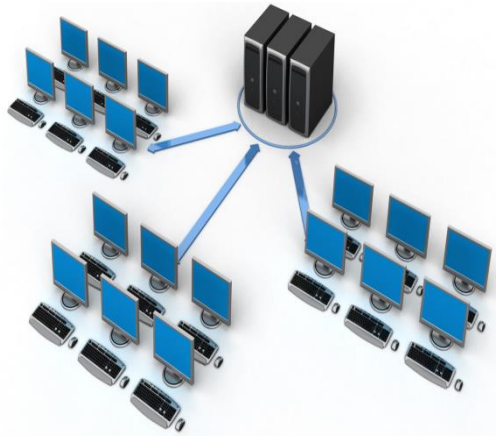
# Tema 5
## Transacciones y Concurrencia en Bases de Datos

Parte 1/2

Grado en Ingeniería Informática



Bases de Datos

2020/21

Departamento de Tecnologías de la Información
Universidad de Huelva

# Transacciones y Concurrencia en Bases de datos

## Objetivos

- Conocer el concepto de transacción
- Saber definir las propiedades de las transacciones
- Analizar el problema del control de concurrencia
- Distinguir las principales técnicas de Control de Concurrencia en Bases de Datos

## Índice

# Transacciones y Concurrencia en Bases de datos

# Transacciones y Concurrencia en Bases de datos

## Duración

- 5 Clases

## Bibliografía

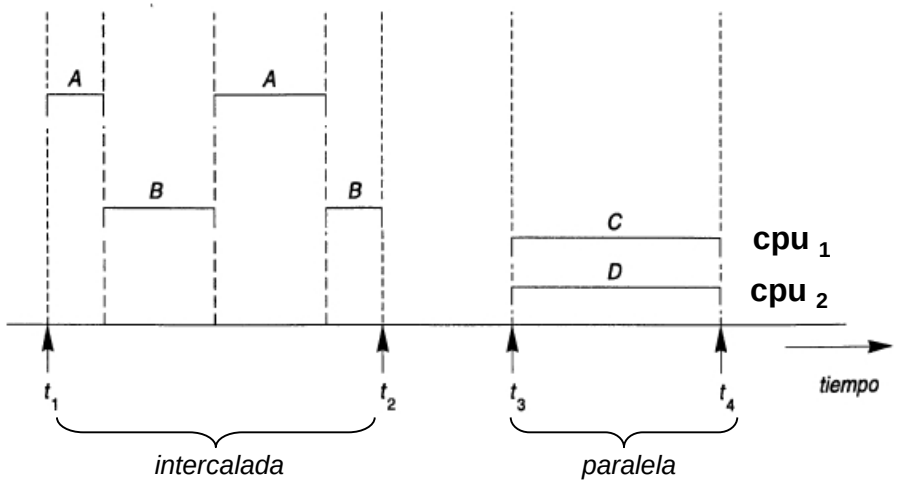
- Capítulos 19 y 20 de [Elmasri 02]
- Capítulo 20 de [Connolly 05]

# 1. Introducción

## Tipos de sistemas:

- Monousuario
  - Sólo un usuario utiliza el sistema

- Multiusuario
  - Varios usuarios acceden a la vez al sistema
  - Ejemplos: Sistemas bancarios, sistemas para la reserva de vuelos, etc.
  - Se caracterizan porque se ejecutan varios programas a la vez
  - Como el acceso a disco es bastante frecuente, y relativamente lento, hay que mantener la CPU ocupada en distintos programas de ejecución concurrente
  - Si hay varias CPU's estos programas pueden procesarse en paralelo
  - Si sólo hay una CPU, en realidad, sólo se puede procesar un programa cada vez, sin embargo, se produce la sensación de procesado en paralelo porque la ejecución de los programas se va alternando o intercalando en la CPU




Ejecución intercalada **vs** ejecución paralela

# 1.1 Conceptos: transacción y concurrencia

- Un programa de usuario puede llevar a cabo varias operaciones, pero a nivel del SGBD, sólo nos interesan las de lectura y escritura.
- Una **transacción** es una serie de acciones llevadas a cabo por un único usuario o por un programa de usuario, que lee y actualiza el contenido de la BD.
- Una transacción es una abstracción de un programa de usuario, en términos de lecturas y escrituras.
- **Ejemplo**: Operación ☞ transferencia de fondos de una cuenta corriente (CC) a una cuenta de ahorro (CA).
  - Parece que la operación está formada por una sola operación "*transferir un saldo X de CC a CA* ".
  - En realidad está formada por varias operaciones. Entre otras, se hace:
    - Restar X de la CC
    - Sumar X a la CA
  - Si en una de estas operaciones se produce un fallo, el saldo en CC o en CA o en ambas puede ser incorrecto.
  - **Idea importante (sobre todo para el usuario que está transfiriendo los fondos)**: Que tengan lugar todas las operaciones o, en caso de fallo, que no tenga lugar ninguna.
    - Si algo falla entre el "restar X de CC" y "sumar X a CA", el usuario de la CC puede verse muy afectado

# 1.2 Operaciones de Lectura y Escritura

- En todo el capítulo hablaremos de transacciones en el nivel de elementos de información o bloques de disco. En este nivel, las operaciones de una transacción sólo pueden incluir las siguientes operaciones:
  - **Leer (X)**: Lee un elemento X de la base de datos lo coloca en una variable de programa (por simplificar, supondremos que la variable de programa también se llama X)
  - **Escribir (X)**: Escribe una variable de programa llamada X en un elemento de la base de datos también llamado X
- Un **bloque** es la unidad básica de transferencia de disco a memoria principal. En general, un "elemento de información" puede ser un campo de la base de datos, una fila completa, o, incluso, toda una tabla. Este concepto es conocido como "granularidad".
- Normalmente, un **elemento de datos** será uno de estos:
  - un valor de **campo** de un registro de la BD
  - un **registro** de la BD
  - una **página** (uno o varios bloques de disco)
  - un **fichero**
  - la **BD** completa
- **Granularidad** = tamaño del elemento de información
  - Granularidad **fina** ➡ elementos de tamaño pequeño
  - Granularidad **gruesa**➡ elementos grandes

# 1.2 Operaciones de Lectura y Escritura

- En general, la ejecución de una orden **Leer(X)** incluye los siguientes pasos:
  - Encontrar la dirección del bloque en disco que contiene al elemento X
  - Copiar dicho bloque en el almacenamiento intermedio (buffer) dentro de la memoria principal (si dicho bloque no estaba ya en el almacenamiento intermedio)
  - Copiar el elemento X del almacenamiento intermedio (buffer) a la variable de programa llamada X

- La ejecución de una orden **Escribir (X)** incluye los siguientes pasos:
  - Encontrar la dirección del bloque en disco que contiene al elemento X
  - Copiar ese bloque de disco a un almacenamiento intermedio de la memoria principal (si no estaba)
  - Copiar el elemento X desde la variable de programa llamada X al bloque de disco
  - Transferir el bloque actualizado desde el almacenamiento intermedio hasta el disco (ya sea de forma inmediata, o bien en algún momento posterior)

# 1.2 Operaciones de Lectura y Escritura

## Elección del tamaño adecuado del elemento de datos

En el contexto de los métodos de bloqueo, **el tamaño del elemento de datos afecta al grado de concurrencia**:

- ⬇ tamaño(elemento) ⇨ ⬆ Grado de **concurrencia**

  Y también...

  - ⇨ ⬆ número de **elementos** en la BD
  - ⇨ ⬆ **carga de trabajo** para la **gestión de bloqueos**, y
  - ⇨ ⬆ **espacio** ocupado por la **información de bloqueo**

Pero... ¿Cuál es el **tamaño adecuado** para los elementos?

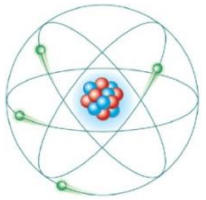
Pues **depende de la naturaleza de** las **transacciones T**:

- Si una T representativa **accede a pocos registros**
  - ⇨ elegir granularidad de **registro**
- Si T **accede a muchos registros de un mismo fichero**
  - ⇨ elegir granularidad de **página** o de **fichero**

# 1.3 Propiedades de las Transacciones

- Las Transacciones deben poseer 4 propiedades, conocidas como propiedades ACID (por sus siglas en ingles) y deben ser impuestas por los métodos de control de concurrencia y recuperación de los SGBD. Estas propiedades son:

  - **Atomicidad:**
    - El usuario debe pensar en las transacciones como en un "todo"; o bien se ejecutan todas sus operaciones o bien no se ejecuta ninguna.
  - **Consistencia:**
    - Tras la ejecución de una transacción, la BD debe quedar en un estado consistente.
  - **Aislamiento:**
    - Las transacciones están protegidas de los efectos de la ejecución concurrente con otras transacciones.
  - **Durabilidad:**
    - Tras finalizar con éxito una transacción, los cambios realizados en la BD son permanentes, incluso si hay fallos en el sistema.



**ACID**

**A**tomicity
**C**onsistency
**I**solation
**D**urability

# 1.3 Propiedades de las Transacciones

- Todas estas propiedades son muy importantes para:
  - Control de concurrencia
  - Recuperación
  - El Administrador de Transacciones: parte del SGBD que controla la ejecución de transacciones

- Las transacciones están delimitadas por:
  - Inicio de transacción
  - Fin de transacción (commit / abort )

- Todas las operaciones que se ejecuten entre el inicio y el fin son parte de la transacción.

# 1.3 Propiedades de las Transacciones

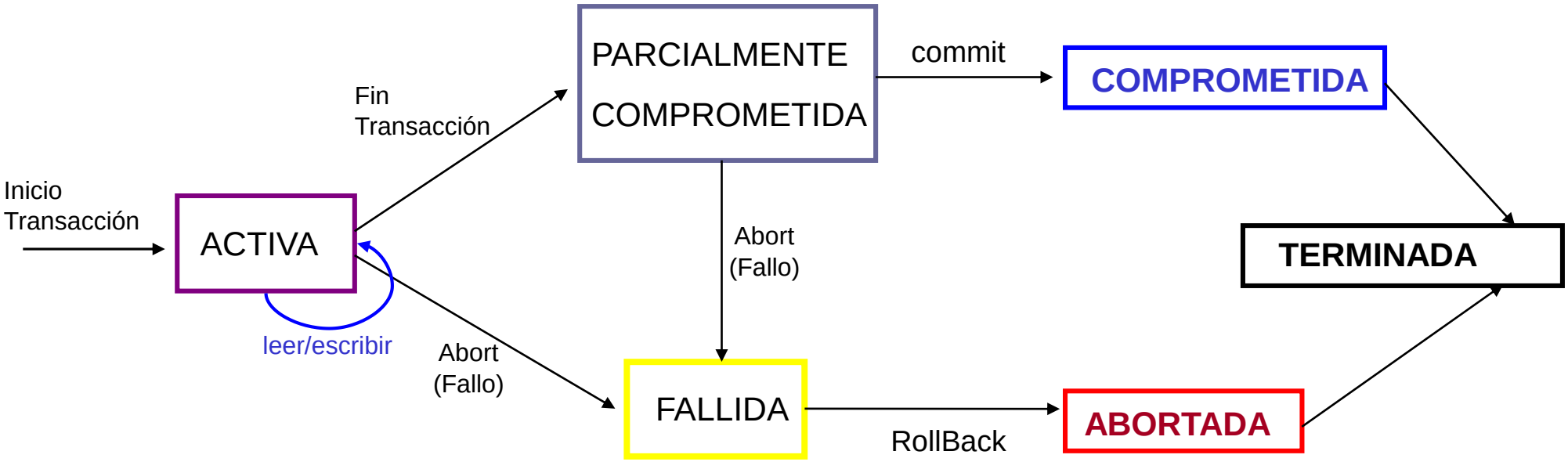



Diagrama de Transición de Estados
de la ejecución de una transacción

# 2. Interferencia de Transacciones. Nivel de paralelismo

- Cuando varias transacciones se ejecutan de forma intercalada, el orden en el que las operaciones se llevan a cabo constituye lo que se conoce como **plan** (o historia) de las transacciones

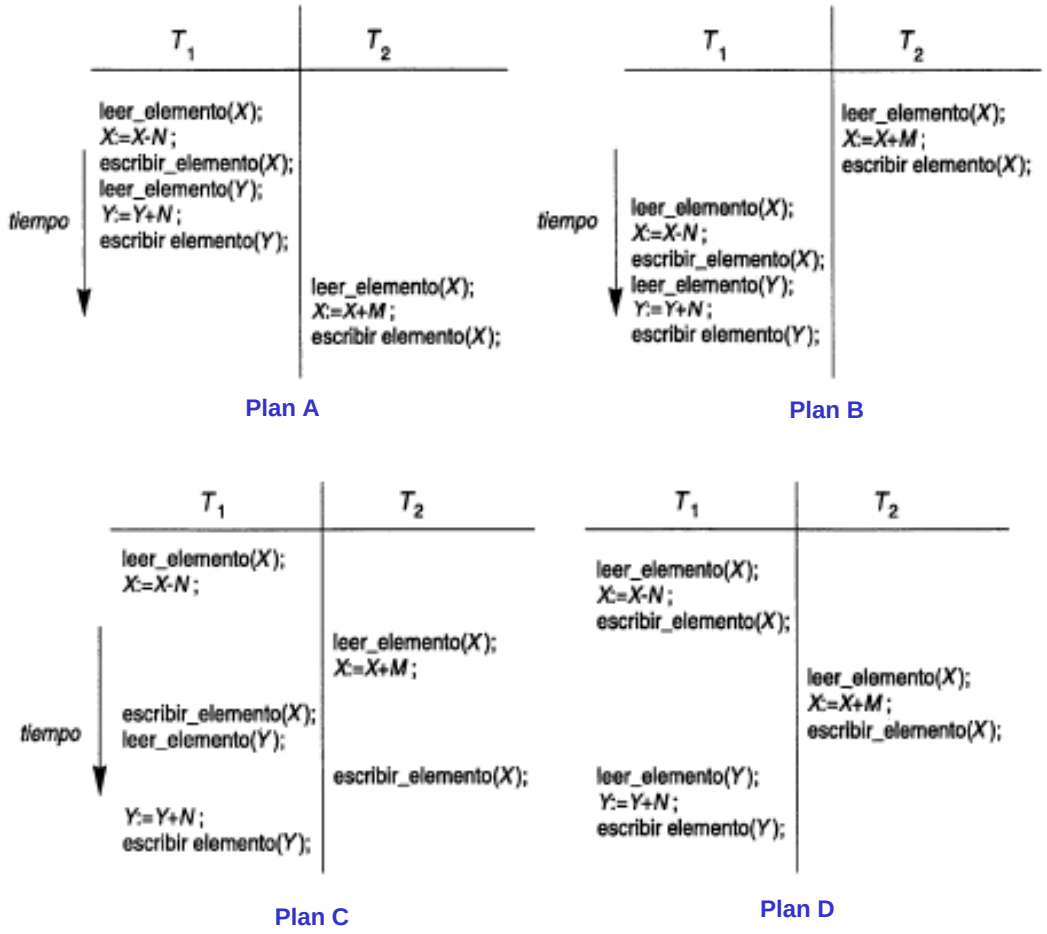
- Un **Plan** (o historia) P de *n* transacciones, T1, T2…Tn, es un ordenamiento de las operaciones de las transacciones sujeto a la restricción de que, para cada transacción Ti que participe en P, las operaciones de Ti en P deben aparecer en el mismo orden que en Ti

# 2. Interferencia de Transacciones. Nivel de paralelismo

**T1:**
```
leer_elemento(X);
X:=X-N;
escribir_elemento(X);
leer_elemento(Y);
Y:=Y+N;
escribir_elemento(Y);
```

**T2:**
```
leer_elemento(X);
X:=X+M;
escribir_elemento(X);
```

Ejemplo de dos transacciones




Por claridad, se muestran las operaciones que se realizan con los datos, pero, para el resto del tema, tan sólo se tendrán en cuenta las operaciones de lectura y escritura

Ejemplo de 4 planes de ejecuciones *intercaladas* de T1 y T2

# 2. Interferencia de Transacciones. Nivel de paralelismo

- Esquema (o plan) en serie.
  - Plan o esquema que no intercala las operaciones de diferentes transacciones.
- Esquema serializable.
  - Es aquel que es equivalente a un plan en serie.
- Esquemas equivalentes.
  - Para cualquier estado de la BD, el efecto de ejecución del primer esquema (o plan) es idéntico al de la ejecución del segundo.

| $P_1$ | $P_2$ |
|---|---|
| leer_elemento(X); | leer_elemento(X); |
| X:=X+10; | X:=X*1.1; |
| escribir_elemento(X); | escribir_elemento(X); |

Estos planes son equivalentes para el valor inicial X=100, pero **no** lo son en general

# 2. Interferencia de Transacciones. Nivel de paralelismo

- **Nivel de paralelismo (o nivel de concurrencia):**

  Grado de aprovechamiento de los recursos de proceso disponibles, en función del solapamiento de la ejecución de las transacciones que acceden concurrentemente a la BD y se confirman.

- **Objetivo:**
  - Conseguir el nivel de paralelismo adecuado: No se produzcan cancelaciones o suspensiones de ejecución cuando no es necesario para impedir una interferencia.
  - Compromiso entre el nivel de paralelismo y el coste de las tareas de control

- **Ejemplo. Reserva de asientos en una línea aérea**.
  - **T1:** cancela N reservas de un vuelo (nº de asientos reservados en X) y reserva el mismo número de asientos en otro vuelo Y.
  - **T2:** reserva M asientos en el vuelo X.

# 2. Interferencia de Transacciones. Nivel de paralelismo

- Operaciones:
  - T1: X:=X-N; Y:= Y+N.
  - T2: X:=X+M
- Planes:
  - T1: R(X); W(X); R(Y); W(Y);
  - T2: R(X); W(X);

**Reserva de asientos en una línea aérea**.

**T1**: cancela N reservas de un vuelo (nº de asientos reservados en X) y reserva el mismo número de asientos en otro vuelo Y.

**T2**: reserva M asientos en el vuelo X.

**Notación**

**R(X)** → Leer(X)
**W(X)** → Escribir (X)

**Transacción T1**

R(X) { leer_elemento(X); X:=X-N;
W(X) { escribir_elemento(X);
R(Y) { leer_elemento(Y); Y:=Y+N;
W(Y) { escribir_elemento(Y);

**Transacción T2**

R(X) { leer_elemento(X); X:=X+M;
W(X) { escribir_elemento(X);

**Operaciones de Lectura/Escritura en el ejemplo**

# 3. Problemas de ejecución concurrente: Actualización Perdida

**Problema 1**: La modificación realizada por T1 es sobrescrita por T2

T1: R(X) W(X) R(Y) W(Y) commit

T2: R(X) W(X) commit

- Problema de la **actualización perdida**
- Se produce cuando se pierde el cambio que ha afectado a una operación de escritura.

**Reserva de asientos en una línea aérea**.

**T1:** cancela N reservas de un vuelo (nº de asientos reservados en X) y reserva el mismo número de asientos en otro vuelo Y.

**T2:** reserva M asientos en el vuelo X.

# 3. Problemas de ejecución concurrente: Actualización Perdida

T1: R(X) W(X) R(Y) abort

T2: R(X) W(X) commit

Fallo en T1 (abort)

Al abortar la transacción T1, el sistema deshace sus cambios, con lo que se perderá la actualización de T2

**Reserva de asientos en una línea aérea**.

**T1:** cancela N reservas de un vuelo (nº de asientos reservados en X) y reserva el mismo número de asientos en otro vuelo Y.

**T2:** reserva M asientos en el vuelo X.

# 3. Problemas de ejecución concurrente: Lectura no Confirmada

**Problema 2**. Alguna transacción lee datos y posteriormente se produce un fallo.

T1: R(X) W(X) R(Y) abort. Fallo en T1 (abort)

T2: R(X) W(X)

**T1** falla ☞ X vuelve a su primitivo valor.

**T2** ha leído datos "sucios", incorrectos o no comprometidos, lo que puede modificar la BD produciendo inconsistencia.

**Reserva de asientos en una línea aérea**.

**T1:** cancela N reservas de un vuelo (nº de asientos reservados en X) y reserva el mismo número de asientos en otro vuelo Y.

**T2:** reserva M asientos en el vuelo X.

# 3. Problemas de ejecución concurrente:
## Lectura no Repetible

**Problema 3**. Una transacción lee dos veces el mismo dato y obtiene diferentes valores en cada lectura

T1: R(X) W(X) commit

T2: R(X) R(X) commit

# 3. Problemas de ejecución concurrente: Resumen Incorrecto

- Visión que dos transacciones tienen respecto a un conjunto de datos.

- Ejemplo:
  - **T3:** calcula el nº total de reservas de todos los vuelos.
    - T3: suma:=0;suma:=suma+A; suma:=suma+B;...;suma:=suma+Z;
  - **T4:** cancela N asientos de X y se los asigna a Y.
    - T4: X:=X-N; Y:=Y+N.

Visión de cada Transacción:

**T3:** lee unos datos (A, B, …, Z)

**T4:** actualiza una parte de estos datos (X e Y)

# 3. Problemas de ejecución concurrente:
## Resumen Incorrecto

**T3:** R(A);W(s);....R(Z);W(s)

**T4:** R(X);W(X);R(Y);W(Y)

- **T3:** R(A)W(s) ... R(X)W(s)R(Y)W(s) ... R(Z);W(s) commit
- **T4:** R(X)W(X) R(Y) W(Y) commit

T3 lee X después de restarse X y lee Y antes de sumarse N ☞ resultado incorrecto por N asientos.

## ❑ Caso particular: fantasmas

- T1 lee todos los registros que cumplen la condición C
- T2 actualiza (o inserta) un registro, que no cumplía C, de manera que pasa a cumplir la condición C
- T1 vuelve a leer los datos iniciales o alguna información dependiente de los mismos ☞ T1 encuentra un fantasma

# 4. Serializabilidad y Recuperabilidad

- El objetivo del control de concurrencia es planificar las transacciones de forma tal que se evite cualquier interferencia entre ellas, impidiendo que se produzcan los problemas descritos anteriormente.
- No podemos dejar que se ejecuten las transacciones de una en una en un SGBD multiusuario.
- Los problemas descritos en la sección anterior dejan a la BD en un estado inconsistente.
- Una ejecución en serie nunca producirá un resultado inconsistente, por lo que las ejecuciones en serie se consideran correctas.
- El objetivo de la **serializabilidad** es encontrar planificaciones no serie que permitan ejecutar concurrentemente transacciones sin que éstas interfieran entre sí, y produciendo así un estado de la BD al que pueda llegarse mediante una ejecución en serie.
- Decimos que una planificación no en serie es correcta si produce los mismos resultados que alguna planificación en serie. A estas planificaciones las denominados serializables.

# 4. Serializabilidad y Recuperabilidad

- Con respecto a la serializabilidad, el orden en el que se ejecutan las lecturas y las escrituras es importante.
  - Dos transacciones que sólo leen nunca entran en conflicto.
  - Dos transacciones que leen o escriben elementos distintos nunca entran en conflicto
  - Si una transacción lee y otra escribe el mismo elemento de la BD el orden en el que lo hagan sí es importante.

- Criterios (ignorando posibles cancelaciones):
  - ✓ **Seriabilidad de visión**: Una ejecución alternada es correcta si hay una ejecución en serie equivalente.
    - ➢ Difícil de implementar
  - ✓ **Seriabilidad por conflictos**: Si una planificación P se puede transformar mediante una serie de intercambios de instrucciones no conflictivas en otra planificación P', se dice que P y P' son equivalentes por conflictos.

# 4. Serializabilidad y Recuperabilidad

- **Ejemplo:**

T1: R(A)W(A) R(B)W(B)
T2: R(A)W(A) R(B)W(B)

**Conflicto**: Aquellas operaciones de diferentes transacciones sobre el mismo elemento donde al menos una de ellas sea una operación de **escribir**.
**Regla:** Se pueden **intercambiar** operaciones consecutivas que no estén en conflicto.

T1: R(A)W(A) R(B) W(B)
T2: R(A) W(A) R(B)W(B)

T1: R(A)W(A)R(B) W(B)
T2: R(A) W(A) R(B)W(B)

T1: R(A)W(A)R(B) W(B)
T2: R(A) W(A)R(B)W(B)

T1: R(A)W(A)R(B)W(B)
T2: R(A)W(A)R(B)W(B)

# 4. Serializabilidad y Recuperabilidad

- Una planificación **es recuperable** si para todo par de transacciones Ti, Tj, tales que Tj lee (o escribe) elementos de datos que Ti ha escrito previamente.
- La operación comprometer (commit) de Ti aparece **antes** que la de Tj.

- **Ejemplo 1 (no recuperable):**

  T1: R(A)W(A) R(B) abort
  T2: R(A) commit

  Fallo en escritura de T1 (Ti)

- **Ejemplo 2 (serial, no recuperable):**

  T3: W(A) abort
  T4: W(A) commit

  Fallo en escritura de T3 (Ti)

- **Ejemplo 3 (serial, no recuperable):**

  T5: R(A) commit
  T6: W(A) abort

  Fallo en escritura de T6 (Ti)

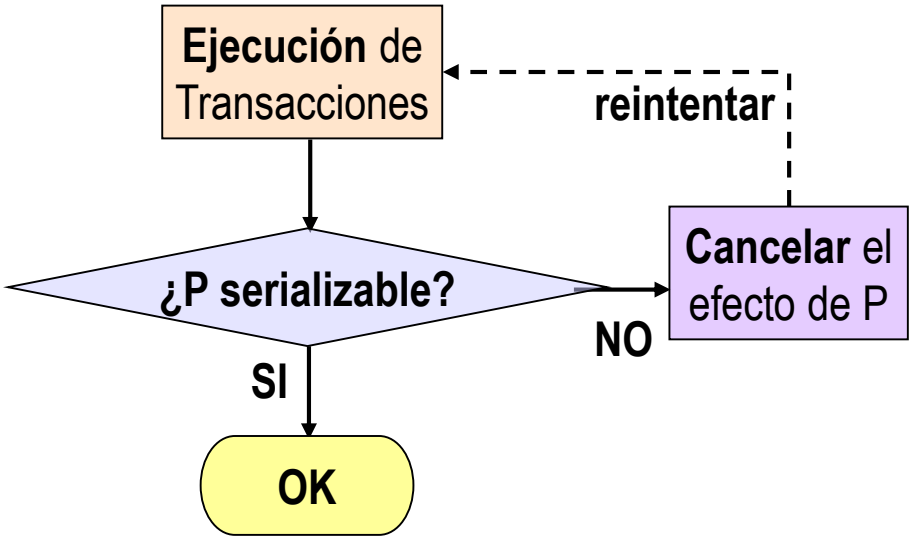
- **Ejemplo 4 (serializable y recuperable)**

  T7: R(A) commit
  T8: W(A) commit

  **T7** (Tj) lee un dato que **T8** (Ti) ha escrito previamente

# 4. Serializabilidad y Recuperabilidad

- Parece, pues, que habría que comprobar si un plan P es serializable **una vez ejecutadas** las transacciones incluidas en P...




Es necesario encontrar **técnicas que garanticen la serializabilidad**, sin tener que verificar a posteriori

- Métodos **basados en la teoría de la serializabilidad**, que definen un conjunto de **reglas** (o protocolo) tal que...
  - si todas las transacciones las cumplen, o
  - el subsistema de control de concurrencia del SGBD las impone (automáticamente)

  ... **se asegura la serializabilidad de toda planificación** de transacciones

# 4. Serializabilidad y Recuperabilidad

- Se permiten los siguientes Niveles de Aislamiento:
  - ✓ Serializable (por defecto):
    - La elección de este nivel afecta sólo a quien lo elige y no al resto de usuarios de la BD.
  - ✓ Repeatable Read:
    - Este nivel evita el problema de “lectura no repetible”
  - ✓ Read Commited:
    - Este nivel evita la lectura sucia, ya que la transacción sólo podrá leer datos que han sido reafirmados por el commit de otra transacción.
  - ✓ Read Uncommited:
    - Es el nivel más permisivo
    - Una transacción que se ejecuta con este nivel puede ver valores que otra transacción ha escrito, o deja ver valores que otra haya borrado, a pesar de que ésta no haya hecho commit.
    - Permite lecturas sucias, lecturas no repetibles y lecturas fantasmas

- Para establecer estos niveles se usa *set transaction*: “set transaction isolation level read commited”

# 4. Serializabilidad y Recuperabilidad

- Relajación del nivel de aislamiento:
  - ✓ Hay situaciones en las que es conveniente limitar el nivel de aislamiento de las transacciones para mejorar la concurrencia (sobre todo si se sabe que las interferencias no se van a producir realmente o no es importante que se produzcan).

| CONFLICTO / NIVEL AISLAMIENTO | ACTUALIZACION PERDIDA | LECTURA NO CONFIRMADA | LECTURA NO REPETIBLE Y ANALISIS INCONSISTENTE (EXCEPTO FANTASMAS) | FANTASMAS |
|---|---|---|---|---|
| **READ UNCOMMITED** | SI | NO | NO | NO |
| **READ COMMITED** | SI | SI | NO | NO |
| **REPEATABLE READ** | SI | SI | SI | NO |
| **SERIALIZABLE** | SI | SI | SI | SI |

**SI=** Si Protege / **NO=** No Protege

# 4. Serializabilidad y Recuperabilidad

## EJEMPLO Niveles de Aislamiento

- ✓ Supongamos una BD con la relación:

  **VENDE** (bar, cerveza, precio)

  CP: (bar, cerveza)

- ✓ Supongamos que el bar "Pepe" sólo vende dos marcas de cerveza:

  "Cruzcampo - 2€" y "Mahou – 2,5€"

- ✓ Supongamos que existen dos usuarios en la BD: Pepe y Juan
- ✓ Estos dos usuarios desean realizar las siguientes transacciones:
  - **T1:** Juan desea saber cual es la cerveza más cara y más barata del bar "Pepe"
  - **T2:** Pepe elimina las cervezas Cruzcampo y Mahou y pasa a vender sólo "San Miguel – 3€"

- ✓ Estas dos transacciones definirían las siguientes operaciones:

  [Juan] T1: - Select Max (precio) From VENDE Where bar="Pepe" → MAX

  - Select Min (precio) From VENDE Where bar="Pepe" → MIN

  [Pepe] T2: - Delete From VENDE Where bar="Pepe" → DEL

  - Insert Into VENDE Values ('Pepe', 'San Miguel', 3) → INS

# 4. Serializabilidad y Recuperabilidad

- Si ambas transacciones T1 y T2 se ejecutan simultáneamente, lo único que podemos afirmar es que MAX va antes que MIN en T1, y que DEL va antes que INS en T2.
- Un posible plan ejecución de ambas transacciones podría ser:

| | | | |
|---|---|---|---|
| [Juan] T1 | MAX | | MIN |
| [Pepe] T2 | | DEL | INS |

Juan lee que: MAX= Mahou -2.5€
MIN= San Miguel - 3€

- **Nivel Serializable:** Si Juan ejecuta sus instruciones con este nivel → verá la BD antes o después de las instrucciones de Pepe, pero nunca en medio
  - La elección del nivel serializable afecta sólo a quien lo elige (y no al resto)
  - Si Pepe ejecuta T2 con nivel serializable y Juan no → Juan podría ver los datos como si T1 se ejecutara a la mitad de la de Pepe

# 4. Serializabilidad y Recuperabilidad

- **Nivel READ Commited:** Este nivel evita la "Lectura sucia" (o no confirmada) → La transacción sólo podrá leer los datos que hayan sido confirmados (commit) por otra transacción.

- Ejemplo de lectura sucia:

| | | | | |
|---|---|---|---|---|
| [Juan] T1 | | MAX | MIN | |
| [Pepe] T2 | DEL | INS | | Rollback |

Juan leerá 3€ como precio MAX y MIN aunque ese dato nunca llegue a existir

- En este nivel, el SGBD asegura que Juan no lea 3€ si Pepe hace Rollback
- Este nivel es más permisivo que el serializable, ya que pueden darse problemas, como:

| | | | | |
|---|---|---|---|---|
| [Juan] T1 | MAX | | MIN | |
| [Pepe] T2 | | DEL | INS | Commit |

Sería factible en este nivel siempre que Pepe haga commit → En ese caso, Juan verá: MAX= Mahou -2.5€
MIN= San Miguel - 3€

# 4. Serializabilidad y Recuperabilidad

- **Nivel Repeatable READ:** Este nivel evita la "Lectura no repetible" → Este nivel es similar al "Read Commited" añadiendo la restricción de que "*en una transacción todo lo que se vió en una lectura inicial debe ser visto si se ejecuta de nuevo posteriormente la lectura*"
  - La segunda y posteriores lecturas pueden tener más datos que la inicial, pero nunca menos datos
- Supongamos que Juan ejecuta T1 con nivel "Repeatable Read":

| [Juan] T1 | MAX | | MIN |
|---|---|---|---|
| [Pepe] T2 | | DEL INS | |

El sistema debe asegurar que al ejecutar MIN se vean además del valor 3€, los valores 2€ y 2,5€ vistos en MAX

En este caso Juan verá: MAX= 2,5€ y MIN= 2€, aunque estos datos no reflejan el estado actual de la BD

- Este nivel es más permisivo que el serializable, ya que pueden darse problemas, como:

| [Juan] T1 | MAX | | MAX |
|---|---|---|---|
| [Pepe] T2 | | DEL INS | |

Si Juan ejecuta T1 con nivel "Repeatable Read" → lo que lee en el 1º MAX lo lee También en el 2º MAX, pero en un caso obtiene: MAX= 2,5€ y en otro MAX= 3€

# 4. Serializabilidad y Recuperabilidad

- **Nivel READ Uncommited:** Este nivel más permisivo.
  - Una transacción que se ejecute con este nivel de aislamiento puede ver valores que haya escrito otra transacción o dejar de ver valores que ésta borre, a pesar de que la transacción no haya hecho commit (y nunca lo haga)

- Juan podría ver el valor 3€ como precio MAX o MIN, a pesar de que Pepe posteriormente al INS "San Miguel - 3€" haga Rollback.

- Este nivel permite por tanto:
  - Lectura Sucia
  - Lectura no repetible
  - Lectura fantasma