UT8. SENTENCIAS DE MANIPULACIÓN DE DATOS EN SQL.

Módulo: BASES DE DATOS

Curso 2022/2023. 1° DAM

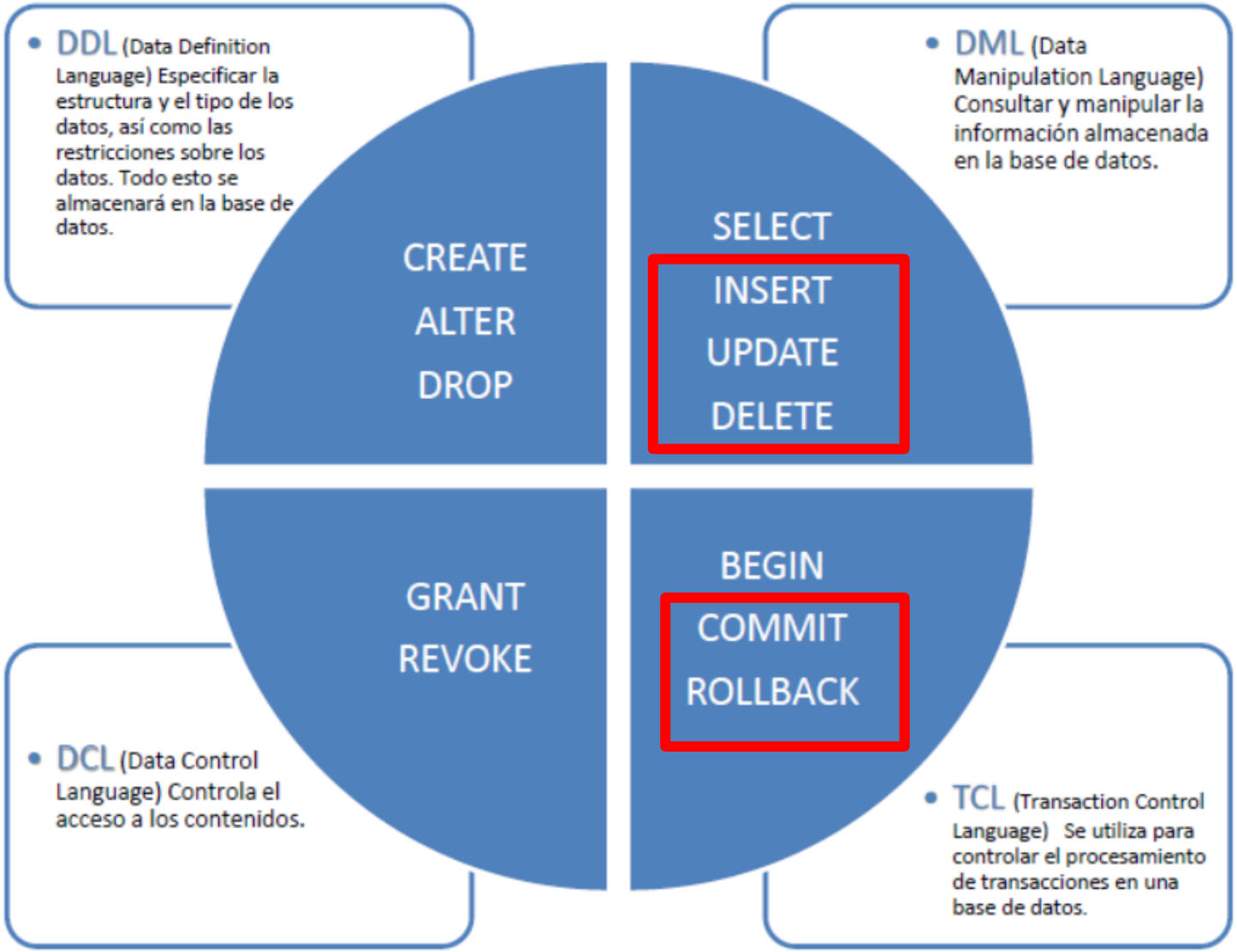
Ruth Lospitao Ruiz

CONTENIDOS

- Sentencias DML:
  - de inserción
  - eliminación
  - y actualización de registros.
- Integridad Referencial
- Transacciones. Estados temporales intermedios de la base de datos. Sentencias de procesamiento de transacciones.
- Acceso simultáneo a los datos: políticas de bloqueo. Niveles de bloqueo (fila, tabla).

SENTENCIAS DML
INSERT

LENGUAJE SQL



LENGUAJE SQL

	SENTENCIA	DESCRIPCIÓN
DML	*Manipulación de datos*	
	SELECT	Recupera datos de la base de datos.
	INSERT	Añade nuevas filas de datos a la base de datos.
	DELETE	Suprime filas de datos de la base de datos.
	UPDATE	Modifica datos existentes en la base de datos.
DDL	*Definición de datos*	
	CREATE TABLE	Añade una nueva tabla a la base de datos.
	DROP TABLE*	Suprime una tabla de la base de datos.
	ALTER TABLE*	Modifica la estructura de una tabla existente.
	CREATE VIEW*	Añade una nueva vista a la base de datos.
	DROP VIEW*	Suprime una vista de la base de datos.
	CREATE INDEX*	Construye un índice para una columna.
	DROP INDEX*	Suprime el índice para una columna.
	CREATE SYNONYM*	Define un alias para un nombre de tabla.
	DROP SYNONYM*	Suprime un alias para un nombre de tabla.
DCL	*Control de acceso*	
	GRANT	Concede privilegios de acceso a usuarios.
	REVOKE	Suprime privilegios de acceso a usuarios.
	Control de transacciones	
	COMMIT	Finaliza la transacción actual.
	ROLLBACK	Aborta la transacción actual.

EL LENGUAJE DML

- Las sentencias DML del lenguaje SQL son las siguientes:
  - SELECT: Se utiliza para extraer información de la base de datos, tanto de una tabla, como de varias.
  - INSERT: Inserta uno o varios registros en una tabla
  - DELETE: Borra registros de una tabla
  - UPDATE: Modifica registros en una tabla.

Estudiado en UT5 Y UT 6

UT8

- A todas las sentencias o comandos que se ejecutan en un SGBD se les denomina CONSULTAS o QUERIES (a veces, también se denominan consultas solo a las sentencias SELECT, por lo que da lugar a errores de interpretación).
- Todas las sentencias terminan en punto y coma (';').

TABLAS DE EJEMPLOS

EDITORIALES

Nombre_e	Direccion	Pais	Ciudad
Universal Books	Brown Sq. 23	EEUU	Los Ángeles
Rama	Canillas, 144	España	Madrid
Mc Graw-Hill	Basauri, 17	España	Madrid
Paraninfo	Virtudes, 7	España	Madrid

LIBROS

Codigo	Titulo	NumPaginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	48	Rama
493942	Turbo C++	125	Mc Graw-Hill
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	358	Rama

INSERT

- Se utiliza para insertar o añadir datos a una tabla.
- INSERT inserta filas enteras en la tabla indicada.

```
INSERT INTO mitabla [ (columna1 [ , columna2... ] ) ] VALUES (valor1  [ , valor2... ] );
```

- El número de valores debe corresponderse con el número de columnas.
- Además, el tipo de valor i-ésimo debe corresponderse con el tipo de datos de la columna i-ésima indicada. Por ejemplo, el valor2, debe ser del mismo tipo que la columna2.
- Ejemplo:

INSERT INTO libros (codigo, titulo, numPaginas, editorial) VALUES (34587, 'Int. Artificial', 50, 'Paraninfo');

Ojo, si ya existe el código 34587 la línea anterior nos daría un error; Duplicate entry '34587' for key 'libros.PRIMARY'

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo

INSERT

- Si al insertar una fila, para una columna no se hubiera indicado valor, se pondrá el valor por defecto para esa columna.
- Si al crear la tabla no se ha indicado la clausula DEFAULT para una columna, el valor por defecto es NULL.
- Ejemplos:

```
INSERT INTO libros (codigo, titulo, num_paginas, editorial) VALUES (34587, 'Int. Artificial', 50, 'Paraninfo');
INSERT INTO libros (codigo, titulo, num_paginas) VALUES (45307, 'Virus Informát.', 50);
INSERT INTO libros (codigo, titulo, editorial) VALUES (112313, 'Sist. Informac.', 'Rama');
```

LIBROS

Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	NULL	Rama

INSERT

- En un INSERT, las columnas no tienen porqué aparecer en el mismo orden que al crear la tabla:

INSERT INTO libros (codigo, titulo, num_paginas, editorial) VALUES (34587, 'Int. Artificial', 50, 'Paraninfo');

INSERT INTO libros (codigo, num_paginas, titulo) VALUES (45307, 50, 'Virus Informát.');

INSERT INTO libros (titulo, editorial, codigo) VALUES ('Sist. Informac.', 'Rama', 112313);

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	NULL	Rama

INSERT

- En un INSERT, los valores, pueden ser cualquier expresión válida que sea concordante en tipo con el campo:

INSERT INTO libros (codigo, titulo, num_paginas, editorial) VALUES (34587, 'Int. Artificial', 50, 'Paraninfo');

INSERT INTO libros (codigo, titulo, num_paginas) VALUES (**45000 + 307**, 'Virus Informát.', **90 - 40**);

INSERT INTO libros (codigo, titulo, editorial) VALUES (112313, **concat('Sist. ', 'Informac.')**, 'Rama');

LIBROS

Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	NULL	Rama

COLUMNAS AUTOINCREMENTALES

En MySQL y otros gestores (no es estándar), hay una forma de controlar que, en un campo numérico, los valores sean secuenciales en cada fila que se inserta. Suele utilizarse con la Primary Key de solo un atributo. Consiste en utilizar **auto_increment** en la definición:

```
create table libros (
        Codigo int primary key auto_increment,
        Titulo varchar(20),
        Num_paginas int,
        Editorial varchar(30));
```

En este caso, cuando se va a insertar una nueva fila, si se indica valor para la columna autoincremental, se utiliza ese valor. Si no se especifica, se toma el valor siguiente al mayor insertado.

COLUMNAS AUTOINCREMENTALES

INSERT INTO libros (titulo, num_paginas, editorial) VALUES ('Int. Artificial', 50, 'Paraninfo');

INSERT INTO libros (codigo, titulo, num_paginas) VALUES (125, 'Virus Informát.', 50);

INSERT INTO libros (codigo, titulo, editorial) VALUES (100, 'Sist. Informac.', 'Rama');

INSERT INTO libros (titulo, editorial) VALUES ('Turbo C++', 'Mc Graw-Hill');

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
1	Int. Artificial	50	Paraninfo
125	Virus Informát.	50	NULL
100	Sist. Informac.	NULL	Rama
126	Turbo C++	NULL	Mc Graw-Hill

INSERT: VALORES DEFAULT

```
create table libros (
        codigo int primary key auto_increment,
        Titulo varchar(20) not null,
        Num_paginas int DEFAULT 0,
        Editorial varchar(30)
);
```

- A una columna se le puede asignar el valor que tenga por defecto. Se puede hacer de dos formas:
  - No indicando la columna en el INSERT (y por tanto no dando valor).

    ```
    INSERT INTO libros (titulo, editorial) VALUES ('Int. Artificial', 'Paraninfo');
    ```

  - Indicando como valor la palabra DEFAULT.

    ```
    INSERT INTO libros (codigo, titulo, num_paginas) VALUES (125, 'Virus Informát.', DEFAULT);
    ```

INSERT

- Si una columna es NOT NULL, se le debe asignar un valor (o bien el valor por defecto), de forma que nunca quede con valor NULL. Si fuese a quedar con valor NULL, la instrucción fallaría, y el SGBD nos mostraría un error.
- Se puede hacer que en un SELECT se asignen valores a todas las columnas:
  - Indicando todas las columnas.

```
INSERT INTO libros (codigo, titulo, num_paginas, editorial) VALUES (34587, 'Int. Artificial', 50, 'Paraninfo');
```

  - No indicando la lista de columnas. En este caso, hay que indicar todos los valores para todas las columnas en el orden en que están definidas las columnas en la tabla.

```
INSERT INTO libros VALUES (45307, 'Virus Informát.', 50, NULL);
```

No se recomienda esta última opción, porque, el orden de las columnas puede cambiar, ya que en una base de datos relacional el orden no es significativo; o bien, porque si se añade una nueva columna, las sentencias INSERT dejan de funcionar, porque faltará un valor, cuando esta columna añadida puede tener un valor por defecto (o querer que sea nulo).

INSERT EXTENDIDO

- Algunos SGBD, como MySQL, permiten una sintaxis extendida para insertar más de una fila a la vez. Para ello, la lista de valores aparece más de una vez (cada vez con valores de una fila) separándose por comas.

```
INSERT INTO libros (titulo, num_paginas, editorial)
VALUES ('Int. Artificial', 50, 'Paraninfo'),
       ('Virus Informát.', 50, NULL),
       ('Sist. Informac.', NULL, 'Rama'),
       ('Turbo C++', NULL, 'Mc Graw-Hill');
```

LIBROS

Codigo	Titulo	Num_paginas	Editorial
1	Int. Artificial	50	Paraninfo
2	Virus Informát.	50	NULL
3	Sist. Informac.	NULL	Rama
4	Turbo C++	NULL	Mc Graw-Hill

INSERT EXTENDIDO

- Si cualquiera de las filas a insertar falla, no se crea ninguna fila.

```
INSERT INTO libros (titulo, num_paginas, editorial)
VALUES ('Int. Artificial', 50, 'Paraninfo'),
       ('Virus Informát.', 50, NULL),
       ('Sist. Informac.', NULL, 'Rama', 'Fila con un campo de más'),
       ('Turbo C++', NULL, 'Mc Graw-Hill');
```

LIBROS			
Codigo	Titulo	Num_paginas	Editorial

INSERT: RESTRICCIONES

- Además de la condición de que el número de columnas deba coincidir con el número de valores y de que el tipo de dato deba ser concordante, entre cada expresión y la columna correspondiente, pues si no, se produce un error al insertar, y no se inserta, se deben cumplir las condiciones del modelo (**restricciones**), o la sentencia de INSERT daría error. Entre ellas:
  - Restricción **NOT NULL**.
  - Restricción **PRIMARY KEY**.
  - Restricción **CHECK**.
  - Restricción **UNIQUE**.
  - Restricción **FOREIGN KEY**.

Para completar más información sobre la sentencia SQL INSERT consulta:

https://dev.mysql.com/doc/refman/5.7/en/insert.html

INSERT: INTEGRIDAD REFERENCIAL

- No se puede crear una fila que tenga valor en un campo que tenga una restricción Foreign Key, cuando el valor no aparece en la tabla a la que referencia.
- Por ejemplo, no se puede insertar un libro que tenga como editorial un valor que no exista en la tabla editoriales.
- Antes de realizar:

INSERT INTO libros (titulo, num_paginas, editorial) VALUES ('Int. Artificial', 50, 'Paraninfo');

- Hay que hacer:

INSERT INTO editoriales (Nombre_e, Direccion, País, Ciudad) VALUES ('Paraninfo', 'Virtudes, 7', 'España', 'Madrid');

INSERT DESDE SELECT

- Se utiliza para insertar o añadir datos a una tabla a partir de los resultados obtenidos con una SELECT. Es decir, los datos que se obtienen con la SELECT se insertan en la tabla correspondiente.
- Debe existir concordancia de tipos, igual que en cualquier INSERT. Es decir, cada columna resultado de la SELECT debe concordar en tipos con las columnas indicadas en el INSERT. Se deben cumplir también todas las restricciones.
- Se insertan tantas filas como filas devuelva la SELECT.

```
INSERT INTO mitabla [ (columna1, [ columna2,... ] ) ] SELECT …
```

- La sentencia SELECT puede ser tan compleja como se desee. Es decir, puede ser cualquier SELECT de los vistos en los temas anteriores.

INSERT DESDE SELECT

```
create table libros (
        Codigo int primary key auto_increment,
        Titulo varchar(20),
        Num_paginas int,
        Editorial varchar(30));
```

```
INSERT INTO libros (num_paginas, editorial) SELECT 10, nombre_e FROM editoriales;
```

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
1	NULL	10	Universal Books
2	NULL	10	Rama
3	NULL	10	Mc Graw-Hill
4	NULL	10	Paraninfo

SENTENCIAS DML DELETE

DELETE

- Borra registros de una tabla.

```
DELETE FROM nombre_tabla [ WHERE filtro ]
```

- Si no aparece la parte WHERE, borra todos los registros. Si aparece, solo borra los registros que cumplan la condición indicada en filtro.
- El filtro puede ser cualquiera de los indicados en los temas anteriores, incluida las subconsultas.

DELETE FROM libros; → Borra todos los libros.

DELETE FROM libros WHERE num_paginas > 50; → Borra los libros de más de 50 páginas.

Para completar más información sobre la sentencia SQL DELETE consulta:

https://dev.mysql.com/doc/refman/5.7/en/delete.html

DELETE

DELETE FROM libros WHERE num_paginas > 50;

LIBROS

Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	48	Rama
493942	Turbo C++	125	Mc Graw-Hill
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	358	Rama

LIBROS

Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	48	Rama
45307	Virus Informát.	50	NULL

DELETE

DELETE FROM libros WHERE editorial IN (SELECT nombre_e FROM editoriales WHERE país = 'España');

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	48	Rama
493942	Turbo C++	125	Mc Graw-Hill
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	358	Rama

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
45307	Virus Informát.	50	NULL

DELETE

- Hay que tener presente que pueden existir **restricciones de integridad referencial** (foreign key) que indicarán el comportamiento y permitirán borrar o no.
  - Si son de tipo **NO ACTION** (en MySQL no funciona) no permite borrar.
  - Si son de tipo **RESTRICT** no permite borrar y da un error.
  - Si son de tipo **CASCADE**, no solo borra en la tabla indicada, sino que intenta borrar en la tabla de la referencia. Si no puede borrar en esa otra tabla, da un error.
  - Si son de tipo **SET NULL**, aplica este cambio en la tabla referenciada.
- El borrado, o borra todas las filas indicadas o no borra ninguna. Es decir, es todo parte de la misma acción.

TRUNCADO DE TABLAS

- Se puede eliminar todo el contenido de una tabla, sin necesidad de borrar la tabla.
- Si hay una foreign key referenciando esta tabla (aunque tenga comportamiento cascade), da error.
- **Es una sentencia del Lenguaje de Definición de Datos** (DDL) por lo que **no se puede hacer ROLLBACK** (no se puede deshacer).
- Reduce el tamaño del fichero físico, reduce los índices,…

```
TRUNCATE TABLE [nombreBaseDatos.]nombreTabla;
```

No estándar

```
TRUNCATE TABLE empresa.personas;

TRUNCATE TABLE personas;
```

SENTENCIAS DML UPDATE

UPDATE

- Modifica el contenido de una tabla. Concretamente, modifica una o varias columnas de 0 a todas las filas.

```
UPDATE nombre_tabla SET columna1=exp1 [, columna2 = exp2 ... ] [ WHERE filtro ]
```

- Se actualiza la tabla indicada, modificando cada columna indicada en la parte SET (columna1, columna2,...) con el valor indicado en la expresión correspondiente (exp1, exp2,...).
- La parte WHERE es igual que en el borrado. Se puede utilizar todo lo visto en las consultas para la parte WHERE en los temas anteriores, incluida las subconsultas. Si no aparece la parte WHERE, se actualizan todas las filas. Si aparece, solo las que cumplan la condición del filtro.

Para completar más información sobre la sentencia SQL UPDATE consulta:

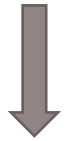
https://dev.mysql.com/doc/refman/5.7/en/update.html

UPDATE

UPDATE libros

SET Titulo = editorial;

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	48	Rama
493942	Turbo C++	125	Mc Graw-Hill
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	358	Rama

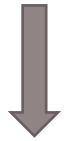


LIBROS			
Codigo	Titulo	Num_paginas	Editorial
34587	**Paraninfo**	50	Paraninfo
1022305	**Rama**	48	Rama
493942	**Mc Graw-Hill**	125	Mc Graw-Hill
45307	**NULL**	50	NULL
112313	**Rama**	358	Rama

UPDATE

```
UPDATE libros
SET num_paginas =
Num_paginas + 50
WHERE codigo > 100000;
```

LIBROS			
Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	48	Rama
493942	Turbo C++	125	Mc Graw-Hill
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	358	Rama



LIBROS			
Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	**98**	Rama
493942	Turbo C++	**175**	Mc Graw-Hill
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	**408**	Rama

UPDATE Y DELETE CON OTRAS TABLAS

- Se pueden realizar actualizaciones y borrados utilizando los valores de otras tablas.
- Pero no es estándar, y en cada SGBD se realiza de forma distinta.
- El siguiente ejemplo es válido en MySQL.

UPDATE

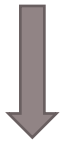
```
UPDATE libros lib
INNER JOIN editoriales e
ON lib.editorial = e.nombre_e
SET titulo = e.direccion
```

LIBROS

Codigo	Titulo	Num_paginas	Editorial
34587	Int. Artificial	50	Paraninfo
1022305	Concep. Y Dis.	48	Rama
493942	Turbo C++	125	Mc Graw-Hill
45307	Virus Informát.	50	NULL
112313	Sist. Informac.	358	Rama

EDITORIALES

Nombre_e	Direccion	Pais	Ciudad
Universal Books	Brown Sq. 23	EEUU	Los Ángeles
Rama	Canillas, 144	España	Madrid
Mc Graw-Hill	Basauri, 17	España	Madrid
Paraninfo	Virtudes, 7	España	Madrid



LIBROS

Codigo	Titulo	Num_paginas	Editorial
34587	**Virtudes, 7**	50	Paraninfo
1022305	**Canillas, 144**	48	Rama
493942	**Basauri, 17**	125	Mc Graw-Hill
45307	**NULL**	50	NULL
112313	**Canillas, 144**	358	Rama

INTEGRIDAD REFERENCIAL

SITUACIÓN

- En la base de datos de una plataforma de juegos online …
- ¿Qué ocurre si el registro correspondiente a los datos de un determinado juego es eliminado y existen partidas creadas de dicho juego?
- Para eso existe la integridad referencial, que además asegurará otras cosas como por ejemplo que no puedan existir partidas que reflejen que su creador es un usuario que no existe en la base de datos

CONCEPTO

- Dos tablas pueden ser relacionadas entre ellas si tienen en común uno o más campos, que reciben el nombre de clave ajena.
- La restricción de integridad referencial requiere que haya coincidencia en todos los valores que deben tener en común ambas tablas. Cada valor del campo que forma parte de la integridad referencial definida, debe corresponderse, en la otra tabla, con otro registro que contenga el mismo valor en el campo referenciado.
- Siguiendo con el ejemplo de juegos online, supongamos que en una determinada partida de un juego, se han unido una serie de usuarios. En la tabla de PARTIDAS existe un campo de referencia al tipo de juego al que corresponde, mediante su código de juego. Por tanto, no puede existir ninguna partida cuyo código de juego no se corresponda con ninguno de los juegos de la tabla JUEGOS.

RESTRICCIONES DE INTEGRIDAD Y ACTUALIZACIONES

Como ya hemos vistos los gestores de bases de datos relacionales permiten especificar ciertas restricciones que deban cumplir los datos contenidos en las tablas, como por ejemplo:

- **Restricción de clave primaria (PRIMARY KEY)** : columnas que identifican cada fila Restricción de clave ajena (FOREIGN KEY): columnas que hacen referencia a otras de la misma o de otras tablas
- **Restricción de comprobación de valores (CHECK):** conjunto de valores que puede o debe tomar una columna.
- **Restricción de no nulo (NOT NULL):** columnas que tienen que tener necesariamente un valor no nulo.
- **Restricción de unicidad (UNIQUE):** columnas que no pueden tener valores duplicados.

BORRADO Y MODIFICACIÓN DE DATOS CON INTEGRIDAD REFERENCIAL

`ON DELETE` y **`ON UPDATE`**: Nos permiten indicar el efecto que provoca el borrado o la actualización de los datos que están referenciados por claves ajenas. Las opciones que podemos especificar son las siguientes:

- **`RESTRICT`**: Impide que se puedan actualizar o eliminar las filas que tienen valores referenciados por claves ajenas. Es la opción por defecto en MySQL.
- **`CASCADE`**: Permite actualizar o eliminar las filas que tienen valores referenciados por claves ajenas.
- **`SET NULL`**: Asigna el valor NULL a las filas que tienen valores referenciados por claves ajenas.
- **`NO ACTION`**: Es una palabra clave del estándar SQL. En MySQL es equivalente a RESTRICT.
- **`SET DEFAULT`**: No es posible utilizar esta opción cuando trabajamos con el motor de almacenamiento InnoDB.

Puedes encontrar más información en la documentación oficial de MySQL.

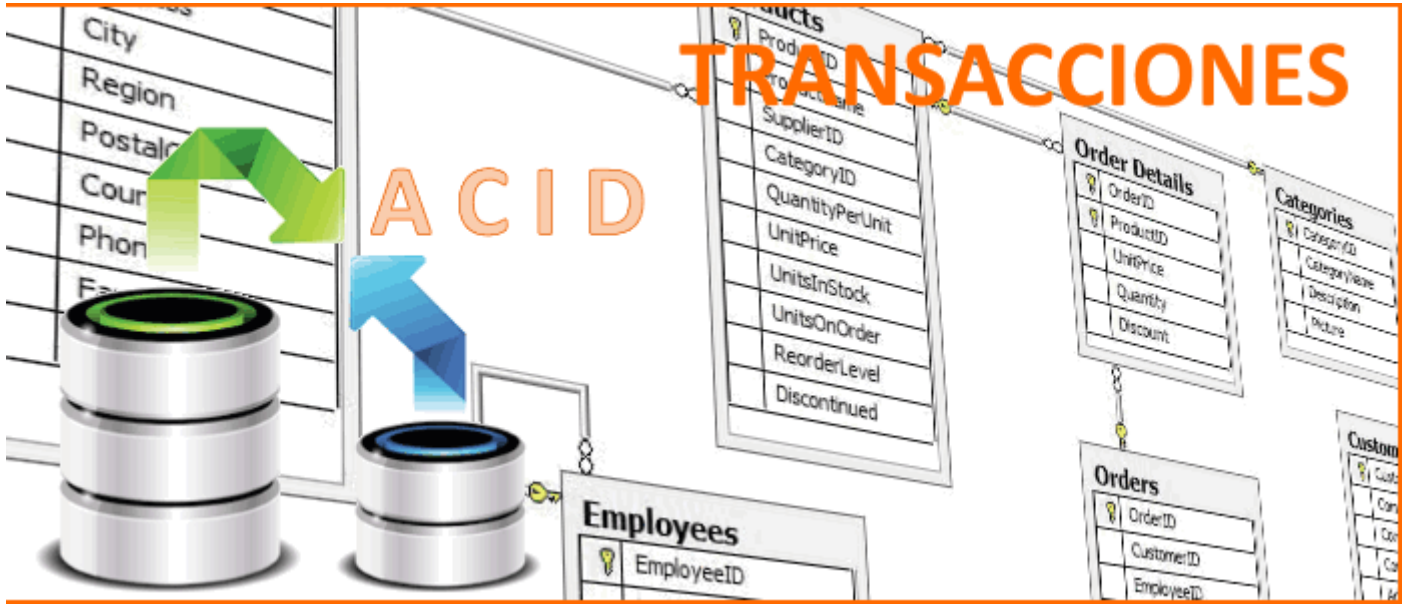
https://dev.mysql.com/doc/refman/5.7/en/create-table-foreign-keys.html (En el apartado Referencial Actions)

TRANSACCIONES

CONCEPTO

- Un SGDB actualiza múltiples datos a través de una transacción.

Una transacción es un conjunto de sentencias SQL que se tratan como una solo instrucción (atómica)



PROPIEDADES ACID

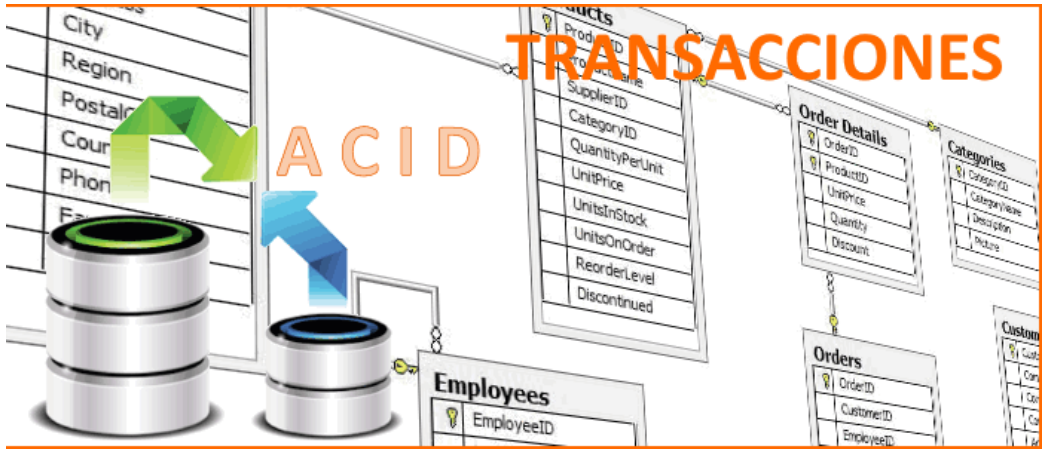
- Una transacción, para cumplir con su propósito debe presentar las siguientes características:

➢ Atomicidad (*Atomicity)*: las operaciones que componen una transacción deben considerarse como una sola.

➢ Consistencia (*Consistency)*: una operación nunca deberá dejar datos inconsistentes.

➢ Aislamiento (*Isolation*): evitar que los usuarios utilicen información que aún no está confirmada o validada.

➢ Durabilidad (*Durability)*: una vez completada la transacción los datos actualizados ya serán permanentes y confirmados.



TRANSACCIONES

- Una transacción puede ser:
  - Confirmada (con el uso de `commit`), si todas las operaciones individuales se ejecutaron correctamente
  - Abortada (con el uso de `rollback`) si hubo algún problema durante su ejecución.
    - Ejemplo: El producto pedido no está en stock y no se puede generar el pedido. Es decir, si se quiere enviar un pedido, habrá que anotar el pedido, rellenar la factura, y eliminar los productos del stock. Si primero se anota el pedido, luego se rellena la factura, y a la hora de eliminar los productos del stock, resulta que no hay stock suficiente, con una transacción se puede abortar la transacción, para no anotar el pedido ni haber generado la factura.

TRANSACCIONES

- Se considera transacción, a un conjunto de instrucciones o sentencias SQL que se tratan como una sola instrucción, y puede ser confirmada (commit) o abortada (rollback).
- De forma, que, o **se hacen todas las operaciones, o no se hace ninguna.** Pero tened presente que las sentencias DDL no son parte de las transacciones, y siempre se ejecutan sin necesidad de confirmarse.
- Las **transacciones son útiles para poder realizar varias operaciones y mantener la integridad de los datos**, de forma que o se ejecutan todas, o no se ejecutan ninguna.

TRANSACCIONES

- Cuando hacemos modificaciones en las tablas estas no se hacen efectivas (llevan a disco) hasta que no ejecutamos la sentencia COMMIT.
- **Cuando ejecutamos comandos DDL** (definición de datos) **se ejecuta un COMMIT automático**, o cuando cerramos la sesión.
- El comando ROLLBACK no permite deshacer estos cambios sin que lleguen a validarse. Cuando ejecutamos este comando se deshacen todos los cambios hasta el último COMMIT ejecutado.
- Hay dos formas de trabajar con AUTO_COMMIT (validación automática de los cambios:
  - Si está activado se hace COMMIT automáticamente cada sentencia y no es posible hacer ROLLBACK.
  - Si no lo está, tenemos la posibilidad de hacer ROLLBACK después de las sentencias DML (INSERT, UPDATE, DELETE) dejando sin validar los cambios. Es útil para hacer pruebas

TRANSACCIONES

En MySQL, se puede eliminar el autocommit de la sesión con:

SET AUTOCOMMIT = 0;

En MYSQL, para iniciar una transacción (y por tanto no hacer autocommit mientras dura la transacción), se hace:

START TRANSACTION;

ACCESO SIMULTÁNEO A LOS DATOS

INTRODUCCIÓN

- Cuando se realizan transacciones, puede haber **problemas en el acceso a los datos**, porque dos transacciones (en dos sesiones distintas) accedan y manipulen el mismo dato. Estos problemas son:
  - **Lectura sucia** (dirty read): Una transacción lee datos que en otra sesión ya han sido modificados, pero como no se ha realizado el commit, no ve estos cambios.
  - **Lectura no repetible** (Nonrepeateable read): Una transacción vuelve a leer datos que había leído, y se encuentra que han sido modificados por otra transacción.
  - **Lectura fantasma** (phantom read): Una transacción lee unos datos que no existían cuando se inició la transacción.
- Para solucionar estos problemas, el SGBD puede bloquear un conjunto de datos. Así, el Gestor bloquea o aísla los datos, y no permite a otras transacciones operar con ellos.

ACCESO SIMULTÁNEO A LOS DATOS

- Existen cuatro niveles de aislamiento, que definen como los cambios hechos por una transacción son visibles a otras transacciones (de otras sesiones).
  - **Lectura no persistida** (Read Uncommited): Permite que sucedan los tres problemas. Las sentencias SELECT son efectuadas sin realizar bloqueos, por tanto, todos los cambios hechos por una transacción pueden verlos las otras transacciones.
  - **Lectura persistida** (Read commited): Los datos leídos por una transacción pueden ser modificados por otras transacciones. Se pueden dar los problemas de Phantom Read y Nom Repeteable Read.
  - **Lectura repetible** (Repeateable Read); Ningún registro leído con un SELECT se puede cambiar en otra transacción. Tan solo se permite el problema del Phantom Read.
  - **Serializable:** Las transacciones ocurren de forma totalmente aislada a otras transacciones. Se bloquean las transacciones de tal manera que ocurren una detrás de otras, sin capacidad de concurrencia. El SGBD las ejecuta concurrentemente, y previene los interbloqueos.

UT8. SENTENCIAS DE MANIPULACIÓN DE DATOS EN SQL.

Módulo: BASES DE DATOS

Curso 2022/2023. 1° DAM

Ruth Lospitao Ruiz