

Ud 5 : Tratamiento/Edición de Datos

Insert/Update/Delete



IES Maestro de Cva.
Isabel Guerrero

Objetivos

- ❖ Identificar herramientas y sentencias para modificar el contenido de la base de datos
- ❖ Insertar, borrar y actualizar datos en las tablas
- ❖ Incluir en una tabla información de una consulta
- ❖ Adoptar medidas para mantener la integridad y consistencia de la información
- ❖ Reconocer el funcionamiento de transacciones y anular parcial o totalmente cambios producidos por una transacción
- ❖ Identificar efectos de las políticas de bloqueo de registros
- ❖ Crear vistas y usuarios y asignar privilegios

Pag. :2

Contenidos

- ❖ Sentencia INSERT
- ❖ INSERT y SELECT
- ❖ Sentencia UPDATE
- ❖ Sentencia DELETE
- ❖ UPDATE y DELETE con subconsultas
- ❖ Borrado y modificación de registros con relaciones
- ❖ Transacciones
- ❖ Acceso concurrente a los datos
- ❖ Vistas, usuarios y privilegios

Pag. :3

La sentencia INSERT

- ❖ La sentencia **INSERT** de SQL permite **insertar** una fila en una tabla, es decir, añadir un registro de información a una tabla.
- ❖ El formato de uso es muy sencillo:

```
INSERT [INTO] nombre_tabla [(nombre_columna, ... )]  
VALUES ({expr|DEFAULT}, ... )
```

- ❖ **nombre_tabla** es el nombre de la tabla donde se quiere insertar la fila.
- ❖ **nombre_columna:** (opcional) se pueden indicar las columnas donde se va a insertar la información.
- ❖ **VALUES:** valores que se van a insertar en la tabla.

Pag. :4

La sentencia INSERT

- ❖ Si se especifican las columnas, el orden de los valores (VALUES) se corresponderá cada uno con las columnas asociadas.
- ❖ Si no se especifican las columnas, la lista de valores se escribirá conforme al orden de las columnas en la definición de la tabla.
- ❖ Los valores que se dan a las columnas deben coincidir con el tipo de dato definido en la columna.
- ❖ Los valores constantes de tipo carácter y los de tipo fecha han de ir entre comillas simples (").
- ❖ Si el valor a insertar no cabe en la columna especificada da un error(ej: valor tiene 20 caract. y en la tabla sólo hay 10 car), tanto si es numérico como alfanumérico.

Pag. :5

La sentencia INSERT

- ❖ Crearemos la siguiente tabla:

```
CREATE TABLE MASCOTAS(
    CODIGO    INTEGER PRIMARY KEY,
    NOMBRE    VARCHAR(60),
    RAZA VARCHAR(60) DEFAULT 'CHUCHO',
    CLIENTE    VARCHAR(9)
);
```

Pag. :6

La sentencia INSERT

- ❖ INSERT especificando la lista de columnas:

```
INSERT INTO MASCOTAS (CODIGO, NOMBRE, RAZA)  
VALUES (1, 'PEQUITAS', 'GATO COMÚN EUROPEO');
```

- ❖ Este tipo de INSERT, hace corresponder a la columna CODIGO el valor 1, a la columna NOMBRE el valor 'PEQUITAS' y a la columna RAZA el valor 'GATO COMÚN EUROPEO'.
- ❖ La columna cliente, queda con un valor NULL, puesto que no se ha indicado un valor.

Pag. :7

La sentencia INSERT

- ❖ INSERT sin especificar la lista de columnas.

```
INSERT INTO MASCOTAS VALUES (2,'CALCETINES',  
'GATO COMÚN EUROPEO', '59932387L')
```

- ❖ En este caso, al no especificarse 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.

Pag. :8

La sentencia INSERT

- ❖ INSERT con columnas con valores por defecto

INSERT INTO MASCOTAS

VALUES (3,'TOTO',DEFAULT,'1111111K');

- ❖ Aquí, se ha usado el valor DEFAULT para asignar el valor por defecto a la tercera columna de la tabla MASCOTAS, es decir, la columna RAZA tiene definida la asignación por defecto del valor 'CHUCHO'.
- ❖ El valor por defecto si no se especifica en el CREATE es NULL.

Pag. :9

La sentencia INSERT

- ❖ Si se construye una sentencia INSERT con más campos en la lista de valores que el número de columnas especificadas (o número de columnas de la tabla) el SGBD informará del error.

INSERT INTO MASCOTAS(CODIGO,NOMBRE,RAZA)

VALUES (3,'TOTO');

- ❖ En Oracle:
ORA-00947-No hay suficientes valores
- ❖ En MySQL:
ERROR 1136 (21S01): Column count doesn't match value count at row 1

Pag. :10

La sentencia INSERT

- ❖ Si se construye una sentencia INSERT faltándole alguna columna que no pueda ser nula da un error:

```
INSERT INTO MASCOTAS(NOMBRE,RAZA)
VALUES ('TOTO','PASTOR ALEMÁN');
```

- ❖ En Oracle:

ORA-01400: No se puede realizar una inserción NULL en
('ISABEL"."MASCOTAS","CODIGO")

Pag. :11

La sentencia INSERT extendida

- ❖ La sintaxis extendida de **INSERT** para gestores tipo MySQL es la siguiente:

```
INSERT [INTO] nombre_tabla [(nombre_columna, ... )]
VALUES ({expr | DEFAULT}, ... ),( ... ), ...
```

- ❖ Los puntos suspensivos del final indican que se puede repetir varias veces la lista de valores , es decir insertar varios registros en un solo INSERT.

Pag. :12

La sentencia INSERT

- ❖ Si se construye una sentencia INSERT faltándole alguna columna que no pueda ser nula da un error:

```
INSERT INTO MASCOTAS(NOMBRE,RAZA)
VALUES ('TOTO','PASTOR ALEMÁN');
```

- ❖ En Oracle:

ORA-01400: No se puede realizar una inserción NULL en
("ISABEL"."MASCOTAS","CODIGO")

Pag. :13

INSERT y SELECT

- ❖ Una variante de la sentencia **INSERT** consiste en una utilizar la sentencia **SELECT** para obtener un conjunto de datos y, posteriormente, insertarlos en la tabla.

```
INSERT [INTO] nombre_tabla [(nombre_columna, ... )]
SELECT ... FROM ...
```

- ❖ Se puede ejecutar la siguiente consulta: inserta en una tabla BACKUP todos los vehículos

```
INSERT INTO COPIA_VEHIC1 SELECT * FROM VEHICULOS;
INSERT INTO COPIA_VEHIC2(MATRICULAC,DNIC)
SELECT MATRICULA,DNI FROM VEHICULOS
WHERE UPPER(POBLACION)='CIUDAD REAL';
```

- ❖ En el ejemplo anterior, la tabla BACKUP tiene una estructura idéntica a la tabla vehículos (nº de columna y tipos) y el nº de filas de la tabla BACKUP será el mismo que los de VEHICULOS.

Pag. :14

INSERT y SELECT

- ❖ Para insertar un empleado en la tabla EMPLE de apellidos 'GARCIA', la fecha de alta será la actual y nº empleado 1111, nos inventamos el resto de valores, el departamento será el mismo que el empleado cuyo código es el 7100.

```
INSERT INTO EMPLE
SELECT
  1111,'GARCIA','ANALISTA',7566,SYSDATE,2000,120,DEPT_NO
FROM EMPLE
WHERE EMP_NO=7100;
```

Pag. :15

INSERT y SELECT

❖ VERSIÓN DE ORACLE

- ❖ Para insertar un empleado en la tabla EMPLE de nº empleado 1111, apellidos 'GARCIA', oficio 'ANALISTA', su jefe es el empleado 7566, la fecha de alta será la actual, salario 2000, comisión: 120 y su código de departamento será el que tenga mayor nº de empleados.
- ❖ En primer lugar vamos a averiguar la SELECT que nos calcule el departamento con mayor nº de empleados

```
SELECT DEPT_NO FROM EMPLE
GROUP BY DEPT_NO
HAVING COUNT(*)=
  (SELECT MAX(COUNT(*)) FROM EMPLE
    GROUP BY DEPT_NO);
```

Pag. :16

INSERT y SELECT

- ❖ Hacemos la inserción en la tabla EMPLE teniendo en cuenta la SELECT anterior, los valores de las columnas que sabemos las colocamos en la select en el orden que lleven en el INSERT y las que no sabemos (DEPT_NO) se cogen de lo que devuelva la SELECT

INSERT INTO EMPLE

```
SELECT 1111,'GARCIA','ANALISTA',7566,SYSDATE,2000,120,DEPT_NO
FROM EMPLE
GROUP BY DEPT_NO
HAVING COUNT(*)=
(SELECT MAX(COUNT(*)) FROM EMPLE GROUP BY DEPT_NO);
```

Pag. :17

INSERT y SELECT

- ❖ Hacemos la inserción en la tabla EMPLE teniendo en cuenta la SELECT anterior:

INSERT INTO EMPLE

```
SELECT 1111,'GARCIA','ANALISTA',7566,NOW(),2000,120,DEPT_NO
FROM EMPLE
GROUP BY DEPT_NO
HAVING COUNT(*)=
(SELECT MAX(NRO_EMPLE) FROM
(SELECT COUNT(*) AS NRO_EMPLE
FROM EMPLE
GROUP BY DEPT_NO
) AS MAX_EMPLE
);
```

Pag. :18

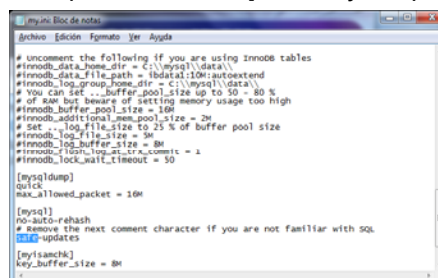
La sentencia UPDATE

- ❖ La sentencia **UPDATE** de SQL permite **modificar** el contenido de cualquier columna y de cualquier fila de una tabla.
- ❖ Su sintaxis es la siguiente:
UPDATE nombre_tabla
SET nombre_col1=expr1 [, nombre_col2=expr2] ...
[WHERE filtro]
- ❖ La actualizaci3n se realiza dando a las columnas nombre_col1, nombre_col2 ... los valores expr1, expr2, ...
- ❖ Se actualizan todas las filas seleccionadas por el filtro indicado mediante la cl1usula WHERE. Esta cl1usula WHERE, es id1ntica a la que se ha utilizado para el filtrado de registros en la sentencia SELECT.

Pag. :19

La sentencia UPDATE

- ❖ Si no nos permite utilizar **UPDATE** con condiciones **WHERE** hay que modificar el fichero de inicializaci3n de MySQL: **MY.INI** que est1 en el directorio de instalaci3n, por defecto ser1:
- ❖ Buscaremos la opci3n : **safe-updates** y le quitamos el comentario:



```
# uncomment the following if you are using InnoDB tables
#innodb_data_home_dir = C:\mysql\data\
#innodb_data_file_path = ibdata1:10M:autoextend
#innodb_log_group_home_dir = C:\mysql\data\
# You can set ..buffer_pool_size up to 50 - 80 %
# of RAM but beware of setting memory usage too high
#innodb_buffer_pool_size = 10M
#innodb_additional_mem_pool_size = 2M
# Set ..log_file_size to 25 % of buffer pool size
#innodb_log_file_size = 5M
#innodb_log_buffer_size = 8M
#innodb_flush_log_at_trx_commit = 1
#innodb_lock_wait_timeout = 50

[mysqldump]
quick
max_allowed_packet = 10M

[mysql]
no-auto-rehash
# Remove the next comment character if you are not familiar with SQL
#safe-updates
[mysandbox]
low_buffer_size = 8M
```

- ❖ Una vez realizada la modificaci3n en **Servicios**: reiniciamos MySQL.

Pag. :20

La sentencia UPDATE

- ❖ Por ejemplo, si se desea actualizar el equipo de 'Pau Gasol' porque ha fichado por otro equipo, por ejemplo, los 'Celtics', habría que ejecutar la siguiente sentencia:

UPDATE JUGADORES

SET NOMBRE_EQUIPO='CELTICS'

WHERE UPPER(NOMBRE)='PAU CASOL';

- ❖ Es posible cambiar más de una columna a la vez:

UPDATE JUGADORES

SET NOMBRE_EQUIPO='CELTICS', PESO=210

WHERE UPPER(NOMBRE)='PAU GASOL';

Pag. :21

La sentencia UPDATE

- ❖ Si se omite el filtro, el resultado es la modificación de todos los registros de la tabla.
- ❖ ¿Qué hubiera pasado si no ponemos la cláusula WHERE en la sentencia anterior?:

UPDATE JUGADORES

SET NOMBRE_EQUIPO= 'CELTICS ', PESO=210

- ❖ Que todos los registros de la tabla tendrían en las columnas anteriores el mismo valor.
- ❖ Otro ejemplo, para cambiar el peso de los jugadores de la NBA de libras a kilos:

UPDATE JUGADORES SET PESO=PESO*0.4535;

Pag. :22

UPDATE con SELECT

- ❖ Es posible actualizar o borrar registros de una tabla filtrando a través de una subconsulta.
- ❖ La única limitación es que hay gestores que no permiten realizar cambios en la tabla que se está leyendo a través de la subconsulta (MySQL).

Pag. :23

UPDATE con SELECT

- ❖ Se actualizan valores concretos en aquellas filas que cumplan una condición en la cual hay una subconsulta.

UPDATE n_tabla

SET columna1=valor1,..., columnan=valorn

WHERE (columna1,columna2,...) =

(SELECT columna1,columna2,...FROM .. WHERE...);

Pag. :24

UPDATE con SELECT

- ❖ Se quiere actualizar el salario y la comisión , aumentándolos en un 10%, sólo a los empleados del departamento VENTAS. .

UPDATE EMPLE

```
SET SALARIO=SALARIO*1.1, COMISION=COMISION*1.1
WHERE DEPT_NO IN
(SELECT DEPT_NO FROM DEPART
WHERE UPPER(DNOMBRE)='VENTAS');
```

Pag. :25

UPDATE con SELECT

- ❖ Los valores a asignar proceden de una consulta.

- ❖ En **Oracle**:

```
UPDATE n_tabla
SET (col1, col2...)=
(SELECT col1, col2... FROM ...WHERE)
WHERE condición;
```

Pag. :26

UPDATE con SELECT

❖ En **Oracle**.

```
UPDATE n_tabla
SET (col1, col2...)=
(SELECT col1, col2... FROM ...WHERE)
WHERE col= (SELECT ...FROM ... WHERE...);
```

Pag. :27

UPDATE con SELECT

❖ Ejemplo: se quiere asignar al empleado 7521 el salario y el jefe que tiene asignado el empleado 7900.

❖ En **Oracle**:

```
UPDATE EMPLE
SET (SALARIO,DIR) = (SELECT SALARIO,DIR FROM EMPLE
                     WHERE EMP_NO=7900)
WHERE EMP_NO=7521;
```

Pag. :28

UPDATE con SELECT

- ❖ **ORACLE:** Se quiere actualizar el salario y la comisión , aumentándolos en un 10%, sólo a los empleados que pertenezcan a departamentos de más de 4 empleados .

UPDATE EMPLE

```
SET SALARIO=SALARIO*1.1, COMISION=COMISION*1.1
WHERE DEPT_NO IN
(SELECT DEPT_NO FROM EMPLE
GROUP BY DEPT_NO
HAVING COUNT(*)>4);
```

Pag. :29

UPDATE con SELECT

- ❖ **ORACLE:** Para todos los empleados de la tabla EMPLE y del departamento de CONTABILIDAD, cambiaremos su salario al doble del salario de 'SANCHEZ' y su apellido a mayúsculas:

UPDATE EMPLE

```
SET SALARIO=( SELECT SALARIO*2 FROM EMPLE
               WHERE APELLIDO='SANCHEZ'),
APELLIDO=UPPER(APELLIDO)
WHERE DEPT_NO=
(SELECT DEPT_NO FROM DEPART
WHERE DNOMBRE='CONTABILIDAD');
```

Pag. :30

UPDATE con SELECT

- ❖ **ORACLE:** A continuación vamos a modificar en la tabla EMPLE, todos aquellos empleados que pertenezcan al departamento con mayor nº de empleados, cambiando su sueldo a la mitad y la comisión a cero.

UPDATE EMPLE

```
SET SALARIO=SALARIO/2, COMISION=0
WHERE DEPT_NO IN
(SELECT DEPT_NO FROM EMPLE
GROUP BY DEPT_NO
HAVING COUNT(*)=
(SELECT MAX(COUNT(*)) FROM EMPLE
GROUP BY DEPT_NO));
```

Pag. :31

La sentencia DELETE

- ❖ En SQL se utiliza la sentencia **DELETE** para **eliminar** filas de una tabla.
- ❖ Su sintaxis es:

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

- ❖ El comando DELETE borra los registros seleccionados por el filtro WHERE, que es idéntico al de la sentencia SELECT.

Pag. :32

La sentencia DELETE

- ❖ Si se desea borrar al jugador 'JORGE GARBAJOSA' de la base de datos, habría que escribir la siguiente sentencia:

```
DELETE FROM JUGADORES  
WHERE UPPER(NOMBRE)='JORGE GARBAJOSA';
```

- ❖ Si se omite el filtro, el resultado es el borrado de todos los registros de la tabla:

```
DELETE FROM JUGADORES;
```

Pag. :33

DELETE con SELECT

- ❖ Se borran registros cuyo filtro proviene de una subconsulta:

```
DELETE [FROM] Nombre_tabla  
[WHERE condición = (SELECT ...);
```

- ❖ Para eliminar los departamentos de DEPART con menos de 2 empleados:

```
DELETE FROM DEPART  
WHERE DEPT_NO IN  
(SELECT DEPT_NO FROM EMPLE  
GROUP BY DEPT_NO  
HAVING COUNT(*)<2)
```

Pag. :34

DELETE con SELECT

- ❖ Se eliminarán los empleados 'Representante Ventas' que no tengan clientes, se podría codificar:

```
DELETE FROM EMPLEADOS
WHERE PUESTO='REPRESENTANTE VENTAS' AND
CODIGOEMPLEADO NOT IN
(SELECT CODIGOEMPLEADOREPVENTAS
FROM CLIENTES);
```

Pag. :35

DELETE con SELECT

- ❖ Para borrar pedidos (clave primaria COD_PEDIDO y FECHA_PEDIDO) que no tengan registros en LINEAS_PEDIDO.

```
DELETE FROM PEDIDOS
WHERE (COD_PEDIDOS,FECHA_PEDIDO)
NOT IN
(SELECT DISTINCT COD_PEDIDOS,FECHA_PEDIDO
FROM LINEAS_PEDIDO);
```

Pag. :36

DELETE con SELECT

- ❖ **Oracle:** Para borrar los empleados cuyos salarios sean mayores que la media de todos los salarios.

```
DELETE FROM EMPL  
WHERE SALARIO >  
      (SELECT AVG(SALARIO)  
       FROM EMPL  
      );
```

Pag. :37

Borrado y modificación de registros con relaciones

- ❖ Hay que tener en cuenta que no siempre se pueden borrar o modificar datos: Considérese por ejemplo, que un cliente llama a una empresa pidiendo darse de baja como cliente, pero el cliente tiene algunos pagos pendientes, si el operador de la BBDD intenta eliminar el registro (**DELETE**), el SGBD debería informar de que **no** es posible eliminar ese registro puesto que hay **registros relacionados**.

Pag. :38

Borrado y modificación de registros con relaciones

- ❖ Si se desea cambiar (**UPDATE**) el NOMBRE de un equipo de la NBA (que es su clave primaria), ¿qué sucede con los jugadores? También habrá que cambiar el nombre del equipo de los jugadores, puesto que el campo NOMBRE_EQUIPO es una clave foránea.

Pag. :39

Borrado y modificación de registros con relaciones

- ❖ Vamos a recordar las cláusulas **REFERENCES** de la sentencia **CREATE TABLE** para crear las relaciones de clave foránea-clave primaria de alguna columna de una tabla:
- ❖ Definición_referencia:
**REFERENCES nombre_tabla [(nombre_columna , , ,.)]
[ON DELETE opción_referencia]**
- ❖ Opción_referencia:
CASCADE | SET NULL | NO ACTION

Pag. :40

Borrado y modificación de registros con relaciones

- ❖ Creamos las tablas con la opción NO ACTION (en MySQL), o no poniendo nada en Oracle:

MySQL:

```
CREATE TABLE CLIENTES (
  DNI          VARCHAR(15) PRIMARY KEY,
  NOMBRE       VARCHAR(50),
  DIRECCION    VARCHAR(50)
) ENGINE=INNODB;

CREATE TABLE PAGOS_PENDIENTES(
  DNI          VARCHAR(15),
  IMPORTE      DOUBLE,
  FOREIGN KEY(DNI) REFERENCES CLIENTES(DNI)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
) ENGINE=INNODB;
```

ORACLE:

```
CREATE TABLE CLIENTES (
  DNI          VARCHAR2(15) PRIMARY KEY,
  NOMBRE       VARCHAR2(50),
  DIRECCION    VARCHAR2(50)
);

CREATE TABLE PAGOS_PENDIENTES(
  DNI          VARCHAR2(15),
  IMPORTE      NUMBER(9),
  FOREIGN KEY(DNI) REFERENCES CLIENTES(DNI)
);
```

Pag. :41

Borrado y modificación de registros con relaciones

- ❖ Si se intenta borrar o actualizar un cliente que tiene registros en pagos pendientes no nos deja.

DELETE FROM CLIENTES WHERE DNI='5555672L';

UPDATE CLIENTES SET DNI='55555555L' WHERE DNI='5555672L';

- ❖ En **Oracle** nos da el error:

ORA-02292: restricción de integridad
(ISABEL.SYS_C0014174) violada - registro secundario encontrado

Pag. :42

Borrado y modificación de registros con relaciones

- ❖ Si se crea con la opción **ON UPDATE CASCADE** y **ON DELETE CASCADE**, si se modifican o se borran los clientes automáticamente se modifican o se borran los pagos pendientes.
- ❖ Si se crea con la opción: **ON UPDATE SET NULL** y **ON DELETE SET NULL**, los valores de las claves ajenas se actualizan a nulo.
- ❖ En **Oracle**, no existe la opción **ON UPDATE**.

Pag. :43

Transacciones

- ❖ Una **transacción** es un conjunto de sentencias SQL que se tratan como una sola instrucción (atómica).
- ❖ Una transacción puede ser confirmada (**commit**), si todas las operaciones individuales se ejecutaron correctamente, o, abortada (**rollback**) a la mitad de su ejecución si hubo algún problema (por ejemplo, el producto pedido no está en stock, por tanto no se puede generar el envío).
- ❖ Trabajar con transacciones puede ser esencial para mantener la integridad de los datos.
- ❖ Por ejemplo, se puede dar el caso de que se descuenta el stock de un producto antes de proceder a su envío, pero cuando se va a generar la cabecera del pedido, la aplicación cliente sufre un corte en las comunicaciones y no da tiempo a generarlo. Esto supone una pérdida de stock.
- ❖ La transacción garantiza la atomicidad de la operación: O se hacen todas las operaciones, o no se hace ninguna.

Pag. :44

Transacciones

- ❖ **AUTOCOMMIT=ON:** cada comando SQL que se ejecute, será considerado como una transacción independiente y no hace falta realizar **COMMIT**.
- ❖ **AUTOCOMMIT=OFF:** activa las transacciones de múltiples sentencias.
- ❖ Generalmente, en **MySQL** cuando se conecta un cliente, por defecto está activado el modo **AUTOCOMMIT=ON**.
- ❖ Cuando se activa **AUTOCOMMIT=OFF**, todos los comandos SQL enviados al SGBD tendrán que terminarse con una orden **COMMIT** o una orden **ROLLBACK**. De este modo, se asegura la integridad de los datos a un nivel más alto.
- ❖ Muchos SGBD requieren de una orden **START TRANSACTION** o **START WORK** para comenzar una transacción y otros lo hacen de forma implícita al establecer el modo **AUTOCOMMIT=OFF**.
- ❖ En **Oracle** la opción por defecto es **AUTOCOMMIT=OFF**.

Pag. :45

Transacciones

- ❖ **Oracle:** por defecto OFF

SET AUTOCOMMIT OFF/ON

Pag. :46

Acceso concurrente a los datos

- ❖ Cuando se utilizan transacciones, puede haber problemas de concurrencia en el acceso a los datos, es decir, problemas ocasionados por el acceso al mismo dato por dos transacciones distintas.
- ❖ Estos problemas están descritos por SQL estándar y son los siguientes:
 - **Dirty Read (Lectura Sucia).** Una transacción lee datos escritos por una transacción que no ha hecho COMMIT.
 - **Nonrepeatable Read (Lectura No Repetible).** Una transacción vuelve a leer datos que leyó previamente y encuentra que han sido modificados por otra transacción.
 - **Phantom Read (Lectura Fantasma).** Una transacción lee unos datos que no existían cuando se inició la transacción.

Pag. :47

Acceso concurrente a los datos

- ❖ El SGBD puede **bloquear** conjuntos de **datos** para evitar o permitir que sucedan estos problemas. Según el nivel de concurrencia que se desee, es posible solicitar al SGBD cuatro niveles de aislamiento.

Pag. :48

Acceso concurrente a los datos

- ❖ Un nivel de aislamiento define cómo los cambios hechos por una transacción son visibles a otras transacciones:
 - **Read Uncommitted (Lectura no confirmada).** Las instrucciones pueden **leer** filas que han sido modificadas por otras transacciones pero todavía **no** se han **confirmado**. Permite que sucedan los **tres** problemas (lectura sucia, lectura repetible, datos fantasmas).
 - **Read Committed (Lectura confirmada).** Las instrucciones **no** pueden **leer** datos que hayan sido **modificados**, pero **no confirmados**, por otras transacciones y Esto evita las lecturas de datos sucios. Otras transacciones pueden cambiar datos entre cada una de las instrucciones de la transacción actual, dando como resultado lecturas no repetibles o datos fantasma.
 - **Repeatable Read (Lectura Repetible).** Las instrucciones **no** pueden **leer** datos que han sido **modificados** pero aún no confirmados por otras transacciones y que ninguna otra transacción puede modificar los datos leídos por la transacción actual hasta que ésta finalice. 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 unas detrás de otras, sin capacidad de concurrencia. El SGBD las ejecuta concurrentemente si puede asegurar que no hay conflicto con el acceso a los datos.

Pag. :49

Acceso concurrente a los datos

- ❖ En **MySQL**, las tablas **innodb** tienen el nivel de aislamiento por defecto establecido en **REPEATABLE READ**, y se puede alterar cambiándolo en el fichero de configuración my.cnf (UNIX) / my.ini(Windows) o ejecutando:

```
SET TRANSACTION ISOLATION LEVEL
{READ UNCOMMITTED|READ COMMITTED|
REPEATABLE READ|SERIALIZABLE}
```

- ❖ En **Oracle**, el nivel por defecto es **READ COMMITTED** y, además de éste, solo permite **SERIALIZABLE**. Se puede cambiar ejecutando el comando:

```
SET TRANSACTION ISOLATION LEVEL
{READ COMMITTED|SERIALIZABLE};
```

Pag. :50