

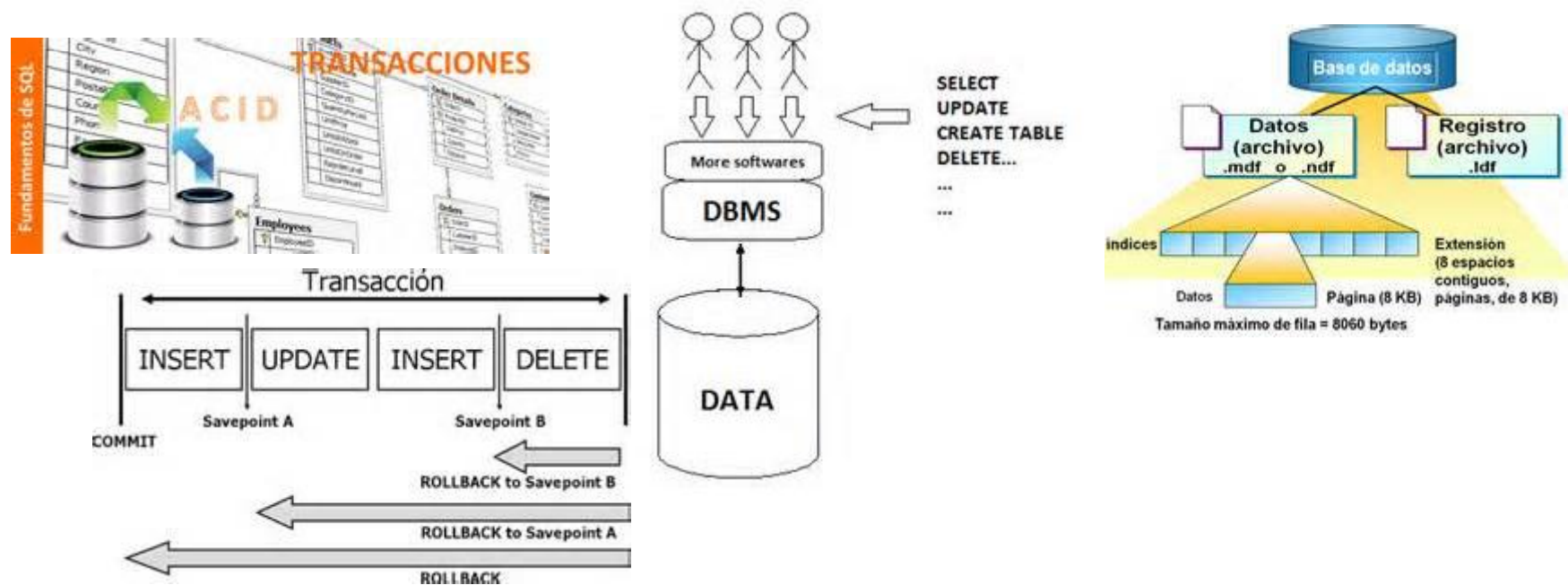
U.T.6

MANIPULACIÓN DE DATOS.


TRANSACCIONES.

GESTIÓN DE USUARIOS Y

PERMISOS EN MYSQL



Creación, supresión y modificación de tablas, vistas y otros objetos

 Views o Vistas

View: Los views son como tablas virtuales que contienen una búsqueda pre-armada muy utiles a la hora de generar reportes.

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}] VIEW
nombre_vista [(columnas)] AS sentencia_select [WITH [CASCADED | LOCAL] CHECK
OPTION]
```

< La ventaja de los views, es que están precompilados en el motor, lo que su acceso y ejecución se torna más rápido. >

```
DROP VIEW IF EXISTS `NombreSchema`.`viewabono_control`;
```

```
CREATE VIEW `Nombreschema`.`nombreview` AS SELECT * FROM tabla;
```

Las vistas comparten el mismo espacio de nombre que las tablas, por lo que no pueden tener los mismo nombres

Las vistas pueden ser usadas en otros comandos SELECT con todas sus opciones

CREACIÓN CON SELECT DE TABLAS

Se puede realizar la creación de una tabla con datos recuperados en una consulta.

La sentencia `CREATE TABLE` permite crear una tabla a partir de la consulta de otra tabla ya existente. La nueva tabla contendrá los datos obtenidos en la consulta. Se lleva a cabo esta acción con la cláusula `AS` colocada al final de la orden `CREATE TABLE`.

No es necesario especificar tipos ni tamaño de las columnas, ya que vienen determinados por los tipos y los tamaños de los recuperados en la consulta. La consulta puede contener una subconsulta, una combinación de tablas o cualquier sentencia `SELECT` válida.

Las restricciones no se crean en la nueva tabla desde la otra.

SINTAXIS 1

- **CREATE TABLE [IF NOT EXISTS] nueva_tabla
SELECT * FROM tabla_origen [WHERE];**

```
mysql> create table if not exists mascotasC select * from mascotas;  
Query OK, 12 rows affected (0.14 sec)  
Records: 12  Duplicates: 0  Warnings: 0  
  
mysql> show create table mascotasC;  
+-----+
```

```
+-----+  
mascotasC | CREATE TABLE `mascotasC` (  
  `idMascota` int(11) NOT NULL DEFAULT '0',  
  `nombre` varchar(30) NOT NULL,  
  `especie` varchar(20) NOT NULL DEFAULT 'canina',  
  `raza` varchar(20) DEFAULT NULL,  
  `pedigree` tinyint(1) DEFAULT NULL,  
  `fechaNacimiento` date DEFAULT NULL,  
  `sexo` char(1) DEFAULT NULL,  
  `propietario` varchar(10) DEFAULT NULL,  
  `nose` int(11) DEFAULT NULL  
ENGINE=InnoDB DEFAULT CHARSET=latin1 |  
+-----+
```

SINTAXIS 1

- **CREATE TABLE [IF NOT EXISTS] nueva_tabla AS SELECT campos FROM tabla_origen [WHERE];**
- Crea la tabla con la misma estructura y copia todos los registros pero sin las restricciones:
- Obviamente habría que añadirle las restricciones posteriormente

```
mysql> create table if not exists mascotasC select idMascota,nombre,especie
from mascotas;
Query OK, 12 rows affected (0.09 sec)
Records: 12 Duplicates: 0 Warnings: 0
mysql>
```

SINTAXIS 2

- **CREATE TABLE [IF NOT EXISTS] nueva_tabla LIKE tabla_origen;**
- Crea la tabla con la misma estructura y con la restricción de clave primaria.

```
| amigos | CREATE TABLE `amigos` (  
  `IdAmigo` int(11) NOT NULL,  
  `nombre` varchar(30) DEFAULT NULL,  
  `telefono` varchar(10) DEFAULT NULL,  
  `trabajo` varchar(15) DEFAULT NULL,  
  PRIMARY KEY (`IdAmigo`),  
  CONSTRAINT `amigos_ibfk_1` FOREIGN KEY (`IdAmigo`) REFERENCES `emple` (`EMP_NO`) ON DELETE CASCADE ON UPDATE CASCADE  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |  
+-----+
```

mysql> create table if not exists amigo2 like amigos;

SINTAXIS 2

- Vemos cómo la nueva tabla se ha creado con la estructura de la tabla origen, con su clave primaria, pero no se ha conservado la clave ajena. Hay que añadirla posteriormente, a través de alter.

```
| amigo2 | CREATE TABLE `amigo2` (  
  `IdAmigo` int(11) NOT NULL,  
  `nombre` varchar(30) DEFAULT NULL,  
  `telefono` varchar(10) DEFAULT NULL,  
  `trabajo` varchar(15) DEFAULT NULL,  
  PRIMARY KEY (`IdAmigo`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |  
+-----+  
-----+  
-----+  
-----+  
1 row in set (0.00 sec)  
  
mysql> select * from amigo2;  
Empty set (0.01 sec)
```

SINTAXIS 2

- mysql> alter table amigo2 add foreign key (idamigo) references emple (emp_no) on delete cascade on update cascade;

```
amigo2 | CREATE TABLE `amigo2` (  
  `IdAmigo` int(11) NOT NULL,  
  `nombre` varchar(30) DEFAULT NULL,  
  `telefono` varchar(10) DEFAULT NULL,  
  `trabajo` varchar(15) DEFAULT NULL,  
  PRIMARY KEY (`IdAmigo`),  
  CONSTRAINT `amigo2_ibfk_1` FOREIGN KEY (`IdAmigo`) REFERENCES `emple` (`  
P_NO`) ON DELETE CASCADE ON UPDATE CASCADE  
ENGINE=InnoDB DEFAULT CHARSET=latin1 |  
-----+
```


Supresión de tablas

- La orden SQL DROP TABLE suprime una tabla de la base de datos. Cada usuario puede borrar sus propias tablas; sólo el administrador de la base de datos o algún usuario con el privilegio DROP ANY TABLE puede borrar las tablas de otro usuario.
- Al suprimir una tabla también se suprimen los índices y los privilegios asociados a ella. Las vistas y los sinónimos creados a partir de esta tabla dejan de funcionar, pero siguen existiendo en la base de datos, por lo que habría que eliminarlos. El formato de la orden DROP TABLE es:

```
DROP TABLE [usuario].nombretabla [CASCADE CONSTRAINTS];
```

- CASCADE CONSTRAINTS elimina las restricciones de integridad referencial que remitan a la clave primaria de la tabla borrada.

Inserción de datos. Orden INSERT

- Empezamos la manipulación de datos de una tabla con la orden **INSERT**. Con ella se añaden filas de datos en una tabla.
- Es posible introducir los valores directamente en la sentencia u obtenerlos a partir de la información existente en la base de datos mediante la inclusión de una consulta haciendo uso de la sentencia SELEC.

Sentencia INSERT

- **INSERT [INTO] Tabla [(campo1, campo2, ..., campoN)] VALUES (valor1, valor2, ..., valorN)**

```
mysql> INSERT INTO mascotas (idMascota,nombre,raza) VALUES (null,'peca','dogo');  
Query OK, 1 row affected (0.05 sec)  
mysql>
```

- **INSERT INTO Tabla VALUES (valor1, valor2, ..., valorN)**

Si no se especifica la lista de columnas, hay que indicar todos los valores para todas las columnas en el orden en que están definidas las columnas en la tabla.

```
mysql> insert into propietarios values ('11111F','Juan Martinez'), ('11111G','Ana Gomez');  
Query OK, 2 rows affected (0.02 sec)  
Records: 2 Duplicates: 0 Warnings: 0
```

INSERT Y SELECT

- **INSERT [INTO] TablaDestino SELECT * FROM TablaOrigen [WHERE...]**

De esta forma los campos de TablaOrigen se grabarán en TablaDestino, para realizar esta operación es necesario que todos los campos de TablaOrigen estén contenidos con igual nombre en TablaDestino, es decir, que TablaDestino posea todos los campos de TablaOrigen (coincidan en orden y tipo).

```
mysql> create table if not exists mascotasB like mascotas;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> insert into mascotasB select * from mascotas where raza like 'dogo';

Query OK, 2 rows affected (0.02 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

INSERT Y SELECT

- Para insertar sólo algunos campos
- Se puede también especificar los campos que queremos copiar, y de esta forma que tabla destino sólo tenga los campos que me interesen.
- En este caso la sintaxis es:
INSERT INTO Tabla Destino [(campo1, campo2, , campoN)] SELECT [campo1, campo2, , campoN] FROM Tabla Origen [WHERE...]

```
mysql> insert into mascotasB (idMascota,nombre,especie,raza) select idMasco  
ta, nombre,especie,raza from mascotas where nombre like 'Donna';  
Query OK, 1 row affected (0.00 sec)  
Records: 1 Duplicates: 0 Warnings: 0  
  
mysql> select * from mascotasB;
```

UPDATE

- **UPDATE Tabla SET Campo1=Valor1 [, Campo2=Valor2, CampoN=ValorN] WHERE filtro**

UPDATE es especialmente útil cuando se desea cambiar un gran número de registros o cuando éstos se encuentran en múltiples tablas. Puede cambiar varios campos a la vez.

UPDATE no genera ningún resultado. Para saber qué registros se van a cambiar, hay que examinar primero el resultado de una consulta de selección que utilice el mismo criterio y después ejecutar la consulta de actualización.

- Si en una consulta de actualización suprimimos la cláusula WHERE todos los registros de la tabla señalada serán actualizados.

UPDATE

```
mysql> select nombre,gama,precioVenta,precioProveedor from productos where
gama='herramientas';
```

nombre	gama	precioVenta	precioProveedor
Sierra de Poda 400MM	Herramientas	13.86	10.89
Pala	Herramientas	13.86	12.87
Rastrillo de Jard?	Herramientas	11.88	10.89
Azad??n	Herramientas	11.88	10.89

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

```
mysql> #Incrementar un 10% el precioVenta y precioVendedor de los productos
de la gama 'herramientas' en la BD jardineria
mysql> update productos set precioventa=precioventa*1.1, precioProveedor=pr
ecioProveedor*1.1 where gama like 'herramientas';
Query OK, 4 rows affected, 8 warnings (0.04 sec)
Rows matched: 4 Changed: 4 Warnings: 8
```

```
mysql> select nombre,gama,precioVenta,precioProveedor from productos where
gama like 'herramientas';
```

nombre	gama	precioVenta	precioProveedor
Sierra de Poda 400MM	Herramientas	15.25	11.98
Pala	Herramientas	15.25	14.16
Rastrillo de Jard?	Herramientas	13.07	11.98
Azad??n	Herramientas	13.07	11.98

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

UPDATE

- Aumenta en 100 euros el salario y en 10 la comisión a todos los empleados del departamento 10 de la tabla EMPLE:
- Update emple set salario=salario+100, comision=comision+10 where dept_no=10;

```
mysql> select salario,comision from emple where dept_no=10;
+-----+-----+
| salario | comision |
+-----+-----+
|    2985 |      NULL |
|    4200 |      NULL |
|     100 |        10 |
+-----+-----+
3 rows in set (0.02 sec)

mysql> Update emple set comision=10 where dept_no=10 and comision is null;
Query OK, 2 rows affected (0.69 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> select salario,comision from emple where dept_no=10;
+-----+-----+
| salario | comision |
+-----+-----+
|    2985 |        10 |
|    4200 |        10 |
|     100 |        10 |
+-----+-----+
```


UPDATE CON SELECT

- Podemos incluir una subconsulta en una sentencia UPDATE que puede estar contenida en la cláusula WHERE o puede formar parte de SET. Cuando la subconsulta (orden SELECT) forma parte de SET, debe seleccionar una única fila y el mismo número de columnas (con tipos de datos adecuados) que las que hay entre paréntesis al lado de SET.
- Los formatos son:

```
UPDATE <NombreTabla> SET columna1 = valor1, columna2 =  
valor2, ... WHERE columna3 = (SELECT ...);
```

UPDATE CON SELECT

```
UPDATE <NombreTabla> SET (columna1, columna2, ...) = (SELECT col1, col2, ...)
WHERE condición;
UPDATE <NombreTabla> SET columna1 = (SELECT col1 ...), columna2 = (SELECT
col2 ...) WHERE condición;
```

En la tabla CENTROS igualar la dirección y el número de plazas del código de centro 10 a los valores de las columnas correspondientes que están almacenadas para el código de centro 50. (Oracle)

```
SQL> UPDATE CENTROS SET (DIRECCION, NUM_PLAZAS) = (SELECT
DIRECCION, NUM_PLAZAS FROM CENTROS WHERE COD_CENTRO = 50)
WHERE COD_CENTRO= 10;
```

A partir de la tabla EMPLE, cambia el salario a la mitad y la comisión a 0, a aquellos empleados que pertenezcan al departamento con mayor número de empleados (Oracle).

```
SQL> UPDATE EMPLE SET SALARIO = SALARIO/2, COMISION = 0 WHERE
DEPT_NO =(SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO HAVING
COUNT(*) >=ALL(SELECT COUNT(*) FROM EMPLE GROUP BY DEPT_NO));
```

UPDATE CON SELECT

- Para todos los empleados de la tabla EMPLE y del departamento de 'CONTABILIDAD', cambiamos su salario al doble del salario de 'SÁNCHEZ' y su apellido, a minúscula. (Oracle)
- SQL> UPDATE EMPLE SET APELLIDO = LOWER(APELLIDO), SALARIO = (SELECT SALARIO*2 FROM EMPLE WHERE APELLIDO = 'SANCHEZ') WHERE DEPT_NO = (SELECT DEPT_NO FROM DEPART WHERE DNOMBRE = 'CONTABILIDAD');

En mysql

- `mysql> set @salarioSanchez=(select salario from emple where apellido='Sanchez');`
- `mysql> UPDATE EMPLE SET APELLIDO = LOWER(APELLIDO), SALARIO = @salarioSanchez*4 where dept_no=(select dept_no from depart where dnombre='contabilidad');`

UPDATE

- Si se omite el filtro, el resultado es la modificación de todos los registros de la tabla.

```
mysql> select numHi from emple2;
+-----+
| numHi |
+-----+
| NULL  |
| NULL  |
| NULL  |
| NULL  |
| 1     |
| 0     |
| 0     |
| 7     |
+-----+
8 rows in set (0.00 sec)

mysql> update emple2 set numhi=1;
Query OK, 7 rows affected (0.042 sec)
Rows matched: 8  Changed: 7  Warnings: 0
```

```
mysql> select numHi from emple2;
+-----+
| numHi |
+-----+
| 1     |
| 1     |
| 1     |
| 1     |
| 1     |
| 1     |
| 1     |
| 1     |
+-----+
8 rows in set (0.00 sec)
```

Borrado de filas. Orden DELETE

- Para eliminar una fila o varias filas de una tabla se usa la **orden DELETE**. La **cláusula WHERE** es esencial para eliminar sólo aquellas filas deseadas. Sin la cláusula WHERE, DELETE borrará todas las filas de la tabla. El espacio usado por las filas que han sido borradas no se reutiliza, a menos que se realice un EXPORT / IMPORT. La condición puede incluir una subconsulta. Éste es su formato:
- **DELETE [FROM] NombreTabla WHERE Condición;**

DELETE

- **DELETE FROM Tabla WHERE [filtro]**

Una vez que se han eliminado los registros utilizando una consulta de borrado, no puede deshacer la operación, a no ser que use rollback. Si desea saber qué registros se eliminarán, primero examine los resultados de una consulta de selección que utilice el mismo criterio y después ejecute la consulta de borrado. Mantenga copias de seguridad de sus datos en todo momento.

- Si elimina los registros equivocados podrá recuperarlos desde las copias de seguridad.

DELETE

- **Borramos los registros con COD_CENTRO =50 de la tabla CENTROS:**
`SQL> DELETE FROM CENTROS WHERE COD_CENTRO=50;`
- **Borramos todas las filas de la tabla CENTROS:**
`SQL> DELETE FROM CENTROS;`
- **Borramos todas las filas de la tabla LIBRERIA cuyos EJEMPLARES no superen la media de ejemplares en su ESTANTE (Oracle) :**
`SQL> DELETE FROM LIBRERIA L WHERE EJEMPLARES <(SELECT
AVG(EJEMPLARES) FROM LIBRERIA WHERE ESTANTE = L.ESTANTE
GROUP BY ESTANTE);`
- **Borramos los departamentos de la tabla DEPART con menos de cuatro empleados. (Oracle y Mysql)**
`SQL> DELETE FROM DEPART WHERE DEPT_NO IN
(SELECT DEPT_NO FROM EMPLE GROUP BY DEPT_NO HAVING COUNT(*) <
4);`

DELETE

- Borra todos los departamentos de la tabla depart para los cuales no existan empleados en emple:
- delete from depart2 where dept_no NOT IN (select dept_no from emple2);

```
mysql> select * from depart2;
```

DEPT_NO	DNOMBRE	NUMCE	DIREC	TDIR	PRESU	DEPDE
10	CONTABILIDAD	2	7782	F	-1233.80005	30
20	INVESTIGACION	1	7566	P	12346.89746	NULL
30	VENTAS	2	7698	P	30000.00000	40

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

Eliminación de todos los registros

- Cuando queremos eliminar todas la filas de una tabla, se puede usar una sentencia DELETE sin condiciones. Sin embargo, existe una sentencia alternativa, TRUNCATE, que realiza la misma tarea de una forma mucho más rápida.
- La diferencia es que DELETE hace un borrado secuencial de la tabla, fila a fila.
Pero TRUNCATE borra la tabla y la vuelve a crear vacía, lo que es mucho más eficiente.

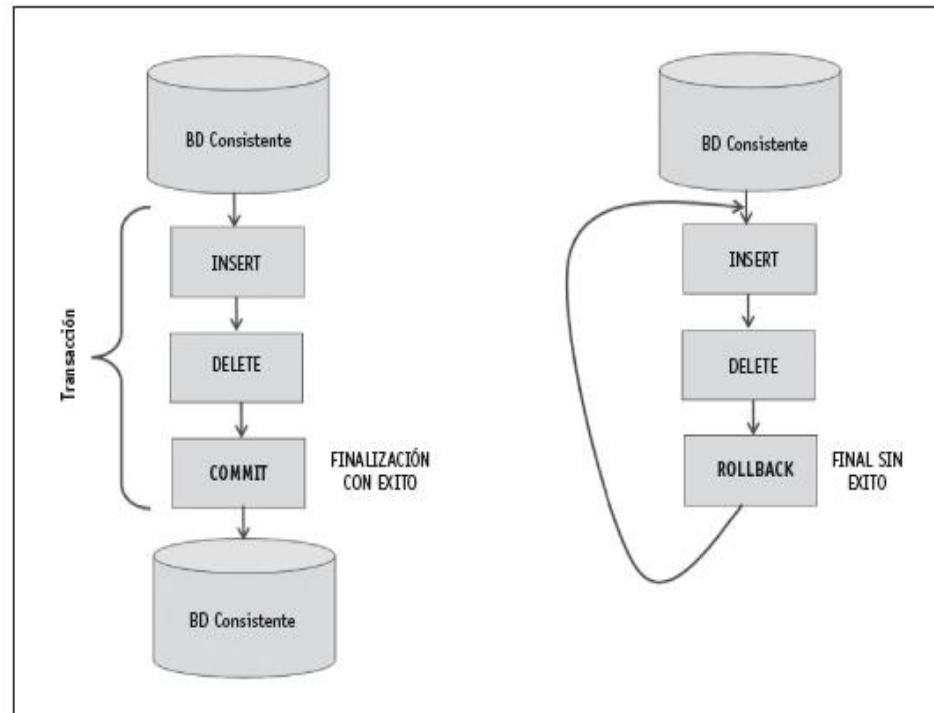
Supresión de tablas

- La orden SQL DROP TABLE suprime una tabla de la base de datos. Cada usuario puede borrar sus propias tablas; sólo el administrador de la base de datos o algún usuario con el privilegio DROP ANY TABLE puede borrar las tablas de otro usuario.
- Al suprimir una tabla también se suprimen los índices y los privilegios asociados a ella. Las vistas y los sinónimos creados a partir de esta tabla dejan de funcionar, pero siguen existiendo en la base de datos, por lo que habría que eliminarlos. El formato de la orden DROP TABLE es:

```
DROP TABLE [usuario].nombretabla [CASCADE CONSTRAINTS];
```

- CASCADE CONSTRAINTS elimina las restricciones de integridad referencial que remitan a la clave primaria de la tabla borrada.

TRANSACCIONES



TRANSACCIONES

- **ROLLBACK, COMMIT Y AUTOCOMMIT**

```
mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | ON    |
+-----+-----+
1 row in set (0.00 sec)
```

```
mysql> SET AUTOCOMMIT=OFF;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'AUTOCOMMIT';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| autocommit    | OFF   |
+-----+-----+
1 row in set (0.00 sec)
```

MySQL tiene una variable de entorno llamada **autocommit**, que por defecto tiene el valor 1. Configurado de esta manera no se pueden usar transacciones, porque MySQL automáticamente hace un COMMIT después de cada consulta.

Para usar transacciones hay que poner autocommit a 0 (desactivarlo).

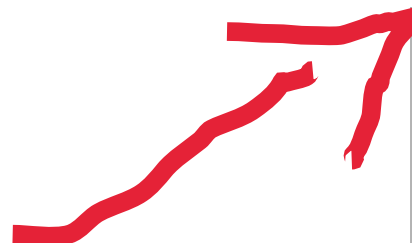
Transacción

- Una transacción es un conjunto de operaciones que van a ser tratadas como una única unidad. Estas transacciones deben cumplir 4 propiedades fundamentales comúnmente conocidas como ACID (atomicidad, coherencia, aislamiento y durabilidad).
- La transacción más simple en SQL es una única sentencia SQL. Por ejemplo una sentencia como esta:

UPDATE Products SET UnitPrice=20 WHERE ProductName ='Chai'

Es una transacción.

Esta es una transacción 'autocommit', una transacción autocompletada.



```
mysql> SHOW VARIABLES LIKE 'a%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| auto_increment_increment | 1 |
| auto_increment_offset | 1 |
| autocommit | ON |
| automatic_sp_privileges | ON |
+-----+-----+
4 rows in set (0.00 sec)
```

ROLLBACK, COMMIT Y AUTOCOMMIT

- **Commit:**

Cuando ejecutamos ordenes estas no son creadas en la tabla hasta que ponemos esta orden, por tanto los cambios realizados se perderán si al salir del programa no realizamos esta acción. Puede programarse para que lo haga automáticamente.

Algunas órdenes llevan COMMIT implícito:

Órdenes con COMMIT implícito:

QUIT

EXIT

CONNECT

DISCONNECT

CREATE TABLE

CREATE VIEW

GRANT

REVOKE

DROP TABLE

DROP VIEW

ALTER

AUDIT

NO AUDIT

ALTER TABLE

ALTER VIEW

TRANSACCIONES

- Por defecto, MySQL se ejecuta en modo autocommit. Esto significa que tan pronto como se ejecuta una sentencia se actualiza (modifica) la tabla, MySQL almacenará la actualización en disco.
- Si se están usando tablas de transacción segura (como **InnoDB** o **BDB**), se puede poner MySQL en modo no-autocommit con el comando siguiente:
- **SET AUTOCOMMIT=0**

TRANSACCIONES

- Después de desconectar el modo autocommit asignando cero a la variable *AUTOCOMMIT*, se debe usar **COMMIT** para almacenar los cambios en disco o **ROLLBACK** si se quieren ignorar los cambios hechos desde el principio de la transacción.
- Si se quiere desactivar el modo autocommit para una serie de sentencias, se puede usar una sentencia **START TRANSACTION**:

```
START TRANSACTION;  
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;  
UPDATE table2 SET summary=@A WHERE type=1;  
COMMIT;
```

TRANSACCIONES

- Se puede usar **BEGIN** y **BEGIN WORK** en lugar de **START TRANSACTION** para iniciar una transacción.
- Si no se están usando tablas de transacción segura, cualquier cambio será almacenado inmediatamente, independientemente del estado del modo autocommit.
- Si se usa una sentencia **ROLLBACK** después de actualizar una tabla no transaccional, se obtendrá un error (ER_WARNING_NOT_COMPLETE_ROLLBACK) como un aviso. Todas las tablas de transacción segura serán restauradas, pero cualquier tabla de transacción no segura no cambiará.

ACID

- Al ejecutar una transacción, el motor de base de datos nos garantizará la **atomicidad, consistencia, aislamiento y durabilidad (ACID, acrónimo de Atomicity, Consistency, Isolation and Durability)** de la transacción (o conjunto de comandos) que se utilice.
- **Atomicidad:** es la propiedad que asegura que la operación se ha realizado o no, y por lo tanto ante un fallo del sistema no puede quedar a medias. Se dice que una operación es atómica cuando en una operación que consiste en una serie de pasos, todos ellos se realizan o ninguno. Por ejemplo, en el caso de una transacción bancaria o se ejecuta tanto el depósito como la deducción o ninguna acción es realizada.

ACID

- **Consistencia o *Integridad***. Es la propiedad que asegura que sólo se empieza aquello que se puede acabar. Por lo tanto se ejecutan aquellas operaciones que no van a romper las reglas y directrices de integridad de la base de datos. La propiedad de consistencia sostiene que cualquier transacción llevará a la base de datos desde un estado válido a otro también válido. "La Integridad de la Base de Datos nos permite asegurar que los datos son exactos y consistentes".

ACID

- **Aislamiento:** es la propiedad que asegura que una operación no puede afectar a otras. Esto asegura que la realización de dos transacciones sobre la misma información sean independientes y no generen ningún tipo de error. Esta propiedad define cómo y cuándo los cambios producidos por una operación se hacen visibles para las demás operaciones concurrentes.
- **Durabilidad o *Persistencia*.** Es la propiedad que asegura que una vez realizada la operación, ésta persistirá y no se podrá deshacer aunque falle el sistema.
- Cumpliendo estos 4 requisitos un sistema gestor de bases de datos puede ser considerado *ACID*.

Desactivado el modo autocommit

- `START TRANSACTION;`
- `SELECT @A:=SUM(salary) FROM table1
WHERE type=1;`
- `UPDATE table2 SET summary=@A WHERE
type=1;`
- `COMMIT;`

EL SISTEMA DE PERMISOS MYSQL

- MySQL precisa de consultar las tablas de **mysql** y **performance_schema**, así como las variables de configuración para la verificación del sistema y su buen funcionamiento.

```

Tables in performance schema
+-----+
| accounts |
| cond_instances |
| events_stages_current |
| events_stages_history |
| events_stages_history_long |
| events_stages_summary_by_account_by_event_name |
| events_stages_summary_by_host_by_event_name |
| events_stages_summary_by_thread_by_event_name |
| events_stage:socket_instances |
| events_stage:socket_summary_by_event_name |
| events_stage:socket_summary_by_instance |
| events_stat:table_io_waits_summary_by_index_usage |
| events_stat:table_io_waits_summary_by_table |
| events_stat:table_lock_waits_summary_by_table |
| events_stat:threads |
| events_stat:users |
+-----+

52 rows in set (0.00 sec)

```

```
mysql> show tables;
```

```
+-----+
| Tables_in_mysql |
+-----+
| columns_priv |
| db |
| event |
| func |
| general_log |
```

```
| time_zone  
| time_zone_leap_second  
| time_zone_name  
| time_zone_transition  
| time_zone_transition_type  
| user  
+-----+  
28 rows in set (0.00 sec)
```


Gestión de usuarios en MYSQL

- Cuando un usuario intenta acceder a la base de datos existen dos etapas de verificación:
- La primera es comprobar que el usuario existe, su contraseña, desde que host puede conectarse y qué permisos generales tiene. Esto se hace consultando las tablas `user`, `hosts` y `db`.
- La segunda es comprobar los privilegios de acceso en cada uno de los objetos a los que accede, cosa que se ira verificando durante toda la sesión. Para ello tenemos las tablas *`db`*, *`tables_priv`* y *`columns_priv`* que proporcionan permisos a nivel de base de datos, tabla y columna.
- Para saber que usuario está conecta actualmente existe la función:
`SELECT CURRENT_USER();`

Gestión de usuarios en MYSQL

- Para comprobar los permisos completos de un usuario podemos usar el comando :

```
mysql> show grants for root@localhost;
+-----+
| Grants for root@localhost |
+-----+
| GRANT ALL PRIVILEGES ON *.* TO 'root'@'localhost' IDENTIFIED BY PASSWORD '5C2F5435B8AE48A558096F128C6595CD7D969AEC' WITH GRANT OPTION |
| GRANT PROXY ON ''@'' TO 'root'@'localhost' WITH GRANT OPTION |
+-----+
2 rows in set (0.00 sec)
```

CREACIÓN Y ELIMINACIÓN DE USUARIOS

- Hay tres formas de crear un usuario:

1- Con la sentencia **CREATE USER**:

```
CREATE USER usuario@host IDENTIFIED BY 'password'
```

- Si omitimos el @host, se almacena %, es decir cualquier host. Ya que se puede usar los comodines % (cualquier grupo de caracteres) y _ (cualquier carácter). También podemos usar máscaras (8,16,32,64) y submáscaras de red.
- Si omitimos el IDENTIFIED BY password el usuario será creado sin password, por lo que podrá entrar el cliente sin el parámetro -p
- Si ponemos @localhost define que el usuario solamente se puede conectar desde el servidor de MySQL. Así debería ser root.
- El usuario recién creado tiene privilegio **USAGE**, es decir, sólo de conexión, por lo que deberemos asignarle permisos utilizando sentencias GRANT.

CREACIÓN Y ELIMINACIÓN DE USUARIOS

2- Con la sentencia GRANT

- Utilizando la sentencia GRANT podemos crear un usuario a la vez que otorgarle uno o varios privilegios sobre los objetos de una base de datos, o la base de datos completa.
- Al encontrarse una sentencia de tipo GRANT, el motor de MySQL revisa si el usuario existe previamente para el contexto que estamos asignándole permisos, y si dicho usuario no está presente en el sistema, lo crea.

GRANT permisos ON objeto TO usuario@host [IDENTIFIED BY password];

3- Insertando en la tabla users de la BD mysql

- Este es un método delicado y recomendado solo para los administradores de sistemas.

CREACIÓN Y ELIMINACIÓN DE USUARIOS

Para Borrar cuentas de usuario utilizamos el comando:

```
DROP USER usuario;
```

- En versiones anteriores a 5.0.2 sólo borra el usuario si ya se han eliminado sus permisos con REVOKE.
- DROP USER no cierra automáticamente ninguna sesión de usuario. En lugar de ello, en el evento que un usuario con una sesión abierta se elimina, el comando no tiene efecto hasta que se cierra la sesión de usuario.

CAMBIO DE CONTRASEÑA y CAMBIO DE NOMBRE

- Podemos cambiar la password de un usuario con el comando :

```
SET PASSWORD [FOR usuario] = PASSWORD ('texto');
```

- mysql> **set password for maite@localhost =password('maite');**
- Podemos cambiar el nombre de un usuario con RENAME USER:

```
RENAME USER nombre_antiguo TO nombre_nuevo;
```

Ejemplos

- Se puede obtener los usuarios, a través de la tabla user de la BD mysql:

```
mysql> use mysql;
Database changed
mysql> select user, host from user;
+-----+-----+
| user  | host      |
+-----+-----+
| pepe  | %         |
| root  | 127.0.0.1 |
| root  | ::1       |
|       | localhost |
| dam   | localhost |
| gestor1 | localhost |
| maite  | localhost |
| pepe  | localhost |
| pma    | localhost |
| prmysql | localhost |
| root  | localhost |
+-----+-----+
11 rows in set (0.00 sec)
```

Ejemplos

- Crea un usuario prueba con contraseña prueba
`mysql> create user prueba@localhost identified by 'prueba';`
- Crea un usuario prueba2 para tu ip con contraseña prueba e indica cómo se conectaría:
`create user prueba2@192.168.1.136 identified by 'prueba';`
`C:\Users\maite>mysql -u prueba2 -pprueba -h 192.168.1.136`
- Cambia el nombre del usuario prueba a prueba1
`rename user prueba@localhost to prueba1@localhost;`
- Cambia la contraseña de prueba1 a prueba1
`set password FOR prueba1@localhost=password('prueba1');`
- Describe las tablas user, hosts y db.

Ejemplos

- Muestra los campos user, host y password de user.
`select user, host, password from user;`
- Borra el usuario prueba2
`drop user prueba2@192.168.1.136;`
- Haz un show grants de prueba y otro de root.
`show grants for prueba1@localhost;`
- Conectate como prueba1 y comprueba el usuario conectado con Current_User();
`select current_user();`

PRIVILEGIOS EN MYSQL

- Un usuario puede obtener privilegios para manipular objetos de una BD con el comando GRANT. De igual forma se le pueden denegar permisos con REVOKE.
- La sintaxis del comando GRANT es:

```
GRANT tipo_privilegio [(columnas)] [, tipo_privilegio[(columnas)]] ...  
ON {nombre_tabla | * | *.* | base_datos.* | base_datos.tabla}  
TO usuario [IDENTIFIED BY [PASSWORD] 'password' ]  
    [, usuario [IDENTIFIED BY [PASSWORD] 'password' ]  
[WITH opcion [opcion ...  
Donde Opcion=  
{GRANT OPTION | MAX_QUERIES_PER_HOUR count | MAX_UPDATES_PER_HOUR  
count | MAX_CONNECTIONS_PER_HOUR count | MAX_USER_CONNECTIONS count }
```

PRIVILEGIOS EN MYSQL

- En mysql se puede otorgar a un usuario permisos para hacer cualquier operación a nivel de host, de BD, de tabla o de columna.
- **CONSULTAR PERMISOS**
- Para consultar los privilegios posibles usamos **SHOW PRIVILEGES**
- También podemos ver los permisos en las tablas correspondientes del diccionario de datos.

PRIVILEGIOS EN MYSQL

- **DAR PERMISOS:**

GRANT permisos ON objeto TO usuario [WITH GRANT OPTION];

- **PERMISOS:** Los privilegios listados a continuación, en caso de que actúen sobre columna como UPDATE se especifica esta entre paréntesis en el objeto.
- **OBJETO:** * Es el comodín de la mascara : base de datos.tabla Si queremos todas las tablas de una base de datos sería base.*, todas las bases de datos y todas sus tablas *.*
- **WITH GRANT OPTION:** Permite que este usuario de permisos a su vez sobre esas tablas.

```
mysql> grant create, select, insert, update on prueba.* to maite@localhost;  
Query OK, 0 rows affected (0.00 sec)
```

PRIVILEGIOS EN MYSQL

- Por ejemplo, para dar permisos de SELECT sobre las columnas nombreCliente, Direccion Telefono de la tabla clientes:
- **GRANT Select(nombreCliente, Direccion, Telefono) ON Clientes To usuario@localhost;**
- Con esta sentencia el usuario usuario@localhost sólo podrá seleccionar esas columnas de la tabla Clientes, no podrá hacer, por ejemplo: select * from Clientes;

PRIVILEGIOS EN MYSQL

Privilegio	Columna	Contexto
CREATE	<u>Create_priv</u>	Bases de datos, tablas o índices
DROP	<u>Drop_priv</u>	Bases de datos o tablas
GRANT OPTION	<u>Grant_priv</u>	Bases de datos, tablas, o procedimientos almacenados
REFERENCES	<u>References_priv</u>	Bases de datos o tablas
ALTER	<u>Alter_priv</u>	Tablas
DELETE	<u>Delete_priv</u>	Tablas
INDEX	<u>Index_priv</u>	Tablas
INSERT	<u>Insert_priv</u>	Tablas
SELECT	<u>Select_priv</u>	Tablas
UPDATE	<u>Update_priv</u>	Tablas
CREATE VIEW	<u>Create_view_priv</u>	Vistas
SHOW VIEW	<u>Show_view_priv</u>	<u>vistas</u>
ALTER ROUTINE	<u>Alter_routine_priv</u>	Procedimientos almacenados
CREATE ROUTINE	<u>Create_routine_priv</u>	Procedimientos almacenados
EXECUTE	<u>Execute_priv</u>	Procedimientos almacenados
FILE	<u>File_priv</u>	Acceso a archivos en la máquina del servidor
CREATE TEMPORARY TABLES	<u>Create_tmp_table_priv</u>	Administración del servidor

CONT. PRIVILEGIOS EN MYSQL

LOCK TABLES	Lock_tablas_priv	Administración del servidor
CREATE USER	Create_user_priv	Administración del servidor
PROCESS	Process_priv	Administración del servidor
RELOAD	Reload_priv	Administración del servidor
REPLICATION CLIENT	Repl_client_priv	Administración del servidor
REPLICATION SLAVE	Repl_slave_priv	Administración del servidor
SHOW DATABASES	Show_db_priv	Administración del servidor
SHUTDOWN	Shutdown_priv	Administración del servidor
SUPER	<u>Super_priv</u>	Administración del servidor

ALL [PRIVILEGES] => Da todos los privilegios simples excepto GRANT
OPTION

USAGE => Sinónimo de 'No Privilegios', permite únicamente la conexión al
gestor

PRIVILEGIOS EN MYSQL

- Si el usuario no existe, se crea, con la password indicada en IDENTIFIED BY.
- Adicionalmente se pueden indicar ciertas opciones precedidas de la cláusula WITH:
- **GRANT OPTION** => Permite dar a otros usuarios los permisos que tiene el usuario.
- **MAX_QUERIES_PER_HOUR *count*** => Permite restringir el número de consultas por hora que puede realizar un usuario
- **MAX_UPDATES_PER_HOUR *count*** => Permite restringir el número de modificaciones por hora que puede realizar un usuario
- **MAX_CONNECTIONS_PER_HOUR *count*** => Permite restringir el número de conexiones (logins) por hora que realiza un usuario
- **MAX_USER_CONNECTIONS *count*** => Permite limitar el número de conexiones simultáneas que puede tener un usuario
- **En cualquiera de las opciones, si a count se le da el valor 0, significa ilimitado.**

PRIVILEGIOS EN MYSQL

- **REVOCAR/QUITAR PERMISOS:**

REVOKE permisos ON objeto FROM usuario;

=> Quitar el permiso de select en la tabla jardineria.Cliente:

REVOKE Select ON jardineria.Cliente from usuario@localhost

⇒ Quitar el permiso ALL PRIVILEGES de todas las tablas de todas las BD:

REVOKE ALL PRIVILEGES ON *.* from usuario@localhost

=> Quitar los permisos de select e insert de todas las tablas de jardineria:

REVOKE Select,Insert ON jardineria.* from usuario@localhost

Bibliografía

- <http://www.monografias.com/trabajos16/transacciones/transacciones.shtml#ixzz3UOJGpHaF>
- http://www.mundoracle.com/funciones-sql.html?Pg=sql_plsql_3.htm
- <http://tallerdebasededatos.obolog.es/unidad-cuatro-control-trasacciones-444803>
- <http://mysql.conclase.net/>