRIMARY KEY
(index_columns)
```

Añadir / Eliminar índices y restricciones

`ALTER TABLE Nombre ADD INDEX nombre_index (columna)`

- Añadimos un índice llamado **nombre_index** sobre la columna **columna**

```
mysql> ALTER TABLE NewCity
-> ADD INDEX Pop (Population);
Query OK, 0 rows affected (0.22 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

- Para visualizarlo: `SHOW CREATE TABLE NewCity \G`

```
mysql> SHOW CREATE TABLE NewCity\G
***** 1. row *****
Table: City
Create Table: CREATE TABLE 'city' (
  'ID' int(11) NOT NULL auto_increment,
  'Name' char(35) NOT NULL default '',
  'CountryCode' char(3) NOT NULL default '',
  'District' char(20) NOT NULL default '',
  'Population' int(11) NOT NULL default '0',
  PRIMARY KEY ('ID'),
  KEY 'CountryCode' ('CountryCode'),
  KEY 'Pop' ('Population')
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Añadir / Eliminar índices y restricciones

`ALTER TABLE Nombre ADD UNIQUE (columna)`

- Añadimos un índice UNICO sobre la tabla **Nombre**

```
ALTER TABLE TABLE_NAME ADD CONSTRAINT constr_ID UNIQUE (user_id, game_id, date, time)
```

```
mysql> alter table t2 add unique (c2);
Query OK, 0 rows affected (0.58 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> show create table t2\G
***** 1. row *****
Table: t2
Create Table: CREATE TABLE 't2' (
  'c1' int(11) NOT NULL,
  'c2' varchar(20) DEFAULT NULL,
  PRIMARY KEY ('c1'),
  UNIQUE KEY 'c2' ('c2')
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)
```

Expresiones SQL

Introducción a las expresiones

- Las condiciones que aparecen en las cláusulas `WHERE` y `HAVING` están compuestas de expresiones SQL.
- Las expresiones SQL son utilizadas para devolver, habitualmente, valores booleanos (`true`, `false`) por lo que son utilizadas en condicionales.
- Dichas expresiones SQL están formadas por
 - **Elementos**
 - Los elementos como:
 - Literales (números, cadenas, fechas y horas)
 - Constantes como `NULL`, `TRUE`, y `FALSO`
 - Referencias a Columnas de las Tablas
 - Funciones
 - **Operador**
 - Operadores de Comparación, Aritméticos, Lógicos, etc

Introducción a las expresiones

Las expresiones pueden aparecer en muchos lugares:

- En la sintaxis de la `SELECT`
- En la cláusula `ORDER BY`
- En la cláusula `GROUP BY`
- Cláusulas `WHERE` y `HAVING`

Introducción a las expresiones

Ejemplos:

```
SELECT Name, Population FROM Country;

SELECT 14, -312.82, 4.32E-03, 'I am a string';

SELECT CURDATE(), VERSION();
```

Operar con valores en la Select sin BBDD

```
mysql> SELECT Name,
  TRUNCATE(Population/SurfaceArea,2)
  AS 'people/sq. km',
  IF(GNP > GNPold,'Increasing','Not increasing')
  AS 'GNP Trend'
  FROM Country ORDER BY Name;
```

Invocación de FUNCIONES

EXPRESIONES complejas

Name	people/sq. km	GNP Trend
Afghanistan	34.84	Not increasing
Albania	118.31	Increasing
Algeria	13.21	Increasing
...		

Expresiones numéricas

Cuando hablamos de expresiones numéricas, distinguimos 2 tipos de expresiones:

- **Valor Exacto** (0,00214, -43, 368.93)
 - Habitualmente usados junto a sentencias SQL.
 - Se escriben como valores enteros o decimales, sin exponente. (INT, DECIMAL)
 - No está sujeto a la inexactitud producido por error de redondeo
- **Valor Aproximado** (-4.3E1, 3.6893E2 y 2.14e-3)
 - No se utiliza habitualmente como se especifica en las instrucciones SQL
 - Son escritos en formato con notación científica, con un exponente (DOUBLE)
 - Sujetos a errores de redondeo
- Casi todas las expresiones numéricas que incluyen una parte que se evalúa como NULL se evalúan como NULL.

Expresiones numéricas

- El resultado de una operación de comparación utilizando números puede variar en función de si utiliza valores exactos o aproximados.

```
mysql> SELECT 1.1 + 2.2 = 3.3, 1.1E0 + 2.2E0 = 3.3E0;
+-----+-----+
| 1.1 + 2.2 = 3.3 | 1.1E0 + 2.2E0 = 3.3E0 |
+-----+-----+
| 1               | 0                       |
+-----+-----+
```

- La representación interna de los números de punto flotante crea la posibilidad de error de redondeo.

Si se mezclan números con cadenas en un contexto numérico

```
mysql> SELECT 1 + '1', 1 = '1';
+-----+-----+
| 1 + '1' | 1 = '1' |
+-----+-----+
| 2       | 1       |
+-----+-----+
```

MySQL convierte las cadenas en números y realiza una operación numérica

Expresiones String

- MySQL permite la utilización de comillas simple (') y comillas dobles (") en las expresiones `String`.
- La interpretación de las comillas, dependerá del `sql_mode` establecido.
- Si `sql_mode=ANSI_QUOTES`
 - MySQL interpreta las comillas dobles como caracteres normales y no se pueden utilizar para delimitar cadenas de caracteres (de acuerdo con el estándar SQL).

Una recomendación:

- Utilización de las comillas simples para indicar literales de cadena.
- Esta notación es portable a través de todas las herramientas de RDBMS, pero también independiente del modo SQL.

Expresiones String

- Se puede utilizar los siguientes tipos de datos de cadena:
`CHAR`, `VARCHAR`, `TEXT` y `BLOB`.
- Podemos utilizar expresiones de cadena en operaciones de comparación y con funciones
 - Operadores de comparación (`=`, `<>`, `<`, `BETWEEN...AND`)

```
mysql> SELECT CONCAT('abc','def',REPEAT('X',3));
+-----+
| CONCAT('abc','def',REPEAT('X',3)) |
+-----+
| abcdefXXX                          |
+-----+
```

Expresiones String

- Por defecto, MySQL trata el operador `||` como el operador lógico **OR**.
- Este comportamiento no es el mismo que para el estándar SQL (concatenación), pero podemos activarlo indicando el `sql_mode` correspondiente:

```
mysql> SET sql_mode = 'PIPES_AS_CONCAT';
Query OK, 0 rows affected (0.0 sec)
mysql> SELECT 'abc' || 'def';
+-----+
| 'abc' || 'def' |
+-----+
| abcdef         |
+-----+
```

```
mysql> SELECT 'abc' || 'def';
+-----+
| 'abc' || 'def' |
+-----+
| 0               |
+-----+
```

MySQL convierte las cadenas en números para hacer el **OR** Lógico
Sino se puede MySQL las convierte en cero

Expresiones String

- Cuando hay comparaciones de igualdad en las cadenas (`=`), debemos de tener en cuenta una serie de elementos
 - MAY/MIN
 - Acentos, etc
- Estas comparaciones dependen de:
 - Character set **latin1**
 - Collation **latin1_swedish_ci**

La collation pueden ser **non-case-sensitive** o **case-sensitive**
`_ci` -> **non_case_sensitive**

Si las cadenas son **iguales** => 1
Si las cadenas son **diferentes** => 0

Podremos definir estas variables de conexión mediante las variables:

```
character_set_connection = utf8  
collation_connection = utf8_general_ci
```

Expresiones String

Ejemplos

- **ü** y **ue** son diferentes en la colación `latin1_swedish_ci`, pero con la colación `latin1_german2_ci` son idénticos.

```
mysql> SELECT 'Hello' = 'hello';
+-----+
| 'Hello' = 'hello' |
+-----+
| 1                |
+-----+

mysql> SELECT 'Müller' = 'Mueller';
+-----+
| 'Müller' = 'Mueller' |
+-----+
| 0                    |
+-----+
```

Expresiones Regulares

- El operador `LIKE` ofrece una poderosa manera de comparar expresiones de cadena utilizando un patrón.

```
mysql> SELECT Name FROM Country
-> WHERE Name LIKE '%United%';
```

```
+-----+
| Name                |
+-----+
| United Arab Emirates |
| :                   |
| United States       |
+-----+
```

```
mysql> SELECT Name FROM Country
-> WHERE Name NOT LIKE 'United%';
```

```
+-----+
| Name                |
+-----+
| Aruba               |
| ...                 |
| Zimbabwe            |
+-----+
```

Expresiones Regulares

- Sin embargo, hay un operador de coincidencia de patrones aún más potente: **RLIKE** (o **REGEXP**).
 - Como es el caso con **LIKE**, **RLIKE** necesita dos elementos:
Cadena **RLIKE** patrón
 - Si la Cadena sigue el patrón indicado devolverá **TRUE** sino **FALSE**
 - La diferencia entre **LIKE** y **RLIKE** es que el patrón de **RLIKE** puede ser una expresión regular.

```
mysql> SELECT Name FROM Country
-> WHERE Name RLIKE '^United';

+-----+
| Name                               |
+-----+
| United Arab Emirates              |
| United Kingdom                    |
| United States Minor Outlying Isl. |
| United States                     |
+-----+
```

Expresiones Regulares

- Con las expresiones regulares, podemos utilizar patrones complejos, utilizados para comprobar:
URLs, Direcciones IP, Correos Electrónicos, Códigos Postales, Tarjetas de Crédito

- Patrones:

Patrón	Definición
Texto	Texto exacto
Punto (.)	Coincide con cualquier carácter individual.
*, ?, +, { m,n }	Especifique el número de ocurrencias.
Pipe ()	Coincidir con cualquiera de las dos alternativas
Escape (\)	Eliminación de carácter especial
^, \$ [...]	Posiciones especiales (inicio, fin) o caracteres determinados
[[:conjunto:]]	Elementos de una colección :digit: :
[a – z]	Rangos

Expresiones Regulares

- Simple (todas las aparicions posibles):

```
mysql> SELECT Name FROM City
-> WHERE Name RLIKE 'nat';

+-----+
| Name          |
+-----+
| Natal         |
| Cabanatuan    |
| Maunath Bhanjan |
| Minatitl  n  |
| Cincinnati    |
+-----+
```

- Que comiencen y acabe por algo determinado

```
mysql> SELECT Name FROM City
-> WHERE Name RLIKE '^new.*rk$';

+-----+
| Name          |
+-----+
| New York      |
| Newark        |
+-----+
```

Expresiones Regulares

- Alternativas

```
mysql> SELECT Name FROM City
-> WHERE Name RLIKE ' Los | Las ';

+-----+
| Name          |
+-----+
| San Nicol  s de los Arroyos |
| Santiago de los Caballeros  |
| Santo Domingo de los Colorados |
| Victoria de las Tunas      |
| San Nicol  s de los Garza   |
| Chilpancingo de los Bravo   |
| San Crist  bal de las Casas |
| East Los Angeles           |
| North Las Vegas            |
+-----+
```

- Diferentes caracteres en una posici  n:

```
mysql> SELECT Name FROM City
-> WHERE Name RLIKE ' L[ao]s ';
```

Expresiones Regulares

Ejemplos

```
SELECT CityName, StreetName FROM Addresses
WHERE PostalCode
RLIKE '^ [A-HJ-NP-Z] [0-9] {4} ([A-Z] {3}) ?$' ;
```

- Rango de caracteres (No elegimos los códigos postales argentinos):
- Letras(A..H; J..N; P..Z) para la provincia: [A-HJ-NP-Z]
- Cuatro dígitos para el municipio: [0-9] {4}
- (Opcional) Tres letras para la calle: ([A-Z] {3}) ?

Full-TEXT Search

- Búsqueda de texto completo es una de las acciones mas habituales realizadas en todo sistema de BBDD.
- Para que las búsquedas sean correctas deben de tener
 - **Precisión**
 - Una correcta precisión le dará unos resultados mas coherentes y menos «falsos positivos»
 - **Rapidez de Recuperación**
 - Una rápida recuperación indicará menos resultados encontrados y mas relevantes.
- Cuando conocemos exactamente lo que estamos buscando, podremos utilizar `LIKE` o expresiones `REGULARES`.
Número de factura, Nombre_cliente, DNI, etc
- Cuando disponemos de valores **imprecisos**, estas opciones no son muy válidas (dirección parcial, inicio de DNI, etc). → **Índice FULLTEXT**

Full-TEXT Search

La utilización de **FULLTEXT**, permite un mejor rendimiento.

- **LIKE y Expresiones Regulares**
 - Realizan **FULL SCAN** en toda la tabla. Bajo rendimiento
 - Poca flexibilidad: Es difícil poner en práctica determinadas búsquedas.
 - No existe un factor relevancia: No se puede especificar qué resultados son más relevantes.
- **FULLTEXT**
 - Requiere el índice **FULLTEXT** en las columnas de texto de búsqueda
 - Mejor rendimiento para consultas complejas
 - El índice es reconstruido de forma automática cuando cambian los datos.
 - Ordenes similar a **SQL** para búsquedas.
 - Utilizado en **CHAR**, **VARCHAR** o columnas **TEXT**
 - Tablas **InnoDB** y **MyISAM**

Full-TEXT Search

- Podemos definir un índice **FULLTEXT** en la creación de la Tabla o mediante la modificación en una Tabla.

```
ALTER TABLE film
ADD FULLTEXT (description);
```

- Para realizar una búsqueda deberemos utilizar las cláusulas **MATCH** (columna)
 - Indicamos la columna donde está creado el índice **FULLTEXT****AGAINST**(palabras a buscar)
 - Indicamos las palabras de búsqueda entre una única comilla simples y separadas por comas

```
SELECT title, description FROM film
-> WHERE MATCH(description) AGAINST ('composer, monkey');
... 164 rows in set (0.00 sec)
```

Full-TEXT Search

- Las respuestas son devueltas ordenadas en función de la relevancia obtenida.
 - Relevancia cero → No hay similitud.
- Relevancia se calcula basándose en
 - Número de palabras en la fila
 - Número de palabras únicas en esa fila
 - Número total de palabras indicadas en la búsqueda, etc
- **FULLTEXT** determina dónde comienzan y terminan las palabras mediante los caracteres " " (espacio) ", " (coma) y "." (punto)
- Habitualmente aparecen palabras que están dentro del mismo campo semántico.

Full-TEXT Search

- **FULLTEXT** puede ser utilizado con diferentes opciones dentro de la cláusula **AGAINST**:

IN NATURAL LANGUAGE MODE

Comportamiento anterior , es la opción por defecto.

IN BOOLEAN MODE

Con esta opción, algunos caracteres tienen un significado especial. Estos permiten a algunas consultas muy sofisticados. (-)

```
SELECT title, description FROM film
-> WHERE MATCH(description)
-> AGAINST ('composer -monkey' IN BOOLEAN MODE );
... 83 rows in set (0.00 sec)
```

Se buscan todas las palabras compose en description que no contengan la palabra monkey

Los resultados son devuelto sin tener en cuenta el orden de relevancia

Expresiones Temporales

Las expresiones temporales se suelen utilizar para:

- Extraer y Manipular fechas y horas
- Comparaciones columnas y tiempos
- Operaciones aritméticas de suma / resta de intervalos temporales.
- Etc.

Podemos utilizar los siguientes tipos de datos temporales:

DATE TIME DATETIME
TIMESTAMP YEAR

Los datos temporales se pueden obtener de la siguiente manera:

- Copiar datos temporales de una columna de tabla existente.
- Ejecuta una función que devuelve datos temporales.
- Construir una representación de cadena con los datos temporales.

Expresiones Temporales

- Los formatos de fecha y hora permitidos para el uso en funciones temporales:

Data Type	Default Format
DATE	YYYY-MM-DD
TIME	HH:MI:SS
DATETIME	YYYY-MM-DD HH:MI:SS
TIMESTAMP	YYYY-MM-DD HH:MI:SS
DAY	DD
MONTH	MM
QUARTER	Q
YEAR	YYYY

Los operadores de comparación utilizados son:

- = <> < BETWEEN . . . AND
- Las funciones que esperan valores de FECHA usualmente aceptan valores DATETIME e ignoran la parte de hora.
 - Las funciones que esperan valores de DATE usualmente aceptan valores DATETIME e ignoran la parte de fecha.

Expresiones Temporales: Intervalos

- Para realizar el incremento o decremento de valores temporales utilizaremos el comando `INTERVAL`.
- Nos permite aumentar o disminuir una determinada porción de TIEMPO.

... `INTERVAL n [day, minute, second, ...]`

```
mysql> SELECT '2012-01-01' + INTERVAL 10 DAY,
              -> INTERVAL 10 DAY + '2012-01-01';
+-----+-----+
| '2012-01-01' + INTERVAL 10 DAY | INTERVAL 10 DAY + '2012-01-01' |
+-----+-----+
| 2012-01-11                      | 2012-01-11                      |
+-----+-----+

mysql> SELECT '2012-01-01' - INTERVAL 10 DAY;
+-----+
| '2012-01-01' - INTERVAL 10 DAY |
+-----+
| 2011-12-22                      |
+-----+
```

Funciones en expresiones

- Dentro de las expresiones utilizadas podemos utilizar funciones (built-in o propias) para devolver un valor que se utilizará en la expresión.
- Cuando utilizamos funciones en MySQL, es obligatorio poner los paréntesis `()` justo detrás del nombre de la función. (sino daría error)

```
mysql> SELECT PI ();
ERROR 1305 (42000): FUNCTION world.PI does not exist
```

- Podemos modificar MySQL para que acepte un espacio en blanco entre el nombre de la función y los parámetros, cambiando el `sql_mode`

```
mysql> SET sql_mode = 'IGNORE_SPACE';
Query OK, 0 rows affected (0.06 sec)

mysql> SELECT PI ();
3.141593
```

Funciones en expresiones: Comparación

- Las funciones de comparación permiten comparar múltiples valores y devolver uno de ellos.

LEAST () :

- Devuelve el valor mas **pequeño** del conjunto (ordenación y no por tamaño)

GREATEST ()

- Devuelve el valor mas **grande** del conjunto (ordenación y no por tamaño)

```
mysql> SELECT LEAST(4,3,8,-1,5) , LEAST('cdef','ab','ghi');
+-----+-----+
| LEAST(4,3,8,-1,5) | LEAST('cdef','ab','ghi') |
+-----+-----+
| -1                | ab                        |
+-----+-----+

mysql> SELECT GREATEST(4,3,8,-1,5) , GREATEST('cdef','ab','ghi');
+-----+-----+
| GREATEST(4,3,8,-1,5) | GREATEST('cdef','ab','ghi') |
+-----+-----+
| 8                    | ghi                       |
+-----+-----+
```

Funciones en expresiones: Comparación

- Una de las funciones de comparación incluida en MySQL es **INTERVAL**.
- No tiene nada que ver con el incremento/decremento temporal
- La función **INTERVAL** compara el primer argumento entero con el resto de los argumentos y devuelve 0, 1, 2, etc en función de esta comparación.

`INTERVAL(<integer1>, <integer2> [{, <integer>}...])`

```
SELECT INTERVAL(10, 1, 2, 4, 8, 16);
+-----+
| INTERVAL(10, 1, 2, 4, 8, 16) |
+-----+
| 4                             |
+-----+
```

Si N1 <N2, la función devuelve 0
Si N1 <N3, devuelve 1.
Si N1 <N4, devuelve 2, y así sucesivamente.

10< 16 → 4

Los argumentos deberían ser indicados en **orden ascendente**

Funciones en expresiones: Control Flujo

- MySQL dispone también de funciones de **Control**, para poder realizar operaciones en función de condiciones.
- Sería a las sentencias **condicionales** de lenguajes de programación 4 GL.

Name	Description
<u>CASE</u>	Case operator
<u>IF ()</u>	If/else construct
<u>IFNULL ()</u>	Null if/else construct
<u>NULLIF ()</u>	Return NULL if expr1 = expr2

Funciones en expresiones: Control Flujo

Función **IF**

- Realiza la evaluación de una condición y dependiendo del resultado realiza una acción u otra.

Sintaxis

`IF (Condicion, Valor_True, Valor_False);`

```
mysql> SELECT IF(1 > 0, 'YES', 'NO');
+-----+
| IF(1 > 0, 'YES', 'NO') |
+-----+
| YES                     |
+-----+
```

```
SELECT IF(1>2,2,3);
-> 3
SELECT IF(1<2,'yes','no');
-> 'yes'
SELECT IF(STRCMP('test','test1'),'no','yes');
-> 'no'
```

Funciones en expresiones: IF () con ORDER BY

Función **IF**. Ejemplos con **ORDER BY**:

```
mysql> SELECT Name FROM Country
      -> ORDER BY IF(code='USA',1,2), Name;
+-----+
| Name |
+-----+
| United States |
| Afghanistan |
| Albania |
| Algeria |
| American Samoa |
| Andorra |
| Angola |
| Anguilla |
| Antarctica |
| Antigua and Barbuda |
| ...
```

Funciones en expresiones: IF () con NULL

Función **IF**. Ejemplos con **NULL**:

```
mysql> SELECT IF(1 > NULL, 'yes', 'no');
+-----+
| IF(1 > NULL, 'yes', 'no') |
+-----+
| no |
+-----+

mysql> SELECT IF(NULL = NULL, 'yes', 'no');
+-----+
| IF(NULL = NULL, 'yes', 'no') |
+-----+
| no |
+-----+

mysql> SELECT IF(NULL <> NULL, 'yes', 'no');
+-----+
| IF(NULL <> NULL, 'yes', 'no') |
+-----+
| no |
+-----+
```

Funciones en expresiones: IF () con SUM()

Función **IF**. Ejemplos con **SUM()**:

```
mysql> SELECT division,
-> SUM(IF(syear=2009,sale,0)) as year09,
-> SUM(IF(syear=2010,sale,0)) as year10,
-> SUM(IF(syear=2011,sale,0)) as year11,
-> SUM(IF(syear=2012,sale,0)) as year12,
-> SUM(IF(syear=2013,sale,0)) as year13
-> FROM sales GROUP BY division;
```

division	year09	year10	year11	year12	year13
A	100	140	10	20	122
B	0	0	80	100	12
C	0	120	20	200	230

Funciones en expresiones: IFNULL

Función **IFNULL(exp, valor)**

- La función **IFNULL**, comprueba la expresión, y si ésta NO ES NULL devuelve el resultado de la expresión, sino devuelve el VALOR.

```
mysql> SELECT IFNULL(1,0);
-> 1
mysql> SELECT IFNULL(NULL,10);
-> 10
mysql> SELECT IFNULL(1/0,10);
-> 10
mysql> SELECT IFNULL(1/0,'yes');
-> 'yes'
```

Funciones en expresiones: IFNULL

Función **NULLIF**(**exp1**, **exp2**)

- La función **NULLIF** devuelve **NULL** si la **expr1 = expr2** de lo contrario se devuelve **expr1**

```
mysql> SELECT NULLIF(1,1);  
      -> NULL  
mysql> SELECT NULLIF(1,2);  
      -> 1
```

Funciones en expresiones: Control Flujo

Función **CASE**

- La función **CASE** evalúa una serie de expresiones condicionales.
- Cada evaluación devuelve un resultado booleano. (TRUE / FALSE)

Sintaxis 1 Devuelve el resultado cuando **value=valor_comparado**

```
CASE value  
  WHEN <compare_value> THEN <result>  
  [WHEN <compare_value> THEN <result> ...]  
  [ELSE <result>]  
END
```

Sintaxis 2 Devuelve el resultado cuando **se cumple la primera condición**

```
CASE  
  WHEN <condition> THEN <result>  
  [WHEN <condition> THEN <result> ...]  
  [ELSE <result>]  
END
```

Funciones en expresiones: CASE

Ejemplo:

```
mysql> SELECT Name FROM Country
-> ORDER BY
-> CASE Code
-> WHEN 'USA' THEN 1
-> WHEN 'CAN' THEN 2
-> WHEN 'MEX' THEN 3
-> ELSE 4 END, Name;
+-----+
| Name          |
+-----+
| United States |
| Canada        |
| Mexico        |
| Afghanistan   |
| Albania       |
| ...          |
| Zimbabwe      |
+-----+
239 rows in set (0.0 sec)
```

Funciones en expresiones: CASE

Ejemplo:

```
mysql> SELECT
-> CASE
-> WHEN Code = 'USA' THEN 'United States'
-> WHEN Continent = 'Europe' THEN 'Europe'
-> ELSE 'Rest of the world'
-> END AS Area,
-> SUM(GNP),
-> SUM(Population)
-> FROM Country
-> GROUP BY Area;
+-----+-----+-----+
| Area          | SUM(GNP) | SUM(Population) |
+-----+-----+-----+
| Europe        | 9498865.00 | 730074600       |
| Rest of the world | 11345342.90 | 5070317850      |
| United States  | 8510700.00 | 278357000       |
+-----+-----+-----+
3 rows in set (0.0 sec)
```


Funciones numéricas

Las funciones numéricas realizan varios tipos de operaciones matemáticas.

Function Syntax	Definition
ABS (<number>)	Devuelve el valor absoluto del número
SIGN (<number>)	Devuelve -1, 0 o 1 dependiendo de si el número es negativo, cero o positivo
TRUNCATE (<number>, <decimals>)	Devuelve el número truncado a decimales
FLOOR (<number>)	Redondea el número hasta el número entero inferior más cercano
CEILING (<number>)	Redondea el número hasta el número entero superior más cercano
ROUND (<number>)	Ronda el número al número entero más cercano
SIN () , COS () , TAN () , PI () , DEGREES () , RADIANS ()	Realiza cálculos trigonométricos, incluyendo conversiones entre grados y radianes

Funciones numéricas : ABS () and SIGN ()

- Devuelve el valor absoluto de los valores negativos y positivos

```
mysql> SELECT ABS (-42) , ABS (42) ;
+-----+-----+
| ABS (-42) | ABS (42) |
+-----+-----+
|      42 |      42 |
+-----+-----+
```

- Devolver los resultados de la determinación del signo
(-1 = negative, 0 = zero, 1 = positive)

```
mysql> SELECT SIGN (-42) , SIGN (-1) , SIGN (0) , SIGN (1) ,
SIGN (42) ;
+-----+-----+-----+-----+-----+
| SIGN (-42) | SIGN (-1) | SIGN (0) | SIGN (1) | SIGN (42) |
+-----+-----+-----+-----+-----+
|    -1     |    -1     |    0     |    1     |    1     |
+-----+-----+-----+-----+-----+
```

Funciones numéricas : FLOOR () y CEILING ()

- Devuelve el entero menor mas cercano al argumento:

```
mysql> SELECT FLOOR(-14.7) , FLOOR(14.7) ;
+-----+-----+
| FLOOR(-14.7) | FLOOR(14.7) |
+-----+-----+
| -15          | 14          |
+-----+-----+
```

- Devuelve el entero mayor mas cercano al argumento:

```
mysql> SELECT CEILING(-14.7) , CEILING(14.7) ;
+-----+-----+
| CEILING(-14.7) | CEILING(14.7) |
+-----+-----+
| -14            | 15            |
+-----+-----+
```

Funciones numéricas : ROUND ()

- Redondea al entero mas cercano. Podemos poner precisión:

```
mysql> SELECT ROUND(28.5) , ROUND(-28.5) ;
+-----+-----+
| ROUND(28.5) | ROUND(-28.5) |
+-----+-----+
| 29          | -29          |
+-----+-----+
```

- 0-4 redondeo hacia abajo. 5-9 redondeo hacia arriba:

```
mysql> SELECT ROUND(2.85,1) , ROUND(2.84,1) ;
+-----+-----+
| ROUND(2.85,1) | ROUND(2.84,1) |
+-----+-----+
| 2.9           | 2.8           |
+-----+-----+
```

Funciones numéricas : TRUNCATE ()

- Trunca al número con los decimales indicados:

```
mysql> SELECT TRUNCATE (28.5) , TRUNCATE (-28.5) ;
+-----+-----+
| TRUNCATE (28.5) | TRUNCATE (-28.5) |
+-----+-----+
| 29             | -29              |
+-----+-----+
```

```
mysql> SELECT TRUNCATE (2.85,1) , TRUNCATE (2.84,1) ;
+-----+-----+
| TRUNCATE (2.85,1) | TRUNCATE (2.84,1) |
+-----+-----+
| 2.8              | 2.8               |
+-----+-----+
```

Funciones numéricas: Ejemplos trigonométricos

Realizar cálculos trigonométricos, incluyendo conversiones entre grados y radianes:

```
mysql> SELECT SIN (0) , COS (0) , TAN (0) ;
+-----+-----+-----+
| SIN (0) | COS (0) | TAN (0) |
+-----+-----+-----+
| 0       | 1       | 0       |
+-----+-----+-----+

mysql> SELECT PI () , DEGREES (PI ()) , RADIANS (180) ;
+-----+-----+-----+
| PI ()   | DEGREES (PI ()) | RADIANS (180) |
+-----+-----+-----+
| 3.141593 | 180             | 3.1415926535898 |
+-----+-----+-----+
```

Funciones de cadena

- Realizar operaciones en cadenas, tales como:
 - Cálculo de longitudes de cadena
 - Extracción de pedazos de cuerdas
 - Buscar subcadenas o reemplazarlas
 - Realización de conversión de casos
- Las funciones de cadena se dividen en dos categorías:
 - **Numerico:** Devuelven números
 - **String:** Devuelven cadenas

Funciones de cadena: Numericas (INSTR, LOCATE, POSITION)

- Devuelve la posición de una cadena específica dentro de otra cadena:

```
mysql> SELECT INSTR('Alice and Bob', 'and'),
-> LOCATE('and', 'Alice and Bob'),
-> POSITION('and' IN 'Alice and Bob')
-> \G
***** 1. row *****
INSTR('Alice and Bob', 'and'): 7
LOCATE('and', 'Alice and Bob'): 7
POSITION('and' IN 'Alice and Bob'): 7
```

- Devuelve el desplazamiento desde donde comienza la búsqueda:

```
mysql> SELECT LOCATE(' ', 'Alice and Bob', 7);
+-----+
| LOCATE(' ', 'Alice and Bob', 7) |
+-----+
| 10                               |
+-----+
```

Funciones de cadena: Numéricas (LENGTH, CHAR_LENGTH, CONVERT)

Devuelve las longitudes de cadena en bytes y unidades de caracteres, respectivamente:

```
mysql> SELECT LENGTH('MySQL'),
-> CHAR_LENGTH('MySQL');
+-----+-----+
| LENGTH('MySQL') | CHAR_LENGTH('MySQL') |
+-----+-----+
| 5               | 5                     |
+-----+-----+

mysql> SELECT LENGTH(CONVERT('MySQL' USING ucs2))
-> AS length,
-> CHAR_LENGTH(CONVERT('MySQL' USING ucs2))
-> AS c_length;
+-----+-----+
| length | c_length |
+-----+-----+
| 10     | 5        |
+-----+-----+
```

El ejemplo de la diapositiva muestra esto, utilizando el conjunto de caracteres **latin1** de un solo byte y el conjunto de caracteres **ucs2** de **varios bytes**.

Funciones de cadena: Numéricas (STRCMP)

- Devuelve la comparación de ordenación de cadenas (0 = igual, -1 = menor, 1 = otro):

```
mysql> SELECT STRCMP('abc','def'),
-> STRCMP('def','def'),
-> STRCMP('def','abc');
+-----+-----+-----+
| STRCMP('abc','def') | STRCMP('def','def') | STRCMP('def','abc') |
+-----+-----+-----+
| -1                 | 0                   | 1                   |
+-----+-----+-----+
```

- Devuelve las comparaciones: 0 → diferentes 1 → iguales

```
mysql> SELECT 'abc'
-> = 'def', 'def' = 'def', 'def' = 'abc';
+-----+-----+-----+
| 'abc' = 'def' | 'def' = 'def' | 'def' = 'abc' |
+-----+-----+-----+
| 0             | 1             | 0             |
+-----+-----+-----+
```

Funciones de cadena: String (CONCAT, CONCAT_WS, LEFT, RIGHT)

- Concatenar los argumentos dados en una cadena:

```
mysql> SELECT
  CONCAT('See','spot','run');
+-----+
| CONCAT('See','spot','run') |
+-----+
| Seespotrun                  |
+-----+
```

- Concatenar los argumentos dados con un separador:

```
mysql> SELECT
  CONCAT_WS(' ','See','spot','run');
+-----+
| CONCAT_WS(' ','See','spot','run') |
+-----+
| See spot run                     |
+-----+
```

- Devuelve los **N** caracteres a la izquierda de la cadena:

```
mysql> SELECT
  LEFT('Alice and Bob', 5);
+-----+
| LEFT('Alice and Bob', 5) |
+-----+
| Alice                    |
+-----+
```

- Devuelve los **N** caracteres a la derecha de la cadena:

```
mysql> SELECT
  RIGHT('Alice and Bob', 3);
+-----+
| RIGHT('Alice and Bob', 3) |
+-----+
| Bob                       |
+-----+
```

Funciones de cadena: String (SUBSTRING, SUBSTRING_INDEX)

- Devuelve una parte de una cadena (subcadena) especificada por un inicio y número:

```
mysql> SELECT SUBSTRING('Alice and Bob', 1, 5);
+-----+
| SUBSTRING('Alice and Bob', 1, 5) |
+-----+
| Alice                             |
+-----+
```

- Devuelve la primera parte de la cadena a partir del valor:

```
mysql> SELECT
  SUBSTRING_INDEX('Alice and Bob', 'and', 1);
+-----+
| SUBSTRING_INDEX('Alice and |
| Bob', 'and', 1)           |
+-----+
| Alice                      |
+-----+
```

- Devuelve la segunda parte de la cadena a partir del valor :

```
mysql> SELECT
  SUBSTRING_INDEX('Alice and Bob', 'and', -1);
+-----+
| SUBSTRING_INDEX('Alice and |
| Bob', 'and', -1)           |
+-----+
| Bob                        |
+-----+
```

Funciones de cadena: String (LTRIM, RTRIM, TRIM)

- Eliminar los caracteres de espacio que aparecen en los extremos especificados de una cadena de argumento:

```
mysql> SELECT CONCAT('<', LTRIM(' Alice '), '>'),
-> CONCAT('<', RTRIM(' Alice '), '>'),
-> CONCAT('<', TRIM(' Alice '), '>') \G
***** 1. row *****
CONCAT('<', LTRIM(' Alice '), '>'): <Alice >
CONCAT('<', RTRIM(' Alice '), '>'): < Alice>
CONCAT('<', TRIM(' Alice '), '>'): <Alice>
```

- Eliminar espacios de ambos extremos de la cadena de argumentos:

```
mysql> SELECT TRIM(LEADING 'Cha' FROM
'ChaChaChalice');
+-----+
| TRIM(LEADING 'Cha' FROM 'ChaChaChalice') |
+-----+
| lice                                     |
+-----+
```

LEADING
Cambia los
elementos
indicados por
espacios en la
cadena

Funciones de cadena: String (INSERT, REPLACE)

- Reemplazar una subcadena particular con otra cadena:

```
mysql> SELECT REPLACE('Alice & Bob', '&', 'and');
+-----+
| REPLACE('Alice & Bob', '&', 'and') |
+-----+
| Alice and Bob                       |
+-----+
```

- Reemplazar una subcadena en una posición determinada por otra cadena :

```
mysql> SELECT INSERT('Alice and Bob', 6, 5, '
Carol & ');
+-----+
| INSERT('Alice and Bob', 6, 5, '
Carol & ') |
+-----+
| Alice, Carol & Bob                       |
+-----+
```

Funciones de cadena: Character Set y Collation

- Devolver datos sobre el conjunto de caracteres y la colación de un argumento de cadena::

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+-----+-----+-----+
| USER() | CHARSET(USER()) | COLLATION(USER()) |
+-----+-----+-----+
| root@localhost | utf8 | utf8_general_ci |
+-----+-----+-----+
```

- Convertir datos entre diferentes juegos de caracteres:
- Convierta una cadena a un conjunto de caracteres diferente:

```
mysql> SELECT
  CONVERT(_latin1'Müller' USING
  utf8);
+-----+
| CONVERT(_latin1'Müller' |
| USING utf8) |
+-----+
| Müller |
+-----+
```

```
mysql> SELECT CAST(_latin1'test' AS
  CHAR CHARACTER SET utf8);
+-----+
| CAST(_latin1'test' AS CHAR |
| CHARACTER SET utf8) |
+-----+
| test |
+-----+
```

Funciones Temporales

- Realizar operaciones tales como:
 - Extracción de partes de fechas y horas
 - Reformatando valores
 - Convertir valores en segundos o días
- Utilice más de un formato para la misma información:
 - 2012-02-10 22:50:15
 - Friday, February 10, 2012
- Genere datos temporales de muchas maneras:
 - Copiar datos existentes.
 - Ejecute la función incorporada.
 - Generar una representación de cadena para ser evaluada por el servidor

Funciones Temporales: Tipos de Funciones

Function Syntax	Definition
<code>NOW ()</code>	Hora actual configurada en el host del servidor (formato <code>TIME</code>)
<code>CURDATE ()</code>	Fecha actual como se establece en el host del servidor (formato <code>DATE</code>)
<code>CURTIME ()</code>	Hora actual configurada en el host del servidor (formato <code>TIME</code>)
<code>DATEDIFF ()</code>	Resta dos fechas
<code>DATE_ADD ()</code>	Añade valores de tiempo (<code>intervalos</code>) a un valor de fecha
<code>YEAR (<date_expression>)</code>	Año en formato de cuatro dígitos <code>YEAR</code>
<code>MONTH (<date_expression>)</code>	Mes del año en formato entero,
<code>DAYOFMONTH (<date_expression>)</code> or <code>DAY (<date_expression>)</code>	Día del mes en formato entero,
<code>DAYNAME (<date_expression>)</code>	Día de la semana en formato de cadena (Inglés)
<code>HOOR (<date_expression>)</code>	Hora del día en formato entero (en rango 0-23)
<code>MINUTE (<date_expression>)</code>	Minuto del día en formato entero
<code>SECOND (<date_expression>)</code>	Segundo del minuto en formato entero
<code>GET_FORMAT (<date_expression>)</code>	Devuelve una cadena de formato de fecha, por valores indicados para el tipo de fecha y el formato internacional

Funciones temporales: Extracción de datos (NOW, CURDATE, CURTIME, DAYNAME)

- Mostrar la fecha y la hora actuales:

```
mysql> SELECT NOW ();
+-----+
| NOW () |
+-----+
| 2013-07-08 18:02:35 |
+-----+
```

- Extraer datos temporales actuales (fecha, hora y día de la semana):

```
mysql> SELECT CURDATE (), CURTIME (), DAYNAME (NOW ());
+-----+-----+-----+
| CURDATE () | CURTIME () | DAYNAME (NOW ()) |
+-----+-----+-----+
| 2013-07-08 | 17:55:40 | Thursday |
+-----+-----+-----+
```

Funciones temporales: Extracción de datos (GET_FORMAT, DATE)

- Extraiga el formato de fecha para un tipo de datos especificado:

```
mysql> SELECT GET_FORMAT(DATE, 'EUR');
+-----+
| GET_FORMAT(DATE, 'EUR') |
+-----+
| %d.%m.%Y                |
+-----+
```

- Hay varias opciones para los dos argumentos:

```
GET_FORMAT(DATE|TIME|DATETIME,
            'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL')
```

Funciones temporales: Extracción de datos (YEAR, MONTH, DAYOFMONTH, DAYOFYEAR, HOUR, MINUTE, SECOND)

Devuelve partes especificadas de fecha y hora:

```
mysql> SELECT YEAR('2013-07-08'), MONTH('2013-07-08'),
-> DAYOFMONTH('2013-07-08');
+-----+-----+-----+
| year('2013-07-08') | month('2013-07-08') | dayofmonth('2013-07-08') |
+-----+-----+-----+
| 2013 | 7 | 8 |
+-----+-----+-----+

mysql> SELECT DAYOFYEAR('2013-07-08');
+-----+
| dayofyear('2013-07-08') |
+-----+
| 189 |
+-----+

mysql> mysql> SELECT HOUR('11:28:45'), MINUTE('11:28:45'),
-> SECOND('11:28:45');
+-----+-----+-----+
| HOUR('11:28:45') | MINUTE('11:28:45') | SECOND('11:28:45') |
+-----+-----+-----+
| 11 | 28 | 45 |
+-----+-----+-----+
```

Funciones temporales: Extracción de datos (NOW, DATE_FORMAT, INTERVAL)

- Personalizar el formato de salida de datos temporales:

```
mysql> SELECT NOW(), DATE_FORMAT(NOW(), '%W the %D of %M');
+-----+-----+
| NOW()                | DATE_FORMAT(NOW(), '%W the %D of %M') |
+-----+-----+
| 2013-07-18 11:33:24 | Thursday the 18th of July              |
+-----+-----+
```

- Añada un intervalo especificado de días a partir de la fecha y hora actuales:

```
mysql> SELECT NOW(), NOW() + INTERVAL 5 DAY;
+-----+-----+
| NOW()                | NOW() + INTERVAL 5 DAY |
+-----+-----+
| 2013-07-18 11:33:41 | 2013-07-23 11:33:41    |
+-----+-----+
```

Funciones temporales: Extracción de datos (MAKEDATE, MAKETIME)

- Devuelve la Fecha con los valores indicados (año, días):

```
mysql> SELECT MAKEDATE(2013,189);
+-----+
| MAKEDATE(2013,189) |
+-----+
| 2013-07-08          |
+-----+
```

- Devuelve la hora con los valores indicados (hora, minuto, seg)

```
mysql> SELECT MAKETIME(9,23,57);
+-----+
| MAKETIME(9,23,57) |
+-----+
| 09:23:57          |
+-----+
```

Funciones relacionadas con NULL: ISNULL y IFNULL

- Específicamente para su uso con **NULL**
- Ejemplos de **ISNULL ()** e **IFNULL ()**:

```
mysql> SELECT ISNULL(NULL), ISNULL(0), ISNULL(1);
+-----+-----+-----+
| ISNULL(NULL) | ISNULL(0) | ISNULL(1) |
+-----+-----+-----+
| 1           | 0         | 0         |
+-----+-----+-----+

mysql> SELECT IFNULL(NULL, 'a'), IFNULL(0, 'b');
+-----+-----+
| IFNULL(NULL, 'a') | IFNULL(0, 'b') |
+-----+-----+
| a                 | 0              |
+-----+-----+
```

- **IFNULL ()** es una función específica de MySQL.

Funciones relacionadas con NULL: CONCAT y CONCAT_WS

- Cualquier argumento **NULL** pasado a **CONCAT ()** hace que devuelva **NULL**.
- **CONCAT_WS ()** ignora argumentos **NULL**.
- Ejemplos de **CONCAT ()** y **CONCAT_WS ()**:

```
mysql> SELECT CONCAT('a', 'b'),
-> CONCAT('a', NULL, 'b');
+-----+-----+
| CONCAT('a', 'b') | CONCAT('a', NULL, 'b') |
+-----+-----+
| ab               | NULL                    |
+-----+-----+

mysql> SELECT CONCAT_WS('/', 'a', 'b'),
-> CONCAT_WS('/', 'a', NULL, 'b');
+-----+-----+
| CONCAT_WS('/', 'a', 'b') | CONCAT_WS('/', 'a', NULL, 'b') |
+-----+-----+
| a/b                     | a/b                           |
+-----+-----+
```

UNIONES con SELECT

- Hay veces que necesitamos combinar los resultados de varias instrucciones `SELECT` en una sola salida.
- Esta combinación es la suma de las filas de la primera `SELECT` + las filas de la segunda `SELECT`.
- No hay intervención de `JOIN`, Relaciones ni columnas, solo filas + filas.

- Sintaxis

```
SELECT column1, column2
UNION [ALL | DISTINCT]
SELECT column1, column2
UNION [ALL | DISTINCT]
SELECT ...]
```

```
mysql> SELECT id, name FROM staff_list
-> UNION
-> SELECT id, name FROM customer_list limit 4;
+----+-----+
| ID | name      |
+----+-----+
| 1  | Mike Hillyer |
| 2  | Jon Stephens |
| 218 | VERA MCCOY   |
| 441 | MARIO CHEATHAM |
+----+-----+
4 rows in set (0.00 sec)
```

UNIONES con SELECT

NOTAS:

- El número de columnas en ambas `SELECT` debe de ser el mismo.
- Las columnas de cada `SELECT` debe de tener el mismo tipo de datos.
- Los nombres de las columnas pueden ser diferentes, aparecen los nombres de la columnas de la primera `SELECT`.
- Si los tipos de datos de las columnas no coinciden, los tipos y longitudes de las columnas en el resultado `UNION` tienen en cuenta los valores recuperados por todas las sentencias `SELECT`.
- `UNION`
 - Si hay filas duplicadas sólo saca un representante
- `UNION ALL`
 - Si hay filas duplicadas saca todas las filas repetidas.

COMENTARIOS

- MySQL soporta tres sintaxis diferentes para comentarios:

Comienza un comentario que se extiende hasta el final de la línea.

Tipo unix/linux

-- Comienza un comentario que se extiende hasta el final de la línea.

/*

*/ Múltiples líneas empezar y terminar una secuencia comentario.

Tipo Lenguaje C

```
/* this is a comment */
/*
this
is a
comment,
too
*/
```

COMENTARIOS Especiales

- MySQL dispone de un tipo de comentarios permite un tratamiento especial.
- Estos comentarios estarán incluidos en declaraciones de sentencia y si el gestor es :
 - MySQL
 - El comentario será tratado como una parte de la consulta ejecutada
 - Otros Gestores
 - Será tratado como un comentario

Sintaxis

/ * ! Comentario */

```
CREATE TABLE t (i INT) /*! ENGINE = MEMORY */;
SHOW /*!50002 FULL */ TABLES;
```

El número indica la versión del Gestor a partir de la cual puede ejecutarse dicha orden.

50002 -> 5.0.2 o superior

COMENTARIOS En objetos de BBDD

- Además de los comentarios de código discutidos en la sección anterior, MySQL también dispone de comentarios en objetos de base de datos.
- **Objetos:**
 - Tablas, Columnas Procedimientos Almacenados
- **Sintaxis**
 - Los comentarios van entre comillas simples detrás de la palabra **COMMENT**.
 - El número máximo de caracteres es de 60

```
CREATE TABLE CountryLanguage (  
    ...  
    ) ENGINE=MyISAM COMMENT 'Lists Languages Spoken'
```

COMENTARIOS En objetos de BBDD

```
CREATE TABLE CountryLanguage (  
    CountryCode CHAR(3) NOT NULL  
        COMMENT 'The code that identifies the Country',  
    Language CHAR(30) NOT NULL  
        COMMENT 'The name of the language spoken in  
            the Country',  
    ...)
```

- Los comentarios se pueden recupera mediante:
 - ✓ `SHOW CREATE TABLE, o`
 - ✓ mediante la base de datos `INFORMATION_SCHEMA`.

Joins de Tablas

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Describir el concepto de `JOIN`
- Explicar la construcción y las propiedades del producto cartesiano
- Utilizar la sintaxis y la aplicación de diferentes tipos de `JOIN`
- Utilice referencias de columna calificadas y alias de tabla para evitar ambigüedad
- Enumerar los diferentes tipos de `JOIN`'s
- Ejecutar `OUTER` e `INNER JOIN`
- `SELF JOIN`
- Utilizar instrucciones `UPDATE` y `DELETE` multitables

Concepto de JOIN

- Las consultas **SELECT** mostradas hasta ahora en este curso recuperar datos de una sola tabla.
- Hay veces en las que no se puede responder a todas las preguntas de una forma tan simple.
- Es necesario combinar varias tablas para poder ofrecer respuestas.
- ¿Qué es un **JOIN**?
 - Un **JOIN** es una operación de combinación entre 1 o varias tablas con el objetivo de devolver los datos pedidos.
 - Los **JOIN** 's crean un conjunto de resultados temporal que contiene los datos combinados.
 - Para unir tablas, debe haber una relación entre ciertas columnas en estas tablas.

Limitación de consulta de tabla única

- Se requieren varios pasos para ver todos los datos deseados:
 - Primero, localice todas las ciudades con el nombre **London** :

```
mysql> SELECT * FROM City WHERE Name = 'London';
```

ID	Name	CountryCode	District	Population
456	London	GBR	England	7285000
1820	London	CAN	Ontario	339917

- En segundo lugar, determinar en qué país se encuentran :

```
mysql> SELECT Code, Name, Continent, Population
```

Code	Name	Continent	Population
CAN	Canada	North America	31147000
GBR	United Kingdom	Europe	59623400

Combinación de dos tablas sencillas

Por ejemplo, antes de unir tablas, cree dos tablas sencillas:

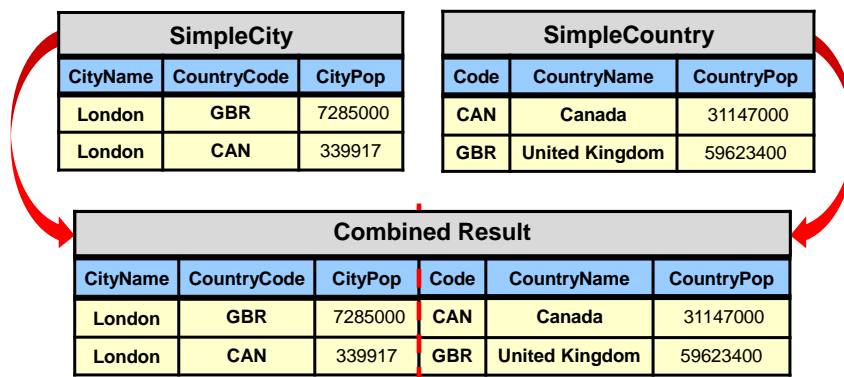
- En primer lugar, una simple tabla `City`:

```
CREATE TABLE SimpleCity AS
SELECT Name AS CityName, CountryCode,
Population AS CityPop
FROM City WHERE Name LIKE 'London';
```

- En segundo lugar, una simple tabla `Country`:

```
CREATE TABLE SimpleCountry AS
SELECT Code, Name AS CountryName, Population
AS CountryPop
FROM Country WHERE Code IN ('CAN', 'GBR');
```

Combinación de dos sencillas tablas: Columnas "pegadas" juntas



- El diseño de la columna del resultado es bueno.
- Las combinaciones de filas no son buenas.
- Otro enfoque es necesario.

Combinación de dos tablas simples: Producto cartesiano

Todas las combinaciones posibles de filas de dos tablas:

SimpleCity			SimpleCountry		
CityName	CountryCode	CityPop	Code	CountryName	CountryPop
London	GBR	7285000	CAN	Canada	31147000
London	CAN	339917	GBR	United Kingdom	59623400

Cartesian Product					
CityName	CountryCode	CityPop	Code	CountryName	CountryPop
London	GBR	7285000	CAN	Canada	31147000
London	GBR	7285000	GBR	United Kingdom	59623400
London	CAN	339917	CAN	Canada	31147000
London	CAN	339917	GBR	United Kingdom	59623400

Combinación de dos sencillas tablas: Dimensiones del producto cartesiano

- **Suma de columnas:** El número total de columnas de las tablas que participan en el producto
- **Suma de filas:** Se calcula multiplicando el número de filas de una tabla por el número de filas de la otra
- Por ejemplo:
 - SimpleCity: 3 columnas, 2 filas
 - SimpleCountry: 3 columnas, 2 filas
 - Producto de SimpleCity y SimpleCountry:
 - $3 + 3 = 6$ columnas
 - $2 * 2 = 4$ filas
 - Los productos cartesianos crecen muy fácilmente.

Filtrado de filas y columnas no deseadas

- Descartar filas no deseadas (filtro):

SimpleCity			SimpleCountry		
CityName	CountryCode	CityPop	Code	CountryName	CountryPop
London	GBR	7285000	CAN	Canada	31147000
London	GBR	7285000	GBR	United Kingdom	59623400
London	CAN	339917	CAN	Canada	31147000
London	CAN	339917	GBR	United Kingdom	59623400

- Descartar las columnas no deseadas (proyección):

SimpleCity			SimpleCountry		
CityName	CountryCode	CityPop	Code	CountryName	CountryPop
London	GBR	7285000	GBR	United Kingdom	59623400
London	CAN	339917	CAN	Canada	31147000

Resultado Final y Recapitulación

- Nombres de la ciudad y nombre del país respectivo:

SimpleCityCountry	
CityName	CountryName
London	United Kingdom
London	Canada

- Recapitulación de la operación de join:
 - Construye el producto cartesiano.
 - Elimine las filas no deseadas.
 - Eliminar las columnas no deseadas.

Calificación de los JOIN' s

- La mayoría de los **JOIN' s** están calificados. (Pertenecen a una serie de agrupaciones según la relación existente)
- La categoría **Qualified Join** es definido dependiendo de la condición asociada a **JOIN**.

Category	Type	Description
Unqualified Join	Cross Join	combina todas las filas de una con todas las filas de la otra tabla (producto Cartesiano)
Qualified Join	Inner Join	Combina todas las filas encontradas en ambas tablas
Qualified Join	Outer Join	Combina las filas encontradas y no encontradas en ambas tablas

Tipos de JOIN: Cross Join

- Un **Cross-Join** también es conocido como un **Producto Cartesiano**.
- Consiste en combinar todas las filas de una tabla con todas las filas de la segunda tabla.
- Unir tablas de esta manera puede producir un número muy grande de filas (filas_1_tabla x Filas_2_tabla => 1000 x 1000 = 1 millón).
- Este tipo de **JOIN** no dispone ninguna condición de combinación entre las tablas. (**Unqualified Join**)
- Es la operación de **JOIN** que mas consume, tanto a nivel de datos como a nivel de **CPU / Memoria**, etc.

Tipos de JOIN: Cross Join

- Two separate tables:

table1

i1	c1
1	a
2	b
3	c

3 rows in set (0.00 sec)

AND

table2

i2	c2
2	c
3	b
4	a

3 rows in set (0.00 sec)

- Tables joined by SELECT:

```
SELECT * FROM table1, table2;
```

i1	c1	i2	c2
1	a	2	c
2	b	2	c
3	c	2	c
1	a	3	b
2	b	3	b
3	c	3	b
1	a	4	a
2	b	4	a
3	c	4	a

9 rows in set (0.00 sec)

- Este JOIN es un producto Cartesiano.

- El resultado incluye todas las combinaciones de tablas entre table1 y table2

JOIN de tablas en SQL

Puede unir tablas utilizando una "coma join" especificando una lista de tablas separadas por comas en la cláusula **FROM** de la sentencia **SELECT**.

Ejemplo:

mysql> SELECT * FROM SimpleCity, SimpleCountry;

CityID	CityName	CountryCode	Code	CountryName	CountryPop
456	London	GBR	CAN	Canada	31147000
1820	London	CAN	CAN	Canada	31147000
456	London	GBR	GBR	United Kingdom	59623400
1820	London	CAN	GBR	United Kingdom	59623400

SimpleCity rows/columns ↑ ↓ SimpleCountry rows/columns

Manteniendo las filas coincidentes

Utilice **SELECT** con la cláusula **WHERE** para retener sólo las filas que satisfacen una condición.

Example:

```
mysql> SELECT * FROM SimpleCity, SimpleCountry
      -> WHERE CountryCode = Code;
```

CityID	CityName	CountryCode	Code	CountryName	CountryPop
1820	London	CAN	CAN	Canada	31147000
456	London	GBR	GBR	United Kingdom	59623400

Condición Join

Una **condición de join** compara las columnas de dos tablas unidas y asegura un resultado que contiene sólo combinaciones de las tablas enumeradas en la cláusula **FROM**.

- No es una cláusula **WHERE** común

Example:

```
mysql> SELECT * FROM SimpleCity, SimpleCountry
      -> WHERE CountryCode = Code
      -> AND CountryPop < 50000000;
```

ID	CityName	CountryCode	Code	CountryName	CountryPop
1820	London	CAN	CAN	Canada	31147000

Diagram annotations:

- From SimpleCity (points to CityID)
- From SimpleCountry (points to CountryName)
- Join condition (points to CountryCode = Code)
- Non-join condition (points to CountryPop < 50000000)

Nombres Ambiguos de Columna

- Una tabla unida puede contener una columna que tiene un nombre idéntico al de una columna de la tabla a la que se une, creando ambigüedad entre tablas.
- Evite la ambigüedad al **cualificar** los nombres de columnas.
 - Utilice los nombres de tabla y columna juntos.
 - Separe los nombres de tablas y columnas con un **punto / punto (.)**.

Ejemplo:

```
SELECT Continent, CountryCode,
       Country.Name, City.Name
FROM City, Country
WHERE CountryCode = Code;
```

Diagram labels for the example:

- Table name (Qualifier) points to `Country` in `Country.Name`.
- Column name points to `Name` in `Country.Name`.
- Qualified column names points to `Country.Name` and `City.Name`.
- punto points to the dot in `Country.Name`.

- Las columnas calificadas pueden aparecer casi en cualquier lugar.
- También puede calificar columnas sin ambigüedades.

Alias de nombre de tabla

- En sentencias de SQL, puede dar a tablas un **alias** como un nombre alternativo para uso local.
- Cuando califica la columna de una tabla de alias, utilice el **alias** como un calificador, no el nombre de la tabla.
- Utilice esta sintaxis para crear un alias de tabla:
 - `<tbl_name> [AS] <alias_name>`

Example:

```
SELECT t1.Name, t2.CountryCode
FROM Country AS t1, City AS t2
WHERE t1.Name = t2.Name;
OR
SELECT t1.Name, t2.CountryCode
FROM Country t1, City t2
WHERE t1.Name = t2.Name;
```

Diagram labels for the example:

- Alias as qualifier points to `t1` in `t1.Name`.
- Optional AS keyword points to `AS` in `Country AS t1`.
- Alias points to `t2` in `City AS t2`.
- Aliased table points to `t2` in `t2.CountryCode`.

Beneficios del uso de alias de tabla

- Puede hacer un alias más corto que el nombre completo de la tabla, que es más conveniente para escribir.
- Las consultas se vuelven más resistentes a los cambios de nombre de tabla.
 - Cambie solamente los nombres de la tabla; Todas las columnas utilizan el alias como calificador.
- Resuelven la ambigüedad de los nombres de las tablas.
 - Necesario cuando una declaración necesita referirse a la misma tabla más de una vez
- Ayudan a aclarar el propósito de la tabla en la declaración.

Errores comunes en los alias

- Si asigna un alias de tabla, debe utilizar el alias en lugar del nombre de tabla al calificar una columna.
- Ejemplo de mal uso de alias:

```
SELECT t1.Name, City.CountryCode
FROM Country AS t1, City AS t2
WHERE t1.Name = t2.Name;
```

- La segunda expresión de la lista `SELECT` hace referencia a la columna `CountryCode` de la tabla `City`.
- A la tabla `City` se le asignó la tabla alias `t2`, cambiando de nombre localmente esa instancia particular de la tabla `City`
- Por lo tanto, se produce un error de esta consulta:

```
ERROR 1054 (42S22):
Unknown column 'City.Name' in 'field list'
```

Sintaxis básica de JOIN

Utilice la palabra clave **JOIN** para combinar tablas y separar la condición de combinación de otros criterios:

```
<table-ref> [ <join-type>] JOIN <table-ref>
ON <join-condition> [WHERE <other-condition>]
```

Ejemplo:

```
mysql> SELECT * FROM SimpleCity JOIN SimpleCountry
-> ON CountryCode = Code ← Join condition
-> WHERE CountryPop < 50000000; ← Non-join condition
```

ID	CityName	CountryCode	Code	CountryName	CountryPop
1820	London	CAN	CAN	Canada	31147000

Sintaxis básica de JOIN

- El **JOIN** crea una nueva tabla "virtual", que es utilizada para mostrar los datos y sólo existe durante la ejecución de la sentencia.

```
SELECT ...
FROM Tabla1, Tabla2
WHERE Condición_de_JOIN => Tabla Virtual
[ GROUP BY ]
[ ORDER BY ]
```

- SINTAXIS SQL 99

```
SELECT ...
FROM Tabla1 INNER JOIN Tabla2 [ USING / ON ] (
Condición_de_JOIN)
WHERE condiciones de SELECT
[ GROUP BY ]
[ ORDER BY ]
```

Reglas para las Combinaciones (JOINS)

- Se pueden obtener la unión de tantas tablas como se quiera
- Puede usarse más de una pareja de columnas para especificar una condición de combinación entre dos tablas (Varias condiciones de JOIN)
- Se pueden seleccionar columnas de todas las tablas implicadas en la combinación
- Si tenemos columnas con idéntico nombre en dos tablas de la combinación se deberán calificar las columnas con el nombre de la tabla a la que pertenecen
- Como regla general, la combinación tendrá como mínimo tantas condiciones en la cláusula **WHERE** como número de tablas de la cláusula **FROM** menos uno

Tipos de JOINS Cualificados

EQUI-JOIN

- Aquella cuyo criterio de combinación se establece a través del operador igual (=).
- Se utiliza para recuperar filas que tengan valores emparejados en cada una de las columnas citadas en el comando **JOIN**

NOT EQUI-JOIN

- Es aquella cuyo criterio de combinación se establece a través del operador no igual (!=).
- Se utiliza para recuperar filas que tengan valores no emparejados en cada una de las columnas citadas en el comando **JOIN**

INNER JOIN = JOIN

- La sintaxis `SQL` ha ido evolucionando y se ha optado por separar las combinaciones (`JOIN's`) y las condiciones aplicadas a `SELECT`.
- La clausula `INNER JOIN` es utilizada para combinar todas las filas coincidentes en ambas tablas

```
SELECT * FROM SimpleCity INNER JOIN
SimpleCountry
ON CountryCode = Code
WHERE CountryPop < 50000000
```

- Para todo `JOIN` debe de tener una condición de unión, que se especifica mediante las clausulas `ON` o `USING`
 - `USING (Columna)`
 - Los nombres de las columnas que intervienen son idénticos en ambas tablas.
 - `ON condición`
 - Nombres de Columnas diferentes en ambas Tablas

INNER JOIN = JOIN

```
mysql> SELECT Country.Name, City.CountryCode
-> FROM Country INNER JOIN City
-> ON Country.Name = City.Name;
```

Name	CountryCode
Djibouti	DJI
Mexico	PHL
Gibraltar	GIB
Armenia	COL
Kuwait	KWT
Macao	MAC
San Marino	SMR
Singapore	SGP

8 rows in set (0.19 sec)

OR

```
SELECT Country.Name,
City.CountryCode
FROM Country INNER JOIN City
USING (Name);
```

`INNER JOIN` es equivalente a `JOIN`

Omitir la condición en JOIN

MySQL le permite omitir la cláusula **ON** desde una **INNER JOIN**.

- Sin una condición de unión, esta declaración simplemente crea un **producto cartesiano**:

```
SELECT * FROM SimpleCity INNER JOIN SimpleCountry;  
... Is equivalent to ...  
SELECT * FROM SimpleCity, SimpleCountry;
```

- La característica de definición de una operación de **inner join** es producir sólo las filas que satisfacen la condición de unión.
 - Esto implica que una condición de unión debe estar presente.

OUTER JOIN

- La cláusula **OUTER JOIN** es utilizada para combinar las filas **coincidentes o NO coincidentes** en ambas tablas
- Este tipo de **JOIN** permite recuperar las filas **HUERFANAS** de las tablas que intervienen en la combinación.

Hay 2 tipos de **OUTER JOIN**

- **LEFT JOIN**
 - Devuelve todas las filas de la tabla de la izquierda, incluso si no hay coincidencias en la tabla de la derecha.
 - Las Filas Huérfanas aparecen y el valor correspondiente a **NULL**
- **RIGHT JOIN**
 - Devuelve todas las filas de la tabla de la derecha, incluso si no hay coincidencias en la tabla de la izquierda.
 - Las **Filas Huérfanas** aparecen y el valor correspondiente a **NULL**

OUTER JOIN

LEFT JOIN:

```
mysql> SELECT column_name(s)
-> FROM left_table
-> LEFT JOIN right_table
-> ON left_table.column_name =
   right_table.column_name;
```

RIGHT JOIN:

```
mysql> SELECT column_name(s)
-> FROM left_table
-> RIGHT JOIN right_table
-> ON left_table.column_name =
   right_table.column_name;
```

```
mysql> SELECT Name, Language
-> FROM Country
-> LEFT JOIN CountryLanguage
-> ON Code = CountryCode;
```

Name	Language
Aruba	Dutch
Antarctica	NULL
French Southern territories	NULL
Antigua and Barbuda	Creole English
Antigua and Barbuda	English
Australia	Arabic

LEFT JOIN

- Con un **LEFT JOIN**, la comparación que se realiza entre las tablas de combinación se basa en la primera (o más a la izquierda).
- Esta tabla se denomina **Tabla de referencia**.
- Si deseamos obtener sólo las filas que no aparecen en la Tabla Derecha utilice la cláusula **WHERE** de la siguiente forma:

```
<left-table> LEFT [OUTER] JOIN <right-table>
ON <join-condition> [WHERE <other-condition>]
```

LEFT JOIN

- Ejemplo

```
mysql> SELECT Name, Language
-> FROM Country
-> LEFT JOIN CountryLanguage
-> ON Code = CountryCode
-> WHERE CountryCode IS NULL;
```

Name	Language
Antarctica	NULL
French Southern territories	NULL
Bouvet Island	NULL
Heard Island and McDonald Islands	NULL
British Indian Ocean Territory	NULL
South Georgia and the South Sandwich Islands	NULL

6 rows in set (0.00 sec)

RIGHT JOIN

- El uso de **RIGHT JOIN**, se utiliza de la misma forma que **LEFT JOIN**.
- Utilizamos la cláusula **WHERE** para las filas no comunes.

```
<left-table> RIGHT [OUTER] JOIN <right-table>
ON <join-condition> [WHERE <other-condition>]
```

El RIGHT OUTER JOIN:

- Devuelve todas las filas que coinciden con la condición del JOIN.
- Conserva filas no coincidentes de **<right-table>**
- Sustituye **NULL** por las columnas **<left-table>** para cada fila no coincidente de **<right-table>**

RIGHT JOIN

- Ejemplo

```
mysql> SELECT Name, Language
-> FROM Country
-> RIGHT JOIN CountryLanguage
-> ON Code = CountryCode
-> WHERE CountryCode IS NULL;
Empty set (0.00 sec)
```

Sintaxis equivalente de LEFT y RIGHT JOIN

- Puede volver a escribir cualquier **LEFT JOIN** como **RIGHT JOIN** y viceversa.
- Ejemplo:

```
SELECT Name, Language
FROM Country LEFT JOIN CountryLanguage
ON Code = CountryCode
WHERE CountryCode IS NULL;

SELECT Name, Language
FROM CountryLanguage RIGHT JOIN Country
ON CountryCode = Code
WHERE CountryCode IS NULL;
```


Elegir entre Inner Joins y Outer Joins

	Pros	Cons
Inner Join	Generalmente es más rápido que los outer joins	Puede perder filas de datos
	Utilizar si no hay valores NULL	-
Outer Join	Utilizar cuando se une a otra tabla mediante outer-joined	Generalmente es más lento que inner join
	Se utiliza para resolver el problema de que no hay filas que cumplan la condición de combinación	Suele implicar valores NULL en expresiones en otras partes de la sentencia
	Se utiliza para agregar filas relacionadas	-

Columnas Nulas en la condición de Join

- Cuando las columnas que aparecen en una condición de combinación permiten valores nulos, debe pensar lo que desea que suceda a las filas que tienen valores nulos.
 - Si desea que el resultado **no contenga estas filas**, puede utilizar **inner join**
 - Si desea lo contrario, **outer join**.

Ejemplo: Capital columna en Country:

Field	Type	Null	Key	Default	Extra
Code	char (3)	NO	PRI		
Name	char (52)	NO			
...					
Capital	int (11)	YES		NULL	
Code2	char (2)	NO			

- Algunas filas de Country tienen un NULL en Capital.
- Estas filas se descartan cuando utiliza una combinación interna.
- Considere el uso de una combinación externa para cualquier consulta que necesite mostrar todos los países con sus mayúsculas.

Uniando a otro Outer Join

- El resultado de un JOIN puede realizar otro JOIN con otra table externa.

```
mysql> SELECT City.Name,  
-> Country.Name AS `Capital of`, Language  
-> FROM City  
-> LEFT JOIN Country          ← Column allows NULL  
-> ON City.ID = Country.Capital  
-> LEFT JOIN CountryLanguage  
-> ON Country.Code =  
CountryLanguage.CountryCode;
```

Cuando no hay filas que cumplan la condición de JOIN

Outer join produce valores `NULL` en caso de que no se encuentre ninguna fila que satisfaga la condición de unión.

- Con una cláusula `WHERE`, puede filtrar las columnas para encontrar estos valores `NULL`.
 - Pruebe una columna que define como `NOT NULL` para distinguir entre un `NULL` "real" y uno generado por la combinación externa.
- Puede usarlo para resolver consultas como:
 - ¿Qué ciudades no son una capital?
 - ¿Qué países no tienen ciudades?
 - ¿Qué ciudades no pertenecen a ningún país?
- Un **inner join** sólo puede recuperar pares de filas coincidentes donde ya existe una coincidencia.

Cuando no hay filas que cumplan la condición: Utilizar LEFT JOIN con IS NULL

Ejemplo:

```
mysql> SELECT Name
-> FROM Country LEFT JOIN CountryLanguage
-> ON Code = CountryCode
-> WHERE Language IS NULL;
```

Name	Language
Antarctica	NULL
French Southern territories	NULL
Bouvet Island	NULL
Heard Island and McDonald Islands	NULL
British Indian Ocean Territory	NULL
South Georgia and the South Sandwich Islands	NULL

6 rows in set (0.00 sec)

Agregación de filas relacionadas

Puede calcular el número de filas **referenciadas** en una tabla relacionada.

Ejemplo:

- ¿Cuántas ciudades hay en la Antártida?

```
mysql> SELECT Country.Code, COUNT(City.ID)
-> FROM Country LEFT JOIN City
-> ON Code = CountryCode
-> WHERE Country.Name = 'Antarctica'
-> GROUP BY Country.Code;
```

Code	COUNT(City.ID)
ATA	0

1 row in set (0.19 sec)

↑ Aggregation

Tipos adicionales de JOIN' s

Además de las uniones internas y externas, hay una serie de otras clasificaciones.

- JOIN con columnas de nombre idéntico:
 - **NATURAL join**
- Join categorías de condición:
 - **Equijoin**
 - **Nonequijoin**
 - **BETWEEN...AND join**
- **autojoin (self-join)**

NATURAL JOIN

- No permite una condición de combinación explícita
- En su lugar, aplica una condición de combinación implícita basada en una comparación de igualdad de todas las columnas idénticamente nombradas presentes en las dos tablas unidas
 - Cuando se utiliza un comodín "todas las columnas" (*) en la lista **SELECT** , las columnas de nombre idéntico se informan sólo una vez.
- Sintaxis:

```
<table-reference>  
NATURAL [<outer-join-type>] JOIN  
<table-reference>
```

NATURAL JOIN: Ejemplo

Ejemplo de **NATURAL JOIN**:

mysql> SELECT **CountryCode**, Language, Name
-> FROM CountryLanguage **NATURAL JOIN** City;

CountryCode	Language	Name
AFG	Balochi	Kabul
AFG	Dari	Kabul
AFG	Pashto	Kabul
AFG	Turkmenian	Kabul
AFG	Uzbek	Kabul
AFG	Balochi	Qandahar
...		

Nombre de columna compartida

Columna combinada para ambas tablas

- La columna `CountryCode` existe en ambas tablas.
 - Se aplica una condición de unión implícita, que requiere igualdad.
- Si utiliza el comodín "todas las columnas" (`*`), se expande para incluir sólo una columna `CountryCode`.

BETWEEN...AND Join

- Una clase especial de **non equi join** es aquella que utiliza el operador **BETWEEN...AND** en su condición **join**.
- Este tipo de **JOIN** se utiliza a menudo para buscar datos que dependen de un rango de valores

Ejemplo:

```
SELECT Employee.ID, Bonus.Amount
FROM Employee INNER JOIN Bonus
ON Employee.Salary
BETWEEN Bonus.LowerSalaryBound
AND Bonus.UpperSalaryBound;
```

Combinar una tabla consigo misma AUTOJOIN

- Un **AUTOJOIN** (también llamado **self-Join**) es una combinación (**JOIN**) entre dos instancias de la misma tabla.
- Ejemplos básicos de **self-Join** son:
 - Una jerarquía de empleados.
 - La tabla de empleados tiene una columna que identifica al empleado y una columna que identifica a su jefe
- Coincidencia de 2 empleados en su fecha de nacimiento

```
SELECT Employee.LastName, Boss.LastName
FROM Employee INNER JOIN Employee AS Boss
ON Employee.BossID = Boss.ID
```

- En la mayoría de los casos, el **self-join** requiere crear un **Alias** en alguna de las 2 instancias de la tabla para evitar ambigüedad.

JOINS con UPDATE y DELETE en multitabla

- En MySQL, los **JOINS** no se limitan a comandos **SELECT**, sino que también pueden ser utilizados con los comandos **UPDATE** y **DELETE**.
- Una instrucción de actualización o borrado junto con un **JOIN** se denomina **ACTUALIZACIÓN Multitabla** o **Borrado Multitabla**
- Estos **JOIN's**, utilizan el resultado de un **JOIN** para hacer una operación específica (actualizar o borrar).
- Las operaciones **Multitabla**, pueden modificar el contenido de varias tablas en una sola sentencia

UPDATE multitable

Sintaxis

```
UPDATE <joined-tables>
SET <column-assignments>
[WHERE <condition>]
```

- La diferencia reside en la inclusión del **JOIN** dentro de la cláusula **UPDATE**.
 - **<joined-tables>**
 - permite indicar cualquier combinación de JOIN
 - **<Asignaciones de Columnas>**
 - Es una lista separada por comas de las asignaciones de columna, tales como:
 - <Columna> = <valor-expresión>
 - **<where>**
 - Es opcional y tiene el mismo significado que en un UPDATE normal

Las cláusulas **ORDER BY** y **LIMIT** NO están permitidos en la **UPDATE** multitable

Multitable UPDATE: Ejemplo

Ejemplo:

- Añadir **20.000** a la población de **Río de Janeiro**, así como a la población total de **Brasil**

```
mysql> UPDATE City
-> INNER JOIN Country
-> ON CountryCode = Code
-> SET City.Population = City.Population + 20000,
-> Country.Population = Country.Population + 20000
-> WHERE Country.Name = 'Brazil'
-> AND City.Name = 'Rio de Janeiro';
Query OK, 2 rows affected (0.09 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```

Column target

Assignment operator

Value expression

DELETE multitable

Sintaxis

```
DELETE <table-list>
FROM <joined-tables>
[WHERE <condition>]
```

```
DELETE
FROM <table-list>
USING <joined-tables>
[WHERE <condition>]
```

- La diferencia reside en la inclusión del **JOIN** dentro de la cláusula **FROM**.
 - **<table-list>**
 - Es una lista separada por comas de nombres de tabla de la que se eliminan las filas.
 - **<joined-tables>**
 - Indica la forma de JOIN entre las diferentes tablas
 - **<where>**
 - Es opcional y tiene el mismo significado que en un UPDATE normal

Todas las tablas incluidas en la **TABLE-LIST** deben de aparecer en **JOINED-TABLE** sino → ERROR

Multitable DELETE

Ejemplo:

- Quitar el país con el código NLD y todas sus ciudades e idiomas::

```
DELETE Country, City, CountryLanguage
FROM Country
LEFT JOIN City
ON Code = City.CountryCode
LEFT JOIN CountryLanguage
ON Code = CountryLanguage.CountryCode
WHERE Code = 'NLD';
```


UPDATE y DELETE en multitabla

VENTAJAS

- Agrupar múltiples **UPDATE** o **DELETE** en una sola ejecución Multi-Tabla para reducir las operaciones de ida y vuelta al servidor.
- Reescribir las operaciones con bajo rendimiento en **UPDATE** o **DELETE** con multi-Tabla en **Subconsultas**.

VENTAJAS

- **UPDATE** o **DELETE** multi-tabla no están garantizados que se ejecute de forma atómica.
- No pueden utilizarse las órdenes **ORDER BY** ni **LIMIT**.
- La eliminación en **CASCADE** y reglas de actualización con las **F.K.** en InnoDB pueden no funcionar correctamente.
- La sintaxis no es estándar, y causa un problema con la portabilidad.

Subconsultas

Objetivos

Después de completar esta lección, usted debería ser capaz de:

- Escribir consultas que contienen consultas anidadas
- Posición de una subconsulta correctamente para recuperar los datos necesarios
- Identificar el tipo apropiado de subconsulta para usar
- Utilice la sintaxis SQL correcta para crear subconsultas
- Describir y utilizar cuantificadores para comparaciones de subconsultas

Sub-Consultas

- Hasta ahora hemos utilizado la cláusula **WHERE** para definir condiciones emparejando un valor de columna con una variable/columnas.
- Pero ¿Qué hacer cuando se desconoce el valor con el que queremos comparar?

SOLUCIONES

- Adivinar el valor a comparar y posteriormente utilizarlo.
 - Efectuar una consulta (**SELECT**) y anotar el resultado y posteriormente usarlo para la consulta inicial.
- Utilizar Sub-consultas.
 - Ejecutar una sentencias **SELECT** con Sub-consultas, de forma única.

Sub-Consultas

- Estas **SUBCONSULTAS** se definen en lugar del valor a "adivinar" dentro de la cláusula **WHERE**.
- La **SUBCONSULTA** también es llamada **Consulta Anidada**
- Es introducir una Consulta (**Select**) dentro de la Consulta (**Select**).

```
SELECT Language      ①
FROM CountryLanguage
WHERE CountryCode = ( ②
    SELECT Code       ③
    FROM Country
        WHERE Name = 'Finland'
    ) ;               ④
```

Sub-Consultas

Características

- La **SUBSELECT** deberá ir encerrada entre paréntesis para que se realice antes esta **SELECT**.
- La **SUBSELECT** nos puede devolver 1 solo valor o Varios valores.
- Es una alternativa a **JOIN**.
- Solo son permitidas en algunas partes de las consultas **SELECT** (**Where**, etc)
- Una vez ejecutada la **SUBSELECT**, los datos son eliminados.

Sub-Consultas

```
mysql> SELECT Language
-> FROM CountryLanguage
-> WHERE CountryCode = (SELECT Code
-> FROM Country
-> WHERE Name='Finland');
```

← Main/outer query

Subquery/
Inner query

Language
Estonian
Finnish
Russian
Saame
Swedish

Sub-Consultas

- Las **SUBCONSULTAS** son útiles en los casos en que los **JOIN's** no funcionan o no son óptimos.
- También son utilizadas para clarificar las sentencias complicadas.

¿Dónde podemos utilizar **SUBSELECT** ?

- Clausula **WHERE**
 - Vistas anteriormente
- Clausula **FROM**
 - Permite poner una Sub-consulta siempre que tenga definido un ALIAS de Tabla al final

```
mysql> SELECT * from (SELECT Code, name
-> FROM Country
-> WHERE Name like 'az') as T1;
```

Code	name
ABW	Aruba
AFG	Afghanistan
AGO	Angola
ATA	Anguilla
ARG	Argentina

```
SELECT ... FROM ( Sub-Select ) AS Alias ;
```

Tipos de Subconsultas

Dependiendo del **RESULTADO** de la Subconsulta

- **VALOR SIMPLE**
 - La Sub-Consulta Devuelve 0 o 1 valor de 1 o mas columnas
- **VALOR MULTIPLE**
 - La Sub-Consulta Devuelve mas de 1 valor de 1 o mas columnas

Dependiendo de relación con la **SELECT** Inicial

- **No-Correlacionada**
 - La Sub-Consulta no hace referencia a la Consulta Select inicial
 - La Sub-Consulta **No depende** de consulta Select inicial
 - La Sub-Consulta puede ejecutarse de forma aislada
- **Correlacionada**
 - La Sub-Consulta **hace referencias** a columnas en la Consulta Select inicial
 - La Sub-Consulta **Si depende** de consulta Select inicial
 - La Sub-Consulta No se puede estar solo

Tipos de Subconsultas

VALORES UNICOS

- Si una **Subselect** devuelve un solo valor, se usarán exclusivamente los operadores de comparación.

= , <> , < , <= , > , >=

```
SELECT columnas FROM tabla
WHERE columna > ( Subselect..... );
```

La consulta siguiente recupera cada país y su población como un porcentaje de la población mundial entera:

```
SELECT Country.Name,
       100 * Country.Population /
       (SELECT SUM(Population) FROM Country)
       AS pct_of_world_pop
FROM Country;
```

Subconsultas de fila

- Las **subconsultas de fila** devuelven una sola fila que contiene al menos dos columnas.

```
SELECT ( Columna1, columna2, .. ) =  
( Subselect..... ) as Alias
```

- Si la expresión `SELECT` entre paréntesis recupera una sola fila, la subconsulta de fila se evalúa en esa fila.
 - Para los operadores de igualdad (`=`, `<=>`), **todas las comparaciones** pairwise necesitan evaluar a true para que la expresión sea verdadera
 - Dos filas son iguales sólo si todos los valores de columna correspondientes son iguales.
 - Para los operadores de desigualdad (`!=`, `Y <>`), **sólo una comparación** pairwise necesita ser verdadera para que la expresión sea verdadera

`<=>` Operador de comparación de NULL
Si una de las 2 condiciones es NULL → 0

Subconsultas de fila

En el siguiente ejemplo, la subconsulta es igual al literal del constructor de filas (`'London '`, `'GBR '`).

```
mysql> SELECT ('London', 'GBR') =  
-> (SELECT Name, CountryCode FROM City  
-> WHERE ID = 456) AS IsLondon;  
  
+-----+  
| IsLondon |  
+-----+  
| 1        |  
+-----+
```

```
mysql> SELECT (SELECT ID, Name, CountryCode FROM City  
-> WHERE ID = 456)  
-> = (SELECT ID, Name, CountryCode FROM City  
-> WHERE CountryCode = 'GBR'  
-> AND Name = 'London') AS IsEqual;  
  
+-----+  
| IsEqual |  
+-----+  
| 1        |  
+-----+
```

Las subconsultas de dos filas
pueden compararse entre sí:

Subconsultas de fila con conjuntos vacíos

El siguiente ejemplo ilustra que una **subconsulta de fila** siempre se evalúa en una fila, incluso si la expresión de consulta entre paréntesis devuelve un conjunto vacío:

```
mysql> SELECT (null, null) <=>
        -> (SELECT GNP, Capital FROM Country LIMIT 0);
+-----+
| (null, null) <=>
  (SELECT GNP, Capital FROM Country LIMIT 0) |
+-----+
|                                           1 |
+-----+
1 row in set (0.00 sec)
```

Subconsultas de tabla

- Las **subconsultas de tabla** actúan como tablas de sólo lectura.
- Las subconsultas de tabla devuelven un conjunto de resultados que contiene cero o más filas con una o más columnas.
- Pueden aparecer en dos contextos diferentes:
 - En la cláusula **FROM** de una consulta adjunta
 - Como operadores lógicos **IN** y **EXISTS**, o como operador comparación regular (**=**, **!=**, **<>**, **<**, **>**, **<=**, **>=**) cuantificado con **ALL**, **ANY** y **SOME**

Subconsultas en la cláusula **FROM**

- El conjunto de resultados de una subconsulta en la cláusula **FROM** se trata de la misma manera que los resultados obtenidos de tablas base o vistas a las que se hace referencia en la cláusula **FROM**.
- El siguiente ejemplo ilustra el uso de una subconsulta en la cláusula **FROM**:

```
SELECT *  
FROM (SELECT Code, Name  
FROM Country  
WHERE IndepYear IS NOT NULL) AS IC;
```

Cálculo de agregados

- Las subconsultas de la cláusula **FROM** son especialmente útiles para calcular agregados de agregados.
- La siguiente consulta encuentra el promedio de las sumas de la población de cada continente:
 - MySQL evalúa primero la subconsulta de la tabla, calculando el total de todas las poblaciones de cada continente.
 - La función **AVG** en la consulta externa agrega las filas resultantes (una para cada continente).

```
SELECT AVG(cont_sum)  
FROM (  
    SELECT Continent, SUM(Population) AS cont_sum  
    FROM Country  
    GROUP BY Continent  
) AS t;
```


Tipos de Subconsultas. Valores Múltiples

VALORES MÚLTIPLES

- Si una **Subselect** puede devolver mas de un valor, usaremos los operadores de comparación unidos a las siguientes cláusulas:

[oper] **ANY**

[oper] **ALL**

IN, EXISTS

- Al igual que los operadores de comparación regular, los operadores de subconsulta de tabla devuelven un resultado booleano

Tipos de Subconsultas. Valores Múltiples

[oper] **ANY**

- Compara la columna con los valores devueltos por la **subselect** y si alguno de ellos cumple la condición del operador, el resultado es **TRUE**.
- Si la **subselect** es 0, el resultado es **FALSE**

```
SELECT 'Finland' = ANY (  
    SELECT Name  
    FROM Country  
);
```

Tipos de Sub-consultas. Valores Múltiples

[oper] **ALL**

- Compara la columna con los valores devueltos por la **subselect** y si la columna cumple la condición del operador con todos los resultados, el resultado es **TRUE**.
- Si la subselect es 0, el resultado es **FALSE**

```
Mysql> SELECT name FROM City
-> WHERE Population >
-> ALL ( SELECT Population FROM City
-> WHERE CountryCode = 'CHN');
```

Alternativas a ANY y ALL

ANY and **ALL** pueden ser confusas para leer.
La siguiente lista muestra alternativas más legibles para un número de expresiones de operador cuantificadas

Expresión cuantificada del operador	Alternativa
scalar > ANY(SELECT column...)	scalar > (SELECT MIN(column) ...)
scalar < ANY(SELECT column...)	scalar < (SELECT MAX(column) ...)
scalar > ALL(SELECT column...)	scalar > (SELECT MAX(column) ...)
scalar < ALL(SELECT column...)	scalar < (SELECT MIN(column) ...)
scalar = ANY(SELECT column...)	scalar IN (SELECT column...)
scalar <> ALL(SELECT column...)	scalar NOT IN (SELECT column...)

Tipos de Sub-consultas. Valores Múltiples

IN (= ANY)

- Es equivalente a **=ANY**.
- Compara la columna con los valores devueltos por la **subselect** y si la columna es igual a alguno de los resultados devueltos por la **subselect**, el resultado es **TRUE**.

```
Mysql> SELECT name, Population FROM City  
-> WHERE CountryCode  
-> IN ( SELECT Code FROM Country  
-> WHERE Continent = 'Europe')  
-> ORDER BY Name;
```

Operación IN por parejas (Pairwise)

El siguiente ejemplo ilustra la **comparación de columnas por parejas**:

```
SELECT *  
FROM City  
WHERE (CountryCode, Name) IN (  
    SELECT Code, Name  
    FROM Country  
    WHERE Continent = 'Asia'  
);
```

Usando **NOT IN**

- También podemos negar el operador **IN** anteponiendo la palabra **NOT**.
- **NOT IN** se evalúa como **TRUE** si el conjunto de resultados no contiene una entrada que sea igual al operando izquierdo.
- Si el conjunto de resultados contiene una entrada igual al operando izquierdo, **NOT IN** se evalúa como **FALSE**.

```
Mysql> SELECT name, Population FROM City
-> WHERE CountryCode
-> NOT IN ( SELECT Code FROM Country
-> WHERE Continent = 'Europe')
-> ORDER BY Name;
```

Operador **EXISTS**

El operador **EXISTS** acepta un único argumento a la derecha, que debe ser una subconsulta de tabla..

- Si el conjunto de resultados de subconsulta contiene al menos una fila, la expresión **EXISTS** se evalúa como **true**.
- Devuelve **false** en todos los demás casos.

Por ejemplo, la siguiente consulta recupera todas las ciudades que son el capital de algún país:

```
SELECT *
FROM City
WHERE EXISTS (
  SELECT NULL
  FROM Country
  WHERE Capital = ID);
```

Operador **NOT EXISTS**

La negación de **EXISTS** se obtiene colocando la palabra clave **NOT** antes de la palabra clave **EXISTS**.

- La construcción **NOT EXISTS** devuelve **TRUE** sólo si su argumento de subconsulta de tabla está vacío y **FALSE** en caso contrario.
- La siguiente consulta ilustra cómo utilizar **NOT EXISTS** para encontrar todos los países donde no se habla inglés:

```
SELECT *  
FROM Country  
WHERE NOT EXISTS (  
    SELECT NULL  
    FROM CountryLanguage  
    WHERE CountryCode = Code  
    AND Language = 'English');
```

Subconsultas correlacionadas

Una **subconsulta correlacionada** contiene una o más expresiones que se derivan de la consulta externa.

Example:

```
SELECT *  
FROM Country  
WHERE NOT EXISTS (  
    SELECT NULL  
    FROM City  
    WHERE CountryCode = Code  
);
```

Subconsultas correlacionadas. Alcance

Las referencias de columna en subconsultas correlacionadas se resuelven utilizando el **ámbito de aplicación** más cercano posible:

```
SELECT *
FROM City
WHERE CountryCode IN (
    SELECT Code
    FROM Country
    WHERE Name = 'Belgium'
);
```

```
SELECT *
FROM City
WHERE CountryCode IN (
    SELECT Code
    FROM Country
    WHERE
        Country.Name='Belgium'
);
```

- En el ejemplo, la subconsulta requiere que la columna **Name**.
- Tanto la tabla **City** en la consulta externa como la Tabla **Country** en la subconsulta contienen columnas **Name**.
- Sin embargo, esta ambigüedad no plantea un problema porque las referencias de columna se resuelven utilizando el ámbito más cercano posible.

Modificación de datos con subconsultas

Utilice subconsultas al modificar datos:

```
DELETE FROM City
WHERE CountryCode IN (
    SELECT Code FROM Country
    WHERE LifeExpectancy < 70.0
);
```

```
UPDATE Country
SET Population = (
    SELECT SUM(Population) FROM City
    WHERE CountryCode = Code
);
```

Vistas

Introducción a las vistas

- Una **vista** es un objeto de base de datos definido como una consulta que es ejecutada en tiempo real.
- La consulta que crea la vista es una sentencia **SELECT** que puede acceder a una o más tablas o vistas.
- Las vistas también se llaman **tablas virtuales** porque no tienen una estructura definida pero si tienen datos.
- Para poder crear vistas, necesitamos algunos privilegios como:
 - **CREATE VIEW, ALTER VIEW, DROP VIEW**
 - **Select** sobre las tablas subyacente y sobre las columnas
 - En algunos casos, cuando unavista es actualizable deberá tener privilegios de **UPDATE, DELETE o INSERT** para modificar una tabla base subyacente.

Introducción a las vistas

- Una vez creada una vista, su definición queda "**congelada**", es decir, los cambios posteriores, producidos en las tablas subyacentes no afectan a la definición de la vista.
 - Por ejemplo, si una vista se define como **SELECT *** sobre una tabla, las nuevas columnas agregadas a la tabla, aparecerán como parte de la vista.
- Las vistas se basan en los privilegios y seguridad del usuario y los de las tablas subyacentes.
- Dependiendo de la creación de la Vista, ésta puede ser:
 - No actualizable
 - Actualizable
 - Una vista actualizable se puede utilizar comandos como **UPDATE**, **DELETE** o **INSERT** para modificar una tabla base subyacente.

Razones para utilizar vistas

Las vistas proporcionan varias ventajas sobre la selección directa sobre las tablas subyacentes:

- Selección un conjunto restringido de filas mediante cláusula **WHERE**, o de columnas en una tabla.
- Selección de un conjunto de columnas extraídas de diferentes tablas.
- Ejecución de un cálculo y mostrar su resultado.
- Selección de un contenido de la tabla de forma diferente para diferentes usuarios, de modo que cada usuario sólo ve los datos relativos a las actividades de ese usuario.
- La vista preserva la apariencia de la estructura de tabla original, para minimizar la perturbación a otras aplicaciones.

Crear una vista

- Para definir una vista, utilice la sentencia **CREATE / REPLACE VIEW**, que tiene la siguiente sintaxis:

```
CREATE [OR REPLACE]
[ALGORITHM = <algorithm_type>]
VIEW <view_name> [(<column_list>)]
AS <select_expression>
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- Solo las clausulas **CREATE VIEW <nombre_vista> AS <sentencia_select>** son obligatorias, el resto son opcionales.

```
mysql> CREATE VIEW CityView AS
-> SELECT ID, Name FROM City;
Query OK, 0 rows affected (0.001 sec)
mysql> SELECT * FROM CityView;
```

Crear una vista: Ejemplo

La siguiente sentencia **CREATE VIEW** define una vista simple en la base de datos `world_innodb`, denominada `CityView`.

```
mysql> CREATE VIEW CityView AS
-> SELECT ID, Name FROM City;
Query OK, 0 rows affected (0.001 sec)

mysql> SELECT * FROM CityView;
+----+-----+
| ID | Name   |
+----+-----+
| 1  | Kabul  |
| 2  | Qandahar |
| 3  | Herat  |
...
```

Crear una vista

```
CREATE [OR REPLACE]
[ALGORITHM = <algorithm_type>]
VIEW <view_name> [(<column_list>)]
AS <select_expression>
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

[OR REPLACE]

- Si la Vista ya existe, realiza un **DROP** de la misma y la vuelve a recrear

[ALGORITHM=<tipo>]

- Especifica el Algoritmo a utilizar cuando invocamos la Vista
 - MERGE** El texto de la consulta es MEZCLADO con el texto de la vista → Reescritura VISTA
 - TEMPTABLE** Los resultados de la Vista es llevado a una tabla temporal la cual es usada para la Consulta
 - UNDEFINED** MySQL elige el Algoritmo a utilizar

<select_expression>

- Es una **SELECT** que indica los datos/columnas a recuperar para la vista.
- La declaración puede seleccionar una o mas Tablas o Vistas.

[WITH CHECK OPTION]

- Utilizadas en Vistas actualizable.
- Se comprueba que en operaciones **DML** en la Vista, éstas operaciones satisfacen el **WHERE** dentro de <select_expresion> [CASCADED | LOCAL] → Alcance de la verificación

Crear una vista: Renombrar Columnas

- Cuando creamos una vista, las columnas que componen la vista son los mismos nombre que las columnas de las tablas subyacente.
 - ¿Qué ocurre cuando necesitamos columnas de diferentes tablas que se llamen igual?
 - ¿Cómo se trata cuando una columna de la Vista es una función de agregación? ¿Qué nombre se le pone?

```
mysql> CREATE VIEW v AS
-> SELECT Country.Name, City.Name
-> FROM Country, City WHERE Code = CountryCode;
ERROR 1060 (42S21): Duplicate column name 'Name'
```

- Algo habitual en la creación de una vista, es renombrar las columnas resultantes.

Crear una vista: Renombrar Columnas

Disponemos de 2 formas de realizar esta operación:

1. Crear la Vista e indicar en la consulta **SELECT** las columnas y sus **ALIAS** que se convertirán en **Nombres_de_Columnas**.

```
mysql> CREATE VIEW v AS
-> SELECT Country.Name AS CountryName, City.Name AS CityName
-> FROM Country, City WHERE Code = CountryCode;
Query OK, 0 rows affected (0.001 sec)
```

2. Definir la Vista indicando los nombres de las columnas dentro de la **Vista**

```
mysql> CREATE VIEW v (CountryName, CityName) AS
-> SELECT Country.Name, City.Name
-> FROM Country, City WHERE Code = CountryCode;
Query OK, 0 rows affected (0.001 sec)
```

Vistas Actualizables: MonoTabla

OPERACIONES			RESTRICCIONES
CONSULTAS			Sin restricciones
B O R R A D O	A C T U A L I Z A C I Ó N	I N S E R C I Ó N	<ul style="list-style-type: none">• Vista Monotabla• Vista creada sin cláusulas: GROUP BY DISTINCT HAVING
			<ul style="list-style-type: none">• No existen columnas en la vista obtenidas como expresión• Algorithm=TEMPTABLE
			<ul style="list-style-type: none">• Todas las columnas obligatorias están en la vista

Vistas Actualizables: MonoTabla

- Los datos se pueden modificar a través de la misma vista o actualizando la tabla a la que pertenece la vista

RESTRICCIONES DE MANIPULACIÓN DE DATOS CON VISTAS

- **Borrado de Filas**
 - La vista solo puede tener filas de una tabla.
 - No se pueden utilizar en la sentencia **SELECT** de creación de la vista las cláusulas **GROUP BY, DISTINCT, HAVING**
 - No se pueden utilizar en la sentencia **SELECT** de creación las funciones de grupo o referencia a **UNION**
- **Actualización de Filas**
 - Todas las restricciones de borrado
 - Ninguna de las columnas a actualizar debe haber sido definida como una expresión
 - **ALGORITHM = TEMPTABLE**
- **Inserción de Filas**
 - Todas las restricciones anteriores
 - Deben estar presentes en la vista todas las columnas obligatorias de la tabla asociada

Vistas Actualizables

Una vista es **updatable** si puede utilizarla con sentencias como **UPDATE** o **DELETE** para modificar la tabla base subyacente.

- Las principales condiciones para la actualización son
 - Debe haber una relación uno a uno entre las filas de la vista y las filas de la tabla base.
 - Las columnas de vista que se deben actualizar deben definirse como simples referencias de columna, no como expresiones.
- No puede actualizar una vista en la tabla si la vista se define con columnas de **valores agregados calculados** a partir de la tabla.

Vistas Actualizables: Ejemplos

Ejemplo:

- Creamos **Vista Actualizable**

```
mysql> CREATE VIEW EuropePop AS
-> SELECT Name, Population FROM Country
-> WHERE Continent = 'Europe';
Query OK, 0 rows affected (0.001 sec)
```

- La vista **EuropePop** satisface los requisitos de actualizable:
 - Sus columnas son referencias de columnas simples
 - No expresiones de agregación ni cálculo.
- Realizamos una actualización

```
mysql> UPDATE EuropePop SET Population = Population + 1
-> WHERE Name = 'San Marino';
Query OK, 1 row affected (0.001 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Vistas Actualizables: Ejemplos

Ejemplo: (continua)

- Podemos Borrar en la Vista

```
mysql> DELETE FROM EuropePop WHERE Name = 'San Marino';
Query OK, 1 row affected (0.001 sec)
mysql> SELECT * FROM EuropePop WHERE Name = 'San Marino';
Empty set (0.001 sec)
```

- La vista **EuropePop** satisface los requisitos de actualizable:
 - Sus columnas son referencias de columnas simples
 - No expresiones de agregación ni cálculo.

```
mysql> UPDATE EuropePop SET Population = Population + 1
-> WHERE Name = 'San Marino';
Query OK, 1 row affected (0.001 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

Vistas insertables

Una vista actualizable es **insertable** si también satisface estos requisitos adicionales a los anteriores para las columnas de vista:

- No hay nombres de columna de vista duplicados.
- La vista contiene todas las columnas de la tabla base que no tienen un valor predeterminado.
- Las columnas de vista son simples referencias de columna y no columnas derivadas.
- Estos son ejemplos de columnas derivadas:
 - 3.14159
 - col1 + 3
 - UPPER(col2)
 - col3 / col4
 - (<subquery>)

Vistas actualizables Multi-Tabla

Puede crear una **vista actualizable Multi-tabla** con las siguientes restricciones:

- La vista se puede procesar con el algoritmo **MERGE**.
- La vista debe utilizar un **inner join** (no se permite **outer join** o **UNION**).
- Sólo se puede **actualizar una única tabla** en la definición de vista.
- No puede **insertar** en una vista que utiliza **UNION ALL** incluso si la vista observa las reglas anteriores.
- Puede **insertar** en la vista sólo si la sentencia **agrega filas a una sola tabla**.
- No puede utilizar la instrucción **DELETE**.

Usando WITH CHECK OPTION

- Si una vista es **actualizable**, podemos utilizar la cláusula **WITH CHECK OPTION** para definir restricciones sobre las modificaciones.
- Esta cláusula comprueba la cláusula **WHERE** definida en la vista cuando se intentan realizar actualizaciones.
 - Se puede realizar una actualización de una fila existente sólo si la cláusula WHERE es cierto para la fila resultante.
 - Se puede realizar un INSERT sólo si la cláusula WHERE es cierta para la nueva fila.

```
mysql> CREATE VIEW LargePop AS
-> SELECT Name, Population FROM Country
-> WHERE Population >= 100000000
-> WITH CHECK OPTION;
Query OK, 0 rows affected (0.000 sec)
```

```
mysql> UPDATE LargePop SET Population = Population + 1
-> WHERE Name = 'Nigeria';
Query OK, 1 row affected (0.000 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
mysql> UPDATE LargePop SET Population = 99999999
-> WHERE Name = 'Nigeria';
ERROR 1369 (HY000): CHECK OPTION failed 'world_innodb.LargePop'
```

Estado de una Vista

- Cuando trabajamos con vistas, los objetos en los que se basa la vista deben de existir y estar correctamente definidos.
 - Tablas subyacentes
 - Vistas utilizadas en la vista
 - Columnas de Tablas
- Si modificamos/borrado alguno de ellos, la vista puede no estar disponible.
- La vista se vuelve **INVALIDA** cuando alguno de sus objetos son **BORRADOS**.
- Para chequear las vistas y solucionar posibles problemas con ellas disponemos del comando:
CHECK TABLE <nombre_vista>

Estado de una Vista

Imaginemos el siguiente ejemplo:

```
mysql> CREATE TABLE t1(i int)
mysql> CREATE VIEW v AS SELECT i FROM t1;
mysql> RENAME TABLE t1 TO t2;
mysql> CHECK TABLE t1\G
```

```
***** 1. row *****
Table: world_innodb.v
Op: check
Msg_type: Error
Msg_text: Table 'world_innodb.t1' doesn't exist
***** 2. row *****
Table: world_innodb.v
Op: check
Msg_type: Error
Msg_text: View 'world_innodb.v' references invalid table(s) or
column(s) or function(s) or definer/invoke of view lack rights to
use them
***** 3. row *****
Table: world_innodb.v
Op: check
Msg_type: error
Msg_text: Corrupt
3 rows in set (0.15 sec)
```

Modificar una vista

- Para cambiar la definición de una vista existente, utilice la sentencia **ALTER VIEW**.
- **ALTER VIEW** modifica la definición actual de la vista y la sustituye por la nueva definición.
- Si la vista indicada no existe, MySQL genera un error.

Sintaxis:

- La sintaxis de **ALTER VIEW** es similar a la de **CREATE VIEW** exceptuando que no se puede utilizar la palabra **REPLACE**

```
CREATE [ALGORITHM = <algorithm_type>]
VIEW <view_name> [(<column_list>)]
AS <select_expression>
[WITH [CASCADED | LOCAL] CHECK OPTION]
```


Borrar una vista

- Para eliminar una o mas vistas utilizaremos la sentencia **DROP VIEW**.

```
DROP VIEW [IF EXISTS] <view_name>
[, <view_name> ... ]
```

- Si la vista indicada no existe, MySQL genera un error.
 - Si incluimos la cláusula **IF EXISTS** MySQL genera un **warning** en vez de un error si la vista no existe.
 - Este **warning** puede ser mostrado mediante **SHOW WARNINGS**.
 - **IF EXIST** es una extensión MySQL de SQL, no es estándar.

Borrar una vista

- Ejemplo

```
mysql> DROP VIEW IF EXISTS v1, v2;
Query OK, 0 rows affected, 1 warning (0.001 sec)

mysql> SHOW WARNINGS;
+-----+-----+-----+
| Level | Code | Message                                     |
+-----+-----+-----+
| Note  | 1051 | Unknown table 'world_innodb.v2'          |
+-----+-----+-----+
1 row in set (0.001 sec)
```

Información de las Vistas

La base de datos `INFORMATION_SCHEMA` contiene dos tablas que son útiles en la búsqueda de información sobre las vistas:

- **TABLES**

- Contiene una fila para cada Tabla y Vistas accesible para el usuario, que ejecuta la consulta.
- La columna `TABLE_TYPE` Indica su tipo. Contiene el valor "VIEW" para todas las vistas.
- El resto de las columnas, contienen, principalmente, datos de las tablas, por lo que contendrán `NULL` para las vistas.

- **VIEWS**

- Contiene metadatos específicos para las vistas.
- La columna `VIEW_DEFINITION` Contiene la sentencia SELECT asociada a la Vista.
- La columna `IS_UPDATABLE` Indica si la vista es actualizable.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.VIEWS  
-> WHERE TABLE_NAME='CityView'\G
```

Obtener Ver metadatos con SHOW CREATE VIEW

También podemos obtener información de una vista concreta mediante el comando `SHOW CREATE VIEW <nombre_vista>`

```
mysql> SHOW CREATE VIEW CityView\G  
***** 1. row *****  
View: CityView  
Create View: CREATE ALGORITHM=UNDEFINED  
DEFINER='root'@'localhost' SQL SECURITY DEFINER VIEW `CityView`  
AS SELECT `City`.`ID` AS `ID`,`City`.`Name` AS `Name` FROM  
`City`  
character_set_client: utf8  
collation_connection: utf8_general_ci  
1 row in set (0.02 sec)
```

Información de las Vistas

- Algunos comandos habituales de MySQL permiten opciones para obtener metadatos e información adicional.

```
mysql> SHOW FULL TABLES
```

```
mysql> SHOW TABLE STATUS
```

Añade una columna adicional con el Tipo
acerca de cada tabla / VISTA

```
mysql> SHOW FULL TABLES FROM world_innodb;
+-----+-----+
| Tables_in_world | Table_type |
+-----+-----+
| City             | BASE TABLE |
| CityView         | VIEW        |
| Country          | BASE TABLE |
| CountryLangCount | VIEW        |
| CountryLanguage  | BASE TABLE |
| EuropePop        | VIEW        |
| LargePop         | VIEW        |
+-----+-----+
```

Ofrece una gran cantidad de información

```
mysql> SHOW TABLE STATUS LIKE 'wp_postmeta'\G
***** 1. row *****
      Name: wp_postmeta
      Engine: MyISAM
      Version: 10
      Row_format: Dynamic
      Rows: 1351
      Avg_row_length: 46
      Data_length: 62960
      Max_data_length: 281474976710655
      Index_length: 58368
      Data_free: 76
      Auto_increment: 1745
      Create_time: 2009-09-08 09:57:51
      Update_time: 2010-06-04 15:06:47
      Check_time: 2010-06-01 11:12:54
      Collation: utf8_general_ci
      Checksum: NULL
      Create_options:
      Comment:
1 row in set (0.00 sec)
```

Obtención de Metadatos

METADATOS

- Una de las partes importantes de cualquier RDBMS son los metadatos.
- METADATOS son "datos sobre datos"
 - Información de administración sobre el servidor MySQL y todas sus características.
 - Mediante ellos, se obtiene una gestión eficaz y rápida de elementos como:
 - Nombre de una base de datos o tabla
 - Tipo de datos de una columna
 - Privilegios de acceso, etc

Hay varias maneras de obtener metadatos:

- Mediante el comando `SHOW`
 - Proporcionar información sobre las bases de datos, tablas y columnas, así como el estado del servidor.
- Mediante consulta de la Base de Datos `INFORMATION_SCHEMA`
 - Esta BBDD almacena los metadatos de todo el Gestor MySQL
 - Es una BBDD idéntica a las anteriores y su consulta se realiza con la cláusula `SELECT`
- Programa `mysqlshow`

INFORMATION_SCHEMA

- La base de datos `INFORMATION_SCHEMA` contiene tablas que contienen información sobre:
 - Las **tablas** de Bases de Datos
 - sus nombres
 - Tamaños
 - Número de filas de cada tabla
 - Tablas usadas
 - Las **columnas** de tablas en Bases de Datos
 - Tipo de datos de cada columna.
 - Restricciones de Columnas
- En terminología de Bases de Datos es denominada `CATALOGO` de la Base de Datos
 - `CATALOGO` = Conjunto de metadatos donde es posible extraer información de gestión mantenimiento de las Bases de Datos.

INFORMATION_SCHEMA

- Dentro de `INFORMATION_SCHEMA` hay varias tablas que son sólo de lectura (**vistas**), por lo que no hay ficheros asociadas a ellas.
- Al ser `INFORMATION_SCHEMA` una BBDD especial, **el Servidor NO crea un directorio para ella**.
- Podemos utilizar esta Base de Datos para obtener información de sus tablas, pero no podemos hacer ninguna otra operación:
 - No se permite operaciones DML
(insertar, actualizar o eliminar registros).
- Accediendo a estas tablas podemos obtener información y guardarla como si fuese un acceso normal a tablas MySQL.

INFORMATION_SCHEMA

```
mysql> SELECT  TABLE_NAME
-> FROM      INFORMATION_SCHEMA.TABLES
-> WHERE     TABLE_SCHEMA = 'information_schema'
-> ORDER BY  TABLE_NAME;

+-----+
| TABLE_NAME |
+-----+
| CHARACTER_SETS |
| COLLATIONS |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS |
| COLUMN_PRIVILEGES |
| ... |
| USER_PRIVILEGES |
| VIEWS |
+-----+
```

Muestra todas las tablas que componen la Base de Datos INFORMATION_SCHEMA

INFORMATION_SCHEMA

```
mysql> SELECT table_name, engine
      -> FROM INFORMATION_SCHEMA.TABLES
      -> WHERE table_schema = 'world_innodb';
```

table_name	engine
city	InnoDB
country	InnoDB
country_view	NULL
countrylanguage	InnoDB
countrylanguage2	InnoDB
per_capita_v	NULL

Muestra todas las tablas que componen la Base de Datos WORLD_INNODB y sus MOTORES de almacenamiento utilizado para su creación

INFORMATION_SCHEMA

- **SCHEMATA** es una Tabla que contiene METADATOS de las BBDD actuales
- Podemos obtener información global de Configuración de la BBDD indicada

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA
      -> WHERE SCHEMA_NAME = 'world_innodb'\G
***** 1. row *****
      CATALOG_NAME: def
      SCHEMA_NAME: world_innodb
  DEFAULT_CHARACTER_SET_NAME: latin1
  DEFAULT_COLLATION_NAME: latin1_swedish_ci
      SQL_PATH: NULL
```

Muestra los metadatos asignados a la Base de Datos WORLD_INNODB

INFORMATION_SCHEMA:

- Mostrar los motores de almacenamiento utilizados para las tablas en una base de datos determinada:

```
SELECT TABLE_NAME, ENGINE
FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'world';
```

- Encuentre todas las tablas que contienen columnas SET:

```
SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME
FROM INFORMATION_SCHEMA.COLUMNS
WHERE DATA_TYPE = 'set';
```

- Muestra el número de tablas en cada base de datos:

```
SELECT TABLE_SCHEMA, COUNT(*)
FROM INFORMATION_SCHEMA.TABLES
GROUP BY TABLE_SCHEMA;
```

Comando SHOW

- Mediante este comando, podremos obtener información extraída de los metadatos de MySQL
- Habitualmente se obtiene información que no está disponible en la BBDD INFORMATION_SCHEMA o es complicada de extraer.
 - **SHOW DATABASES:** Bases de Datos existentes en el MySQL Server
 - **SHOW TABLES:** Muestras las tabla existentes en el BBDD en uso
 - **SHOW TABLE STATUS:** Igual que la anterior pero mostrando el estado de la tabla
 - **SHOW COLUMNS . . .:** Información acerca de columnas de una tabla
 - **SHOW INDEX . . .:** Información relacionada con índices de la tabla
 - **SHOW CREATE TABLE** Muestra el comando de creación de la tabla
 - **SHOW CHARACTER** Muestra información relacionada con el conjunto de caracteres
- Los resultados de esta consulta **NO se pueden almacenar para su uso posterior**

Usando SHOW DATABASES

SHOW DATABASES muestra los nombres de las bases de datos disponibles:

```
mysql> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| information_schema |  
| mysql |  
| performance_schema |  
| sakila |  
| test |  
| world |  
+-----+
```

Usando SHOW TABLES

SHOW TABLES muestra las tablas de la base de datos actual::

```
mysql> SHOW TABLES;  
+-----+  
| Tables_in_sakila |  
+-----+  
| actor |  
| actor_info |  
| address |  
| category |  
| ... |  
| staff |  
| staff_list |  
| store |  
+-----+
```


Usando SHOW COLUMNS

SHOW COLUMNS muestra información de estructura de columna para la tabla nombrada en la cláusula FROM:

```
mysql> SHOW COLUMNS FROM film_text;
```

Field	Type	Null	Key	Default	Extra
film_id	smallint(6)	NO	PRI	NULL	
title	varchar(255)	NO	MUL	NULL	
description	text	YES		NULL	

Usando SHOW INDEX

SHOW INDEX muestra información sobre los índices y las columnas indexadas en una tabla determinada:

```
mysql> SHOW INDEX FROM country\G
```

***** 1. row *****
Table: country
Non_unique: 0
Key_name: PRIMARY
Seq_in_index: 1
Column_name: country_id
Collation: A
Cardinality: 109
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
1 row in set (0.00 sec)

SHOW CHARACTER SET y SHOW COLLATION

- **SHOW CHARACTER SET** muestra los juegos de caracteres disponibles junto con sus agrupaciones predeterminadas:

```
mysql> show character set;
```

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
...			

- **SHOW COLLATION** displays the collations for each character set:

```
mysql> show collation;
```

Collation	Charset	Id	Default	Compiled	Sortlen
big5_chinese_ci	big5	1	Yes	Yes	1
big5_bin	big5	84		Yes	1
...					

Usando SHOW con LIKE y WHERE

- Para algunas sentencias **SHOW**, puede agregar una cláusula **LIKE** o **WHERE** para especificar filas a mostrar:

```
mysql> SHOW TABLES LIKE 'sa%';
```

Tables_in_sakila (sa%)
sales_by_film_category
sales_by_store

- Use **HELP SHOW** para ver las sentencias **SHOW** que soportan cláusulas **LIKE** o **WHERE**.

Usando DESCRIBE

- **DESCRIBE**, es otro comando para la visualización de metadatos, es equivalente a **SHOW COLUMNS**.

```
mysql> DESCRIBE film_text;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| film_id    | smallint(6)   | NO   | PRI | NULL    |       |
| title      | varchar(255)  | NO   | MUL | NULL    |       |
| description | text          | YES  |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
```

- **DESCRIBE** puede ser abreviado como **DESC**.

Programa mysqlshow

- **mysqlshow** es un programa en línea que proporciona similar información a la sentencia **SHOW** pero desde línea de comando.
- Puede ser utilizado para:
 - Obtener una lista de los nombres de las bases de datos.
 - Enumerar las tablas dentro de una base de datos.
 - Mostrar información sobre columnas de tablas o índices.

Sintaxis

```
shell> mysqlshow [options] [db_name [tbl_name [col_name ] ] ]
```

```
shell> mysqlshow -u user -ppassword world_innodb
Database: world_innodb
+-----+
| Tables |
+-----+
| City   |
| Country |
| CountryLanguage |
+-----+
```

Al ser un programa en línea, acepta parámetros como **-host** , **-u** , **-p**

Programa `mysqlshow`

- Si indicamos 2 argumentos, `mysqlshow` los interpreta como **BBDD.Tabla** y muestra una salida como **SHOW FULL COLUMNS** de la tabla.

```
shell> mysqlshow -u user -ppassword sakila film_text
Database: sakila Wildcard: film_text
+-----+
| Tables |
+-----+
| film_text |
+-----+
```

- Si hay tres argumentos, `mysqlshow` los interpreta como **BBDD+Tabla+Columna**.
 - El resultado es similar a **SHOW FULL COLUMNS** de una columna
- La opción `--keys` muestra la estructura del índice y da un resultado similar a la de **SHOW INDEX** para esa tabla.

Manipulando datos de la Tabla

Introducción a las operaciones DML

- Los datos en una base de datos es dinámica.
- Los aplicativos pueden realizar operaciones de mantenimiento en ellas con el objetivo de tenerlas actualizadas:
 - Insertar, actualizar, Reemplazar, Borrar datos
- Debemos de tener cuidado con estas operaciones pues puede producir perdida de datos o daños irreparables.

Precauciones:

- Los usuarios sólo tendrán los permisos necesarios para su función.
- Mantener copias de seguridad de los datos (Administrador)
- Verificar los datos a manipular haciendo una `SELECT` de los mismos antes de la operación.
- Habilitar la opción `--safe-updates` o `SET SQL_SAFE_UPDATES=1` para evitar DML sin clausula WHERE

Introducción a las operaciones DML

- Una característica de MySQL es la posibilidad de trabajar con múltiples MOTORES de ejecución (ENGINE).
- Esto implica que el tratamiento de los datos, puede ser diferentes en función del Motor utilizado para la creación de la tabla.
- Hay motores:
 - **Transacciones**
 - Necesitamos realizar una operación de Validación (`COMMIT`) para que los datos sean validados de forma definitiva. (`InnoDB`)
 - Pueden definirse en modo `autocommit` o `NO-autocommit`

```
Mysql> show variables like '%commit%';  
Mysql> SET autocommit=0 / 1
```

- **NO Transaccionales**
 - Las operaciones DML se aplican cuando finaliza la orden.
 - No necesitan ser validados, pues implícitamente se realiza (`MyISAM`)

Operaciones DML: INSERT

Para insertar filas en una tabla se utiliza el comando **INSERT**

- SINTAXIS**

```
INSERT INTO table_name (<column_list>)
VALUES (<value_list>)
```

- Las columnas se identifican por su nombre.
- La asociación de columna y su valor es posicional.
- Los valores deben cumplir con el tipo de datos de la columna
- Los valores constantes de tipo carácter o fecha deben ir encerrados entre comillas simples (' ')

Operaciones DML: INSERT

Insertando una sola fila

```
INSERT INTO CountryLanguage (CountryCode,
Language, Percentage)
VALUES ( 'ALB', 'Chinese', 20);
```

Select * from CountryLanguage

CountryCode	Language	IsOfficial	Percentage
...
ALB	Chinese	F	20.0
...

- Si introducimos todas las columnas, no es necesario de indicarl

```
INSERT INTO CountryLanguage
VALUES ( Valores);
```

Operaciones DML: INSERT con SET

- Utilice **INSERT** con la opción **SET** para indicar nombres de columna y valores.
- Ejemplo:

```
INSERT INTO City SET ID=NULL, Name='Essaouira',
                CountryCode='MAR';

INSERT INTO City SET ID=NULL,
                Name='Sankt-Augustin', CountryCode='DEU';
```

- Contenido de las nuevas filas:

ID	Name	CountryCode	District	Population
4080	Essaouira	MAR		0
4081	Sankt-Augustin	DEU		0

Operaciones DML: INSERT

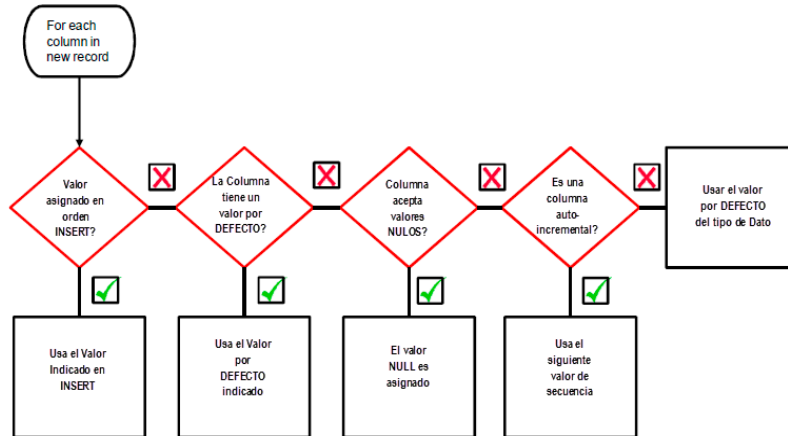
- MySQL permite la inserción de múltiples filas a la vez en una única operación **INSERT**.
- Para ello, incluir múltiples listas de valores de columna, cada uno encerrado dentro de paréntesis y separados por comas

```
INSERT INTO CountryLanguage (CountryCode, Language)
VALUES ('GRL', 'MySQL'),      ← 1st row
      ('FJI', 'MySQL'),      ← 2nd row
      ('BEL', 'MySQL');      ← 3rd row
```

```
INSERT INTO tbl_name (a,b,c) VALUES (1,2,3), (4,5,6), (7,8,9);
```

Operaciones DML: INSERT

- MySQL realiza las siguientes **comprobaciones** antes de realizar una inserción de un registro en las tablas



Operaciones DML: INSERT con SELECT

INSERT INTO ... SELECT

- MySQL permite la inserción de múltiples filas obtenidas mediante resultado de una consulta **SELECT**.

- SINTAXIS

```
INSERT INTO table_name (<column_list>)  
(<query_expression>)
```

- El número y tipo de las columnas recuperadas deben ser iguales a las columnas a insertar

```
INSERT INTO Top10Cities (ID, Name, CountryCode)  
SELECT ID, Name, CountryCode FROM City  
ORDER BY Population DESC LIMIT 10;
```


Insertión de datos en una tabla:

INSERT CON LAST_INSERT_ID

La sentencia **SELECT LAST_INSERT_ID ()** en MySQL recupera el último valor generado para **AUTO_INCREMENT**.

Ejemplo:

```
mysql> INSERT INTO City (Name, CountryCode)
-> VALUES ('Sarah City', 'USA');
Query OK, 1 row affected (0.0 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
| 4080              |
+-----+
```

Recuperar del Last Insert ID de inserción mediante Java

Puede utilizar Java para obtener el último valor **AUTO_INCREMENT** de manera eficiente.

MySQL proporciona el controlador **Connector/J** para la interfaz Java.

- Con **JDBC 3.0**, puede utilizar **getGeneratedKeys()**.

Ejemplo:

```
private void addCities(){
    try{
        Connection c = DriverManager.getConnection(
            "jdbc:mysql://localhost/world_innodb", "sakuser", "sakpwd");
        Statement s = c.createStatement();
        s.executeUpdate("INSERT INTO City (Name, CountryCode) VALUES
            ('Springfield', 'USA'), ('Waterford', 'IRL'), ('Galway',
            'IRL')", Statement.RETURN_GENERATED_KEYS);
        ResultSet rs = s.getGeneratedKeys();
        while(rs.next()){
            System.out.println("Key inserted: " + rs.getInt(1));
        }
    }
}
```

Operación DELETE

- Utilice **DELETE** para quitar filas completas en lugar de columnas individuales.

- **SINTAXIS**

```
DELETE FROM table_name
[WHERE condition]
[ORDER BY ...]
[LIMIT row_count]
```

```
DELETE FROM table_name
```

Si se omite, se borrarán todas las filas de la tabla

- **FROM**

- Indica la tabla de donde se pretende borrar

- **WHERE**

- Condición de selección de las filas que se pretenden borrar.

```
DELETE FROM CountryLanguage WHERE IsOfficial='F';
```

Operación DELETE

- Al igual que ocurre con **Update**, **DELETE** puede ser combinado con las cláusulas **ORDER BY** y **LIMIT**.

ORDER BY

- Ordena los registros seleccionados con el fin de ser eliminados.
- Combinado con LIMIT podremos eliminar los registros ASC o DESC

LIMIT

- Limita el número de registros que se van a eliminar.

```
DELETE FROM CountryLanguage
WHERE Language = 'MySQL'
ORDER BY CountryCode
DESC LIMIT 1;
```

Removed →

CountryCode	Language	IsOfficial	Percentage
BEL	MySQL	F	0.0
GRL	MySQL	F	0.0

DELETE con ORDER BY y LIMIT: Ascendente

- Ejemplo de orden ascendente:

```
DELETE FROM CountryLanguage
WHERE Language = 'MySQL'
ORDER BY CountryCode
ASC LIMIT 1;
```

- Contenido de la fila:

	+-----+-----+-----+-----+			
	CountryCode	Language	IsOfficial	Percentage
Removed →	BEL	MySQL	F	0.0
	FJI	MySQL	F	0.0
	GRL	MySQL	F	0.0

Operaciones DML: UPDATE

- La modificación de los datos ya insertados en una tabla se realiza con el comando **UPDATE**
- **SINTAXIS**

```
UPDATE table_name SET column=expression
[,column=expression,...]
[WHERE condition] [other_clauses]
```

- **SET**

- Indica la columna a modificar y el valor nuevo que tomará la columna

- **WHERE**

- Indica la fila o filas en las que se va a realizar la modificación; si se omite, la modificación se realiza en todas las filas de la tabla

Operaciones DML: UPDATE

- Ejemplo

```
mysql> UPDATE Country
-> SET Population = Population * 2,
-> Region = 'Dolphin Country'
-> WHERE Code = 'SWE';

Query OK, 1 row affected (0.03 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Operaciones UPDATE : Efectos de asignación de columna

- Los efectos están sujetos a restricciones de columnas:
 - Cuando intenta actualizar una columna a un valor que no coincide con la definición de columna, el valor se convierte o trunca por el servidor.
- **UPDATE** no tiene efecto bajo las siguientes condiciones :
 - Cuando no coincide con filas para actualizar
 - Debido a una tabla vacía
 - Si ninguna fila coincide con la cláusula **WHERE**
 - Cuando en realidad no cambia ningún valor de columna
 - Cuando el valor dado es el mismo que el valor existente
 - Si no hay ninguna fila en la tabla que contiene los valores de clave especificados

Operaciones DML: UPDATE

- La actualización de datos se realiza sin orden definido, por lo que en algunas ocasiones nos pueden surgir problemas.

```
mysql> UPDATE City SET ID = ID+1;  
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

- Habitualmente cuando actualizamos columnas Únicas o PK
- Para asegurarnos el orden y que no se produzcan estos errores podemos combinar la clausula **UPDATE** con **ORDER BY**.
- Esta combinación actualiza según el Orden devuelto

```
mysql> UPDATE City SET ID = ID+1  
-> ORDER BY ID DESC;  
Query OK, 4079 rows affected (0.13 sec)  
Rows matched: 4079 Changed: 4079 Warnings: 0
```

Importante no olvidar indicar el ORDEN de las filas ASC / DESC

Operaciones DML: UPDATE

- También podemos utilizar la clausula **LIMIT** para limitar el número de filas actualizadas dentro de todas las posible.

- SINTAXIS**

En clausula **UPDATE** no se permite valores múltiples en **LIMIT**

```
mysql> UPDATE City SET ID = ID-1 LIMIT 1;
```

- La operación anterior modifica sólo la primera aparición encontrada

Resultado

ID	Name	CountryCode	District	Population
1	Kabul	AFG	Kabul	1780000
3	Qandahar	AFG	Qandahar	237500
4	Herat	AFG	Herat	186800
...				

Operaciones DML: REPLACE

- La operación **REPLACE** sólo tiene sentido cuando actuamos sobre tablas que tienen **Primary Key** o **UNIQUE index**.
- Es una extensión de MySQL del SQL estándar.
- Cuando se realiza el **REPLACE** en fila en tablas con **Primary Key** o **UNIQUE index**, se sustituye la fila existente encontrada por los nuevos valores introducidos.

```
REPLACE INTO table_name (<column_list>)  
VALUES (<value_list>)
```

```
REPLACE INTO CountryLanguage (CountryCode, Language,  
Percentage)  
VALUES ('ALB', 'Albaniana', 78.1);
```

Resultados REPLACE

Ejemplo:

```
Query OK, 2 rows affected (0.06 sec)
```

- Devuelve el total para indicar el número de filas afectadas
 - Este recuento es la suma de las filas eliminadas e insertadas.
 - Cuenta de 1: Se insertó una fila y no se eliminaron filas.
 - Contar más de 1: Se eliminaron una o más filas anteriores antes de insertar la nueva fila.
- El recuento de filas hace que sea fácil determinar si **REPLACE** sólo agregó una fila o si también reemplazó las filas.

Algoritmo REPLACE

MySQL realiza los siguientes pasos algorítmicos para **REPLACE** :

1. Intente insertar la nueva fila en la tabla.
2. Cuando la inserción falla porque se produce un error de clave duplicada (para una clave primaria o una restricción única), elimine la fila en conflicto que tenga el valor de la clave duplicada de la tabla.
3. Intente de nuevo insertar la nueva fila en la tabla.

Si la sentencia es aplicada a una tabla sin **Primary Key** o sin o **UNIQUE index**, actúa como una sentencia **INSERT**.

Inserción de filas duplicadas

- Utilice **ON DUPLICATE KEY UPDATE** con **INSERT** para evitar un error de fila duplicado.
- El comportamiento de **ON DUPLICATE KEY UPDATE** es similar a **REPLACE** excepto por lo siguiente:
 - **REPLACE**
 - Descarta la fila existente
 - Agrega la nueva fila a la tabla
 - **ON DUPLICATE KEY UPDATE**
 - Modifica la fila existente
- Esta opción es específica de MySQL.

Eliminación de filas de tablas mediante truncamiento

Utilice la instrucción **TRUNCATE TABLE** para quitar filas de una tabla.

- Esta instrucción siempre elimina todos los registros.

Example:

TRUNCATE [**TABLE**] <table_name>;

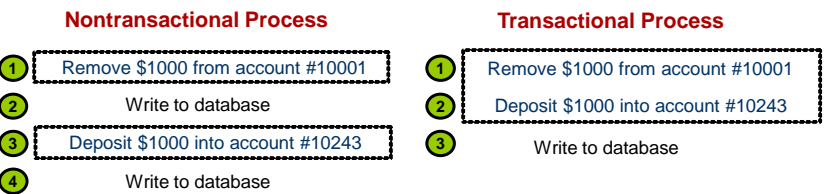
comparado **DELETE** con **TRUNCATE TABLE**:

DELETE	TRUNCATE TABLE
Puede eliminar filas específicas con WHERE	No se pueden eliminar filas específicas; Borra todas
Normalmente se ejecuta más lentamente	Normalmente se ejecuta más rápidamente
Devuelve un valor real de filas	Puede devolver un recuento de fila de cero
Transaccional	No transaccional
	Puede restablecer AUTO_INCREMENT

Transacciones

Procesamiento Transaccional

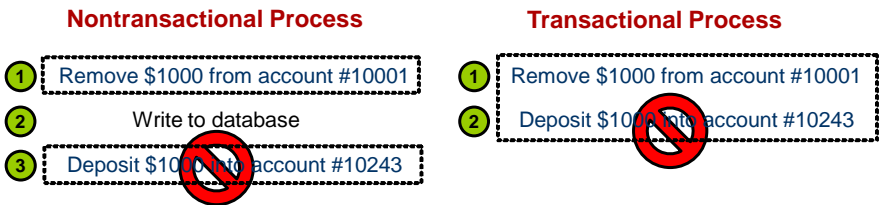
- MySQL dispone de diferentes motores de ejecución y puede tener diferentes operativas:
 - Transaccionales y NO Transaccionales
- Un enfoque NO-Transaccional, con cada sentencia que se ejecuta, se graban los datos.
- Un enfoque Transaccional, las sentencias, que están agrupadas, deben de grabarse conjuntamente



Procesamiento Transaccional

Transacciones

- Una transacción es un grupo de consultas SQL que se tratan atómicamente, como una sola unidad de trabajo.
- El motor de base de datos debe de se capaz de aplicarlo a todo el grupo de consultas.
- El objetivo del enfoque transaccional es asegurar que todos los pasos en una operación tienen éxito, y no aceptar operaciones incorrectas.



Procesamiento Transaccional

Transacciones

- MySQL **no proporciona transacciones a nivel de Servidor**, sino que lo implementan los motores subyacentes.
- Esto significa que no se puede mezclar **con fiabilidad** los diferentes motores en una sola transacción.
- Ejemplo:
 - Si mezclamos tablas transaccionales y no transaccionales (por ejemplo, las tablas MyISAM, InnoDB y) en una transacción, ésta funcionará correctamente si todo va bien.
 - Sin embargo, si se requiere una reversión, los cambios en la tabla no transaccional no se pueden deshacer..
- Es muy importante elegir el motor de almacenamiento adecuado para cada tabla. **MySQL no avisa**

Procesamiento Transaccional: ACID

Los motores transaccionales proporcionan esta integridad de datos mediante **ACID**

- **Atómica:**
 - Todas las declaraciones ejecutar con éxito como una unidad o se cancelan como una unidad.
- **Consistente:**
 - Una base de datos que está en un estado válido cuando comienza una transacción permanece en un estado válido después de la transacción.
- **Aislado:**
 - Una transacción no afecte a otra.
- **Durable:**
 - Todos los cambios realizados por transacciones que completa con éxito se registran correctamente en la base de datos. Los cambios no se pierdan.

Procesamiento Transaccional

Transacciones. Registro de Transacciones

- En los motores Transaccionales, las tablas no se actualizan en disco cada vez que se produce un cambio, se realiza una modificación **in-memory**.
- El motor de almacenamiento escribe un registro de los cambios en el registro de la transacción, que está en el disco.
- Posteriormente, un proceso actualiza los datos físicamente en la tabla.
- Si hay un **Crash** del sistema, los motores Transaccionales son capaces de regenerar la información en las tablas desde el registro de Transacciones.

Procesamiento Transaccional: Sentencias de Control

- Por defecto, MySQL se ejecuta con el modo **AUTO-COMMIT**
 - Esto significa que tan pronto como se ejecuta una sentencia que modifica una tabla, MySQL almacena la actualización en el disco.
- En el modo **AUTO-COMMIT**, cada sentencia individual es considerada como una transacción.
- MySQL por defecto es **AUTO-COMMIT**

```
mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
mysql> SET AUTOCOMMIT = 0;           1 - ON;  0 - Off
```

```
[mysqld]
autocommit=0
```

```
shell> mysql --autocommit=0
```

Procesamiento Transaccional: Sentencias de Control

Disponemos de un conjunto de sentencias para cambiar este funcionamiento:

- **START TRANSACTION (BEGIN)**
 - Comienza una nueva transacción y continua hasta cerrarla expresamente con:
- COMMIT, ROLLBACK o cualquier operación que implique DVL
- **COMMIT**
 - La transacción es validada completamente
- **ROLLBACK**
 - La transacción es revocada y todo se deshace.
- **SET AUTOCOMMIT**
 - Nos permite cambiar el modo de ejecución AUTO-COMMIT / NO-AUTOCOMMIT

```
START TRANSACTION;  
  
SELECT Code FROM Country WHERE Name='Mexico';  
  
UPDATE Country  
SET Name = 'World Cup Winner'  
WHERE Code = 'MEX';  
  
COMMIT;
```

Si estamos en modo **NO-AUTOCOMMIT**, los cambios en tablas transaccionales (como InnoDB) no se hacen permanentes hasta ejecutar **COMMIT**.

Sentencias que causan COMMIT implícitamente

- Algunas sentencias realizan **COMMIT** de forma implícita.
- Al realizar ese **commit**, se realiza una finalización de la transacción en la que se está.
- Estas sentencias son, en sí mismas, no transaccional, lo que significa que no puede realizar un **ROLLBACK** si tienen éxito.

Sentencias:

- Operaciones DDL :
ALTER, CREATE, DROP
- Operaciones DCL:
GRANT, REVOKE, SET PASSWORD
- Operaciones de Bloqueos:
LOCK TABLES, UNLOCK TABLES
- Operaciones BULK:
TRUNCATE TABLE, LOAD DATA INFILE

Lecturas Consistentes

- Cuando varios clientes, al mismo tiempo, acceden a los datos de la misma tabla, se pueden presentar los siguientes problemas de coherencia:

Dirty reads: (lectura sucia)

- Una lectura sucia ocurre cuando una transacción lee los cambios realizados por otra transacción no confirmada.

Nonrepeatable reads:

- Se produce cuando la misma operación de lectura produce diferentes resultados cuando se repite en un momento posterior dentro de la misma transacción.

Phantom reads: (Lectura fantasma)

- Se produce cuando aparece una fila que no era, previamente, visible dentro de la misma transacción.

Lecturas Consistentes

Ejemplos

Dirty reads: (lectura sucia)

- La transacción T1 modifica una fila.
La transacción T2 lee la fila y ve la modificación aunque T1 no la haya validado (lectura sucia)

Si T1 deshace, el cambio se deshace pero T2 no sabe eso.

Nonrepeatable reads:

- Una transacción T1 lee algunas filas.
Otra transacción T2 cambia algunas de esas filas y confirma los cambios.

Si T1 es capaz de ver los cambios realizados en la transacción T2 al recuperar las filas de nuevo, la lectura inicial resultó ser Nonrepeatable.

Problema: T1 no consigue un resultado coherente en la misma consulta dentro de la misma transacción.

Phantom reads: (Lectura fantasma)

- Las transacciones T1 y T2 comienzan, y T1 lee algunas filas.
Si T2 inserta una nueva fila sin validar.

Si T1 ve la fila cuando se repite la misma operación de lectura, sería una lectura fantasma

Niveles de Aislamiento

- MySQL implementa **4 niveles de aislamiento** para controlar los cambios realizados por transacciones y como afectan a otras transacciones concurrentes.

READ UNCOMMITTED

- Permite a una transacción ver los cambios no confirmados realizados por otras transacciones.

READ COMMITTED

- Permite una transacción para ver los cambios realizados por otras transacciones sólo si han sido Validados.

REPEATABLE READ (default)

- Garantiza que si una transacción emite el mismo SELECT dos veces, obtiene el mismo resultado en ambas ocasiones.

SERIALIZABLE

- Aísla completamente los efectos de todas las transacciones entre si.

Niveles de Aislamiento

- La siguiente tabla muestra la relación entre los diferentes problemas de consistencia en lecturas y los diferentes niveles de aislamiento

Isolation Level	Dirty reads	Nonrepeatable reads	Phantom reads
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITTED	Not possible	Possible	Possible
REPEATABLE READ	Not possible	Not possible	Not possible
SERIALIZABLE	Not possible	Not possible	Not possible

```
SELECT @@GLOBAL.tx_isolation, @@tx_isolation;
SET GLOBAL tx_isolation='REPEATABLE-READ';
SET SESSION tx_isolation='SERIALIZABLE';
```

Configurar Niveles de Aislamiento

- El nivel de aislamiento por defecto en MySQL es **REPEATABLE READ**.
 - Garantiza que si una transacción emite el mismo SELECT dos veces, obtiene el mismo resultado en ambas ocasiones.
- Para conocer el nivel de aislamiento actual variable `tx_isolation`

```
mysql> SELECT @@tx_isolation;
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+
1 row in set (0.001 sec)
```

```
mysql> SELECT @@global.tx_isolation
+-----+
| @@tx_isolation |
+-----+
| REPEATABLE-READ |
+-----+
1 row in set (0.001 sec)
```

Configurar Niveles de Aislamiento

- Podemos cambiar el nivel de aislamiento a nivel:
 - **Fichero de opciones**

```
[mysqld]
transaction-isolation = READ-COMMITTED
```

- **GLOBAL**
 - Es aplicado a todas las nuevas conexiones.
 - Necesitamos privilegio de SUPER para cambiarlo

```
mysql> SET GLOBAL TRANSACTION ISOLATION LEVEL
-> READ COMMITTED;
```

- **SESSION**
 - Cualquier cliente puede modificar el nivel de aislamiento de su propia sesión,

```
mysql> SET TRANSACTION ISOLATION LEVEL
-> READ COMMITTED;
```

Bloqueos

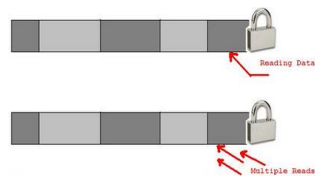
- Los bloqueos aparecen en MySQL para impedir problemas que pueden ocurrir cuando dos o más clientes accedan a los mismos datos al mismo tiempo.
- El bloqueo permite el acceso a los datos por parte del cliente que mantiene el bloqueo, pero limita lo que otros clientes pueden hacer con los datos
- Cuando varios clientes quieren modificar los datos, deben esperar su turno.



Bloqueos

MySQL es compatible con dos tipos de bloqueo:

- **SHARED**
 - Si un cliente quiere leer los datos, otros clientes que quieren leer los mismos datos no causan un conflicto, y todos ellos pueden leer al mismo tiempo.
 - Sin embargo, otro cliente que quiere escribir (modificar) datos debe esperar hasta que la lectura ha terminado.



- **EXCLUSIVE**
 - Si un cliente quiere escribir los datos, todos los demás clientes debe esperar hasta que la escritura ha terminado antes de leer o escribir.



Bloqueos

- Podemos modificar nuestras sentencias **SELECT** para que actúen de forma diferente a lo anteriormente indicado.
- Con el nivel de aislamiento **READ REPEATABLE**, puede utilizar **SELECT ... LOCK IN SHARE MODE**
 - Otras sesiones pueden leer las filas, pero no pueden modificarlas hasta que se compromete la transacción.
- Esto es similar a operar en nivel de aislamiento **SERIALIZABLE**.

```
mysql> SELECT Code FROM Country WHERE Name='Australia'  
-> LOCK IN SHARE MODE;
```

- Si la sentencia **SELECT** procesa filas que se han modificado por otra transacción, **LOCK IN SHARE MODE** **bloquea la operación SELECT** hasta que la transacción se confirme.

Bloqueos

SELECT ... LOCK IN SHARE MODE

- Seleccionamos el valor y lo bloqueamos para que no lo modifiquen

```
mysql> SELECT Code FROM Country WHERE Name='Australia'  
-> LOCK IN SHARE MODE;
```

- Realizamos la inserción de los valores.

```
mysql> INSERT INTO ..... Values ( 'Australia' ,,,,,,);
```

- Realizamos la validación de los datos

```
mysql> COMMIT
```

Bloqueos

- También disponemos de la clausula **FOR UPDATE** en sentencias **SELECT**.
- Esta clausula es utilizada para asegurarnos que nuestros datos no son modificamos mientras los hemos leído y estamos trabajando con ellos.

Sintaxis:

```
SELECT ... FOR UPDATE;
```

1. Bloqueamos los datos que posteriormente vamos a modificar.

```
mysql> SELECT counter INTO @@counter FROM CityCodes  
-> FOR UPDATE;
```

2. Los modificamos con tranquilidad y validamos

```
mysql> UPDATE CityCodes SET counter = @@counter + 1;
```