

Otros componentes de una BDD

*Máster en Business Intelligence e
Innovación Tecnológica*

Índice de contenidos

1. Otros componentes de una tabla (índices, claves, restricciones).
2. Componentes lógicos de datos (esquemas, dominios, vistas).
3. Componentes lógicos de control (procedimientos almacenados, disparadores y funciones).
4. Transaccionalidad y concurrencia
5. Prácticas: integración de los componentes.

- La principal función de una tabla es el almacenamiento de datos para su posterior explotación, por lo tanto su miembro más importante es la información.
- Como herramientas auxiliares para asegurar su óptimo desempeño en el manejo de la información aparecen otros elementos secundarios:
 - Índices
 - Claves
 - Restricciones
- Todos ellos son componentes del SGBD y están vinculados a la tabla, formando parte de su definición de parte explícita o implícita.

Índices

- Mediante estos elementos, el SGBD realiza una indexación de la tabla acelerando la respuesta en las búsquedas.
- Los índices generalmente se vinculan a un campo, sobre el que el SGBD indexará, por lo que generalmente se encontrarán índices sobre los campos con mayor frecuencia de búsquedas o sobre los identificadores, de cara a acelerar la realización de *joins*.
- Algunos SGBD permiten la creación de un índice sobre más de una columna.
- No todo es positivo: la indexación se almacena en disco duro y ésta se debe actualizar en cuanto se modifica el contenido de la tabla indexada, por lo que una actualización masiva puede acarrear un alto coste de CPU para su actualización.

Índices

Creación:

```
CREATE UNIQUE INDEX AUTHOR_INDEX ON tutorials_tbl (tutorial_author  
DESC);
```

Borrado:

```
DROP INDEX `PRIMARY` ON t;
```

También aplicables desde la sentencia *ALTER TABLE*.

Claves

Para asegurar un correcto uso de la indexación, búsquedas, etc., los SGBD introducen el concepto de clave. Éstas pueden ser de tres tipos:

- Clave primaria, *primary key (PK)*: columna o conjunto de columnas que, de entre todas las claves candidatas, identifica de forma única y inequívoca el registro. Puede ser un dato del dominio, suele utilizarse un campo autoincremental o UID autogenerado.
- Clave foránea, *foreign key (FK)*: registro de una tabla que enlaza con una clave foránea de otra table.
- Clave única, *unique key (UK)*: el resto de claves candidatas que no forman parte de la clave primaria, se definen como claves únicas como medida adicional de integridad.

Claves

```
SELECT * FROM dbo.Person
SELECT * FROM dbo.PassportDetails;
```

100 %

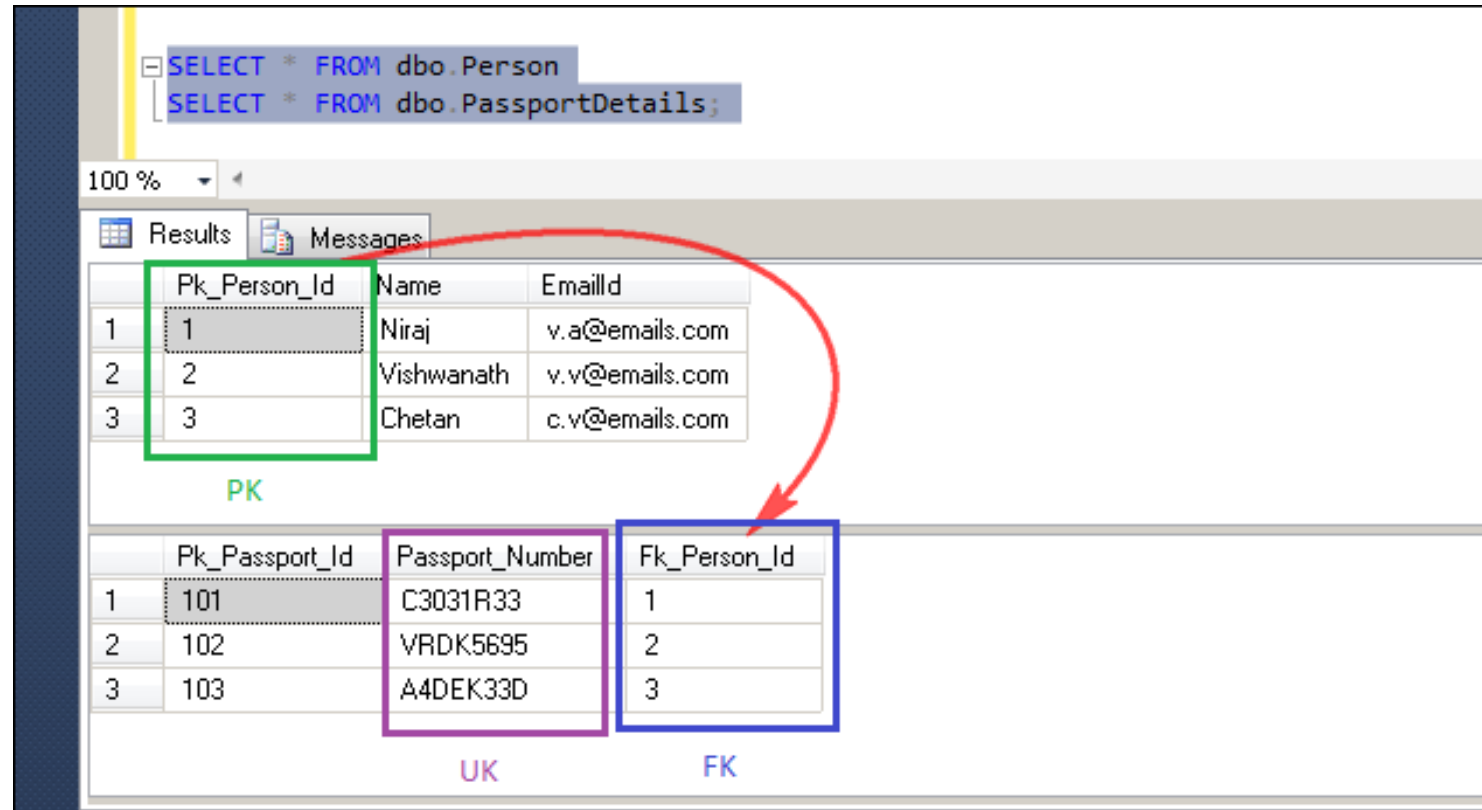
Results Messages

	Pk_Person_Id	Name	EmailId
1	1	Niraj	v.a@emails.com
2	2	Vishwanath	v.v@emails.com
3	3	Chetan	c.v@emails.com

PK

	Pk_Passport_Id	Passport_Number	Fk_Person_Id
1	101	C3031R33	1
2	102	VRDK5695	2
3	103	A4DEK33D	3

UK FK



Claves

Creación en la definición de la tabla (CREATE o ALTER TABLE):

```
CREATE TABLE products (  
  prd_id int(11) NOT NULL AUTO_INCREMENT,  
  prd_name varchar(355) NOT NULL,  
  prd_price decimal(10,0) DEFAULT NULL,  
  cat_id int(11) NOT NULL,  
  vdr_id int(11) NOT NULL,  
  PRIMARY KEY (prd_id),  
  
  CONSTRAINT products_ibfk_2  
  FOREIGN KEY (vdr_id)  
  REFERENCES vendors (vdr_id)  
  ON DELETE NO ACTION  
  ON UPDATE CASCADE  
)
```


Restricciones

- Las restricciones o *constraints* son elementos que restringen el dominio posible para un campo.
- Aunque elementos como FK o UK pueden actuar en parte como tal, no se deberían considerar como restricciones.
- Se tratan de cláusulas agregadas en la definición como complementos de la definición de los datos:
 - *NOT NULL*: define que un campo no puede ser nulo.
 - *DEFAULT*: establece un valor por defecto en el caso de venir nulo.
 - *CHECK*: añade una restricción avanzada mediante una sentencia.

Restricciones

Creación en la definición de la tabla (CREATE o ALTER TABLE):

```
CREATE TABLE Persons(  
  P_Id int NOT NULL,  
  LastName varchar(255) NOT NULL,  
  FirstName varchar(255),  
  Address varchar(255),  
  City varchar(255),  
  CONSTRAINT chk_Person CHECK (P_Id>0 AND City='Sandnes')  
)
```

- Más allá de los componentes propios de una tabla, los SGBD suelen ofrecer otros elementos de control y/o ayuda con un aspecto más general.
- Éstos dependen en gran medida del SGBD y para realizar su estructuración es muy recomendable estudiar las especificaciones del mismo.
- En este punto se verán tres de ellos:
 - Bases de datos
 - Esquemas
 - Dominios
 - Vistas

Bases de Datos

- No es común en muchos SGBD, o por lo menos se suelen utilizar valores por defecto, pero en algunos otros es necesario definir de forma independiente cada base de datos pudiendo albergar dentro del mismo motor varias de ellas.

Creación:

```
CREATE DATABASE [IF NOT EXISTS] db_name;
```

Borrado:

```
DROP DATABASE [IF EXISTS] db_name;
```

Esquemas

- Generalmente los SGBD estructuran su contenido en base a esquemas (*SCHEMA*). Éstos actúan como contenedores lógicos y físicos de las tablas.
- Conceptualmente se podría entender como un directorio que contiene varios ficheros, las tablas.
- Por defecto el esquema utilizado sería el del propio usuario de trabajo, comúnmente se utilizan como agrupadores de tablas: por aplicación, por módulo aplicativo, por características, etc.

Esquemas

Creación:

```
CREATE SCHEMA [IF NOT EXISTS] db_schema;
```

Borrado:

```
DROP SCHEMA [IF EXISTS] db_schema;
```

Dominios

- Basados en el concepto del modelo relacional, los dominios son tipos de datos definidos por el desarrollador. Especifican el tipo de datos nativo añadiendo una serie de restricciones que limitarían el conjunto de datos posibles.
- Posteriormente sería posible utilizarlos en la definición de la tabla.
- Simbolizaría una estructura u objeto en los lenguajes de programación, no es frecuente encontrarlos en las bases de datos existentes.
- En aplicaciones web (LAMP, ...), no confundir con el dominio de la aplicación.

Dominios

Creación:

```
CREATE DOMAIN domain_1 AS SMALLINT  
    DEFAULT 150  
    CONSTRAINT constraint_1  
        CHECK (VALUE IS NOT NULL)  
    CONSTRAINT constraint_2  
        CHECK (VALUE BETWEEN -1000 AND 9999) ;
```

Borrado:

```
DROP DOMAIN <Domain name> {RESTRICT | CASCADE}
```


Vistas

- Simbolizan tablas lógicas, se definen a partir de una cláusula select. Se accede a ellas del mismo modo, no permitiendo modificar el contenido de forma directa.
- Aportan seguridad, otorgando tablas consultivas no modificables.
- En consultas complejas, aquellas con muchos joins, elementos de rendimiento poco óptimo o sencillamente con un alto índice de uso, ofrecen un acceso más rápido ya que se encuentran precalculadas por el SGBD.
- Ante modificaciones masivas de las tablas de origen, suelen suponer un alto coste de CPU en su regeneración.

Vistas

Creación:

```
CREATE [OR REPLACE]  
VIEW view_name [(column_list)]  
AS select_statement;
```

Borrado:

```
DROP VIEW [IF EXISTS] view_name [, view_name] ... [RESTRICT | CASCADE]
```

- Adicionalmente a los elementos directamente relacionados con el almacenamiento o presentación de los datos, los SGBD suelen ofrecer otros elementos que actúan como herramientas para mantener los datos y gestionar la explotación de los mismos.
- Huyendo de lenguajes declarativos y con un aspecto procedural, existen varios componentes que pueden definirse en este término:
 - Procedimientos almacenados
 - Disparadores (triggers)
 - Funciones

Procedimientos almacenados

- Pequeños programas desarrollados dentro del motor de la base de datos que realizan una o varias tareas sobre el contenido de la misma.
- Vendrían a representar operaciones que se realizan en aplicaciones, realizándolas desde el propio motor de la SGBD. Esto supone:
 - Gestión única de la transaccionalidad desde la propia BDD.
 - Mejor rendimiento: menor información entre los servidores, resultsets acotados, etc.
 - Al estar dentro de la misma BDD, el código necesario es mucho más sencillo.
- Almacenados en las bibliotecas del SGBD (schemas), se pueden acceder a ellos desde la misma BDD o desde aplicaciones externas.
- Generalmente desarrollados en PL/SQL.

Procedimientos almacenados

```
CREATE PROCEDURE ordertotal(  
    IN onumber INT,  
    IN taxable BOOLEAN,  
    OUT ototal DECIMAL(8,2)  
) COMMENT 'Obtain order total, optionally adding tax'  
BEGIN  
    -- Declare variable for total  
    DECLARE total DECIMAL(8,2);  
    -- Declare tax percentage  
    DECLARE taxrate INT DEFAULT 6;  
    -- Get the order total  
    SELECT Sum(item_price*quantity)  
    FROM orderitems  
    WHERE order_num = onumber  
    INTO total;  
    -- Is this taxable?  
    IF taxable THEN  
        -- Yes, so add taxrate to the total  
        SELECT total+(total/100*taxrate) INTO total;  
    END IF;  
    -- And finally, save to out variable  
    SELECT total INTO ototal;  
END;
```

```
CALL ordertotal(20005, 0, @total);  
SELECT @total;
```

Disparadores (triggers)

- Similares en cuanto a desarrollo a los *stored procedures*, se ejecutan al suceder un evento concreto sobre una tabla o registro: un insert, un update o un delete.
- Tienen multitud de usos, pero es frecuente encontrarlos como complemento aplicativo de la acción que lo lanza: replicar la información en otras tablas, actualizar campos de resumen, etc.

Disparadores (triggers)

Ejemplo:

```
CREATE TRIGGER before_employee_update
  BEFORE UPDATE ON employees
  FOR EACH ROW
BEGIN
  INSERT INTO employees_audit
  SET action = 'update',
    employeeNumber = OLD.employeeNumber,
    lastname = OLD.lastname,
    changedat = NOW();
END$$
DELIMITER ;
```

Borrado:

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

Funciones

- Las funciones son, igual que en cualquier lenguaje, pequeños módulos que encapsulan una tarea algo más compleja.
- Todos los SGBD incorporan un conjunto de funciones base para las tareas más comunes: obtener una fecha, convertir una cadena a mayúsculas, sustituir un carácter, etc.
- Es posible definir funciones propias para utilizarlas, como las estándar, en cualquier parte: triggers, procedures, otras funciones o selects.
- A diferencia de los *stored procedures*, las funciones devuelven un valor mediante una cláusula *return*, no por parámetros de entrada/salida.

Funciones

```
DELIMITER $$
```

```
CREATE FUNCTION CustomerLevel(p_creditLimit  
double) RETURNS VARCHAR(10)
```

```
    DETERMINISTIC
```

```
BEGIN
```

```
    DECLARE lvl varchar(10);
```

```
    IF p_creditLimit > 50000 THEN
```

```
    SET lvl = 'PLATINUM';
```

```
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >=  
10000) THEN
```

```
        SET lvl = 'GOLD';
```

```
    ELSEIF p_creditLimit < 10000 THEN
```

```
        SET lvl = 'SILVER';
```

```
    END IF;
```

```
    RETURN (lvl);
```

```
END
```

```
SELECT CustomerName,  
CustomerLevel(c.creditLimit)  
FROM customers c
```

Conceptos

TRANSACCIÓN

- Una transacción es un conjunto de una o más operaciones que deben ejecutarse de forma indivisible y atómica.
- Sobre esta definición, pueden considerarse como una única unidad de trabajo.
- Un SGBD transaccional debe asegurar la ejecución de la transacción de forma integral.

CONCURRENCIA

- Los sistemas transaccionales productivos suelen encontrarse en situaciones de alta concurrencia. Esto significa múltiples accesos a los mismos datos, de forma que algunos pueden verse modificados durante las sucesivas consultas.
- El SGBD debe gestionar la presentación de la información correcta y las consecuencias que deriven de la actualización de la información on-line

Conceptos

Una transacción siempre debe conservar las propiedades ACID (del inglés, Atomicity, **C**onsistency, **I**solation and **D**urability).

Atomicity

- Asegurar que la transacción se realice o no, sin quedar a medias ante fallos

Consistency

- Asegurar el estado de validez de los datos en todo momento

Isolation

- Asegurar independencia entre transacciones

Durability

- Asegurar la persistencia de la transacción ante cualquier fallo

Ejemplo ACID

- Sistema Bancario simplificado
- Constituido por varias cuentas y un conjunto de transacciones que acceden y actualizan dichas cuentas.
- Base de datos residente en disco pero con una porción de la misma en memoria principal.
- Acceso a través de dos operaciones: leer(X) y escribir(x) (transfiere el dato X desde la Base de Datos/memoria intermedia local de la transacción a una memoria intermedia local de la transacción/Base de Datos)

```
Leer(A);  
A := A - 50;  
escribir(A);  
leer(B);  
B := B + 50;  
escribir(B);
```

T1

Ejemplo ACID

- Antes de ejecutar T_i : El valor de A es 1000€ y el valor de B es 2000€
- Durante la ejecución de T_i : Se produce un fallo que impide que la transacción finalice con éxito (fallos de alimentación, fallos de hardware, errores software, etc.)
- Valores reflejados en la Base de Datos: El valor de A es 950€ y el valor de B es 2000€
El estado del sistema deja de reflejar el estado real del mundo que se supone que modela \mathcal{AE} estado inconsistente

La propiedad de atomicidad de la transacción implica que debe ejecutarse integralmente.

```
Leer(A);  
A := A - 50;  
escribir(A);  
leer(B);  
B := B + 50;  
escribir(B);
```

T1

Ejemplo ACID

- Antes de ejecutar T_i : El valor de A es 1000€ y el valor de B es 2000€
- Durante la ejecución de T_i : Se produce un fallo que impide que la transacción finalice con éxito (fallos de alimentación, fallos de hardware, errores software, etc.)
- Valores reflejados en la Base de Datos: El valor de A es 950€ y el valor de B es 2000€
El estado del sistema deja de reflejar el estado real del mundo que se supone que modela \mathcal{AE} estado inconsistente

La propiedad de atomicidad de la transacción implica que debe ejecutarse integralmente.

El requisito de consistencia es que la suma de A y B no se vea alterada al ejecutar la transacción. Si una base de datos es consistente antes de ejecutar una transacción, tiene que seguir siéndolo después de ejecutar dicha transacción

```
Leer(A);  
A := A - 50;  
escribir(A);  
leer(B);  
B := B + 50;  
escribir(B);
```

T1

Ejemplo ACID

- Aislamiento: Incluso tras asegurar las propiedades de atomicidad y consistencia para cada transacción pueden ocurrir problemas si varias transacciones concurrentes entrelazaran sus operaciones de modo no deseado (produciendo un estado inconsistente).

```
Leer(A);  
A := A - 50;  
escribir(A);  
leer(B);  
B := B + 50;  
escribir(B);
```

T1

```
Leer(A);  
A := A - 50;  
escribir(A);  
leer(B);  
B := B + 50;  
escribir(B);
```

T2

La Base de Datos puede quedar en un estado inconsistente aunque las dos transacciones terminen

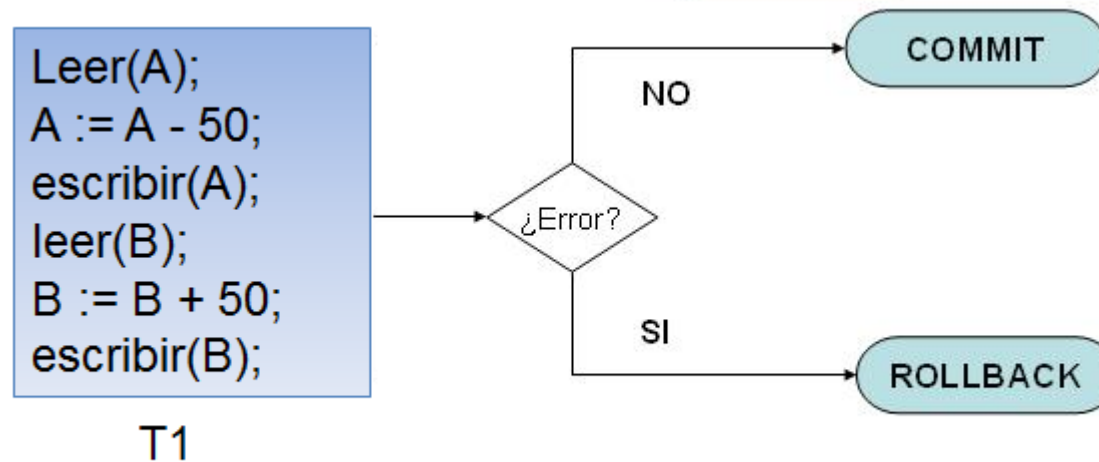
Ejemplo ACID

- Durabilidad: Una vez que se completa con éxito la ejecución de una transacción no puede suceder que un fallo del sistema produzca la pérdida de datos.
- Condiciones:
 - Las modificaciones realizadas por la transacción se guardan en disco antes de finalizar la transacción.
 - La información guardada en disco de las modificaciones realizadas por transacción es suficiente para reconstruir dichas modificaciones
- La responsabilidad de asegurar la durabilidad es del sistema de base de datos (en concreto del componente de gestión de recuperaciones)

Compromiso

Para garantizar las propiedades ACID de una transacción se utilizan las sentencias commit y rollback:

- **Commit:** compromete los cambios efectuados en la transacción en el SGBD.
- **Rollback:** de producirse un error, descarta los cambios de la transacción volviendo al estado inicial.

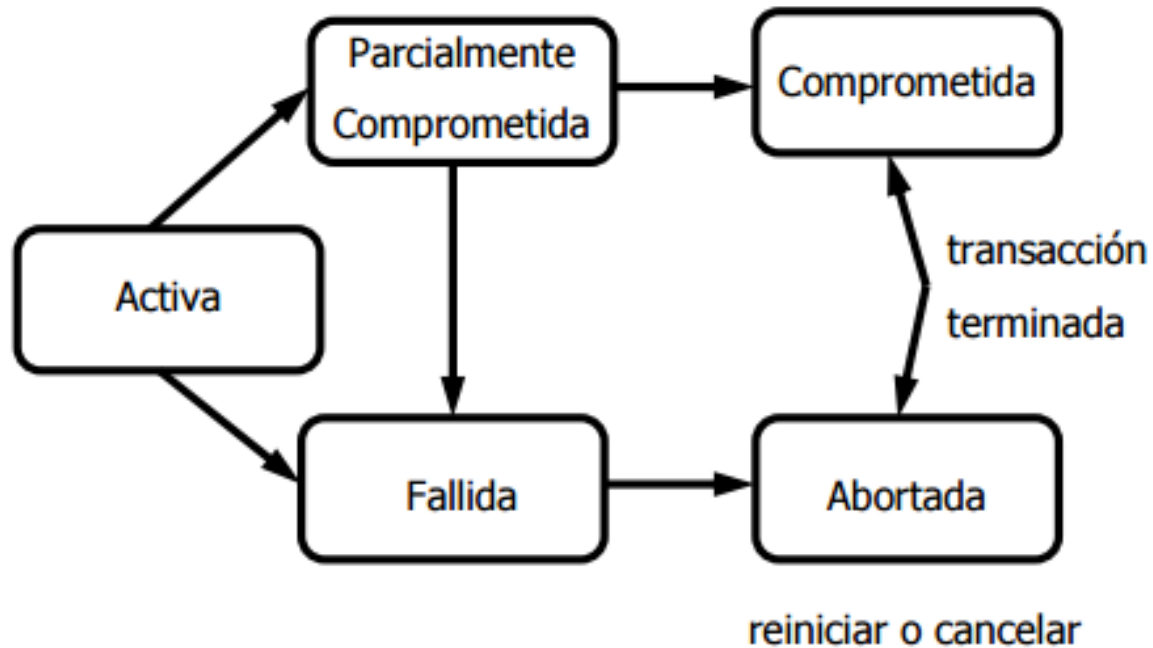


Estados de una transacción

Una transacción debe estar en uno de los siguientes estados:

- Activa (estado inicial): la transacción permanece en este estado durante su ejecución
- Parcialmente Comprometida: la transacción pasa a este estado cuando acaba de realizar la última instrucción
- Fallida: la transacción pasa a este estado tras descubrir que no puede continuar la ejecución normal
- Abortada: la transacción pasa a este estado después de haber restablecido la base de datos a su estado anterior
- Comprometida: la transacción pasa a este estado tras completarse con éxito

Estados de una transacción



Posibles interferencias

- Lecturas sucias: Es una transacción Tr-2 que lee la información que no ha sido confirmada por Tr-1 y que después es abortada, en este caso Tr-2 manipula información falsa que después es almacenada.

Transacción 1	Transacción 2
begin	
SELECT X /* X = 0 */	
X = X + 10 /* X = 10 */	
UPDATE X /* X = 10 en BD pero no comprometido */	begin
	SELECT X /* X = 10 */
rollback /* X = 0 en BD */	X = X + 10 /* X = 20 */
	UPDATE X /* X = 20 en BD pero no comprometido */
	commit /* X = 20 en BD y comprometido */

Posibles interferencias

- Lecturas no repetibles: Consiste en que la lectura de un registro por Tr-1 en T1, en T2 Tr-2 manipula el registro y lo actualiza, haciendo que Tr-1 en T3 lea un registro con un valor distinto en T1.

Transacción 1	Transacción 2
begin	
SELECT X /* X = 0 */	begin
	X = 20
	UPDATE X /* X = 20 en BD pero no comprometido */
	commit /* X = 20 en BD y comprometido */
SELECT X /* X = 20 */	
...	
commit	

Posibles interferencias

- Lecturas fantasma: Al igual que una lectura no repetible, en este caso el resultado es un conjunto de filas en T3 distinto que en T1.

Transacción 1	Transacción 2
begin	
SELECT WHERE condition	begin
	INSERT /* Inserta nuevas filas en BD (no comprometidas), algunas cumpliendo "condition" */
	commit /* Inserciones comprometidas */
SELECT WHERE condition /* Ahora devuelve más filas */	
...	
commit	

Niveles de aislamiento

- **Serializable:**

Este es el nivel de aislamiento más alto. Especifica que todas las transacciones ocurran de modo aislado, o dicho de otro modo, como si todas las transacciones se ejecutaran de modo serie (una tras otra). La sensación de ejecución simultánea de dos o más transacciones que perciben los usuarios sería una ilusión producida por el SGBD.

Si el SGBDR hace una implementación basada en bloqueos, la serialización requiere que los bloques de lectura y escritura se liberen al final de la transacción. Del mismo modo deben realizarse bloqueos de rango -sobre los datos seleccionados con SELECT usando WHERE- para evitar el efecto de las lecturas fantasma.

Cuando se hace una implementación no basada en bloqueos, si el SGBDR detecta una colisión de escritura entre transacciones sólo a una de ellas se le autoriza cometer.

Niveles de aislamiento

- **Lecturas repetibles (Repeatable reads):**

En este nivel de aislamiento, un SGBDR que implemente el control de concurrencia basado en bloqueos mantiene los bloqueos de lectura y escritura -de los datos seleccionados- hasta el final de la transacción.

Sin embargo, no se gestionan los bloqueos de rango, por lo que las lecturas fantasma pueden ocurrir.

- **Lecturas comprometidas (Read committed, lectura registrada)**

En este nivel de aislamiento, un SGBDR que implemente el control de concurrencia basado en bloqueos mantiene los bloqueos de escritura -de los datos seleccionados- hasta el final de la transacción, mientras que los bloqueos de lectura se cancelan tan pronto como acaba la operación de SELECT (por lo que el efecto de las lecturas no repetibles puede ocurrir, como se explica más abajo).

Al igual ocurría en el nivel anterior, no se gestionan los bloqueos de rango.

Niveles de aislamiento

- **Lecturas no comprometidas (Read uncommitted, lectura no registrada)**

Este es el menor nivel de aislamiento. En él se permiten las lecturas sucias (ver más abajo), por lo que una transacción puede ver cambios no cometidos aún por otra transacción.

El nivel de aislamiento por defecto de distintos SGBD varía ampliamente. La mayoría de bases de datos que gestionan transacciones permiten al usuario establecer cualquier nivel de aislamiento.

Algunos SGBD requieren sintaxis especial cuando se realiza una operación SELECT que efectúa bloqueos (e.g. SELECT ... FOR UPDATE para bloquear para escritura aquellas filas accedidas).

Relación entre nivel de aislamiento e interferencia

Nivel de Aislamiento	Situación de Interferencia		
	Lectura sucia	Lectura no repetible	Fantasma
Lectura no registrada	Posible	Posible	Posible
Lectura registrada	No posible	Posible	Posible
Lectura repetible	No posible	No posible	Posible
Serializable	No posible	No posible	No posible