# CC3201-1

## Bases de Datos
## Otoño 2018

## Clase 11: Transacciones y ACID

Aidan Hogan
aidhog@gmail.com

# Una cuenta bancaria ...

**Ingreso**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Deposito iniciál | 2020-01-21 | 20:02:02 | 300000 | 300000 | TRCXGU8JSHD |
| 7873698669 | C0°0°L Designs | 2020-02-06 | 09:15:33 | 50000 | 325000 | TRCCIA2J8A0 |

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,42 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,0001533 |
| USD | CLP | 652,2750000 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

# Una cuenta bancaria ... integridad

**Ingreso**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Deposito iniciál | 2020-01-21 | 20:02:02 | 300000 | 300000 | TRCXGU8JSHD |
| 7873698669 | C0°0°L Designs | 2020-02-06 | 09:15:33 | 50000 | 325000 | TRCCIA2J8A0 |

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,42 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,0001533 |
| USD | CLP | 652,2750000 |

# Restricciones sobre varias tablas (!!)

**Ingreso**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Deposito iniciál | 2020-01-21 | 20:02:02 | 300000 | 300000 | TRCXGU8JSHD |
| 7873698669 | C0°0°L Designs | 2020-02-06 | 09:15:33 | 50000 | 325000 | TRCCIA2J8A0 |

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,94 |



```
CREATE TABLE Cuenta (
  número INTEGER PRIMARY KEY,
  rut VARCHAR (12) NOT NULL,
  tipo VARCHAR (12) NOT NULL,
  saldo_clp BIGINT NOT NULL,
  saldo_usd FLOAT NOT NULL,
  CHECK (
   ( SELECT SUM(monto) FROM Ingreso WHERE cuenta=número )
    - ( SELECT SUM(monto) FROM Gasto WHERE cuenta=número )
      = saldo_clp ) )
```

*¿A. P. A.? ...*

```
INSERT INTO Ingreso ...
```

```
INSERT INTO Gasto ...
```

```
UPDATE Cuenta
SET saldo_clp ...
```

*¿A. S.? ...*

# TRANSACCIONES

# Transacciones

Una transacción es

un conjunto de operaciones que

se ejecutan de manera atómica

(es decir, como si fuera una sola operación)

# Vacaciones …

# Transacciones: START TRANSACTION/COMMIT

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |
| 7873698669 | Noruega | 2020-02-12 | 02:14:20 | 400000 | -175000 | TRCLK9K24KS |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | -175000 | -268,29 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,001533 |
| USD | CLP | 652,275000 |

```
START TRANSACTION;
  INSERT INTO Gasto VALUES
    (7873698669,'Noruega','2020-02-12','02:14:20',400000,-175000,'TRCLK9K24KS');
  UPDATE Cuenta SET saldo_clp=-175000, saldo_usd=-268.29 WHERE número=7873698669;
COMMIT;
```

START TRANSACTION (o a veces BEGIN) inicia la transacción

COMMIT realiza/guarda los cambios

# Transacciones (por defecto)

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |
| 7873698669 | Noruega | 2020-02-12 | 02:14:20 | 400000 | -175000 | TRCLK9K24KS |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | -175000 | -268,29 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,001533 |
| USD | CLP | 652,275000 |

```
INSERT INTO Gasto VALUES
  (7873698669,'Noruega','2020-02-12','02:14:20',400000,-175000,'TRCLK9K24KS');
   -- COMMIT;

UPDATE Cuenta SET saldo_clp=-175000, saldo_usd=-268.29 WHERE número=7873698669;
   -- COMMIT;
```

Si no hay una transacción explicita, por defecto, Postgres hace un COMMIT después de cada sentencia (pero se puede cambiar la configuración)

# Transacciones: ROLLBACK

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,94 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,001533 |
| USD | CLP | 652,275000 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

```
START TRANSACTION;
  INSERT INTO Gasto VALUES
    (7873698669,'Noruega','2020-02-12','02:14:20',400000,-175000,'TRCLK9K24KS');
ROLLBACK;
```

ROLLBACK deshace/borra los cambios desde el inicio de la transacción

# Transacciones: SAVEPOINT

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |
| 7873698669 | Noruega | 2020-02-12 | 02:14:20 | 400000 | -175000 | TRCLK9K24KS |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | -175000 | -268,29 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,001533 |
| USD | CLP | 652,275000 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

```
START TRANSACTION;
  INSERT INTO Gasto VALUES
   (7873698669,'Noruega','2020-02-12','02:14:20',400000,-175000,'TRCLK9K24KS');
  UPDATE Cuenta SET saldo_clp=-175000, saldo_usd=-268.25 WHERE número=7873698669;
  SAVEPOINT CompraNoruega;
  INSERT INTO Gasto VALUES
   (7873698669,'BOSE','2020-02-12',200000,-375000,'TRCASD8PNAK');
  ROLLBACK TO SAVEPOINT CompraNoruega;
COMMIT;
```

ROLLBACK puede deshacer/borrar los cambios desde un punto especifico con SAVEPOINT

# Una transacción con valores dinámicos

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,94 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,001533 |
| USD | CLP | 652,275000 |

```
START TRANSACTION;
  INSERT INTO Gasto VALUES
    (7873698669,'Noruega','2020-02-12','02:14:20',400000,-175000,'TRCLK9K24KS');
  UPDATE Cuenta SET saldo_clp=(saldo_clp-400000) WHERE número=7873698669;
  UPDATE Cuenta SET saldo_usd=(A.saldo_clp/valor)
    FROM ( SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') T,
       ( SELECT saldo_clp FROM Cuenta WHERE número=7873698669 ) A
    WHERE número=7873698669;
COMMIT;
```

*¿Valor final de* saldo_usd *en* Cuenta*?*

# Una transacción con valores dinámicos

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |
| 7873698669 | Noruega | 2020-02-12 | 02:14:20 | 400000 | -175000 | TRCLK9K24KS |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | −175000 | −268,28 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,001533 |
| USD | CLP | 652,275000 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

```
START TRANSACTION;
  INSERT INTO Gasto VALUES
    (7873698669,'Noruega','2020-02-12','02:14:20',400000,-175000,'TRCLK9K24KS');
  UPDATE Cuenta SET saldo_clp=(saldo_clp-400000) WHERE número=7873698669;
  UPDATE Cuenta SET saldo_usd=(A.saldo_clp/valor)
    FROM ( SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') T,
       ( SELECT saldo_clp FROM Cuenta WHERE número=7873698669 ) A
    WHERE número=7873698669;
COMMIT;
```

*¿Valor final de* saldo_usd *en* Cuenta*?*

−268,28 (Se lee el valor actual de la misma transacción)

# Una transacción con CHECK

**Gasto**

| cuenta | comentario | fecha | hora | monto | saldo | id |
|---|---|---|---|---|---|---|
| 7873698669 | Electricidad | 2020-02-02 | 20:00:01 | 8200 | 291800 | TRCJASJDA9A |
| 7873698669 | Calefacción | 2020-02-02 | 20:00:02 | 600 | 291200 | TRC81KAQWAS |
| 7873698669 | Moviestar | 2020-02-02 | 20:00:03 | 16200 | 275000 | TRCK8J7JA8D |
| 7873698669 | Cajero | 2020-02-08 | 16:05:02 | 100000 | 225000 | TRCPM8A45AD |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,94 |

**Divisa**

| d1 | d2 | valor |
|---|---|---|
| CLP | USD | 0,001533 |
| USD | CLP | 652,275000 |

**Cliente**

| rut | nombre | fono | dirección |
|---|---|---|---|
| 32.000.273-K | Kelvin | +56976698463 | Campo de Hielo Sur, Depto 273 |

```
START TRANSACTION;
  INSERT INTO Gasto VALUES
    (7873698669,'Noruega','2020-02-12','02:14:20',400000,-175000,'TRCLK9K24KS');
  UPDATE Cuenta SET saldo_clp=(saldo_clp-400000) WHERE número=7873698669;
  UPDATE Cuenta SET saldo_usd=(A.saldo_clp/valor)
    FROM ( SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') T,
      ( SELECT saldo_clp FROM Cuenta WHERE número=7873698669 ) A
    WHERE número=7873698669;
COMMIT;
```

¿Funciona?

¡No!

```
CREATE TABLE Cuenta ( ...
 CHECK ( saldo_clp = saldo_usd * (SELECT valor FROM Divisa WHERE d1='USD' AND d2='CLP') ) )
```

# Transacciones / Restricciones: IMMEDIATE

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,94 |

```
CREATE TABLE Cuenta (
  ...,
  CONSTRAINT Cuenta_PK
   PRIMARY KEY (número)
)
```

```
START TRANSACTION;
  INSERT INTO Cuenta VALUES
    (7873698669,'32.000.273-K','Estacional',
       -175000,-268.29);
  DELETE FROM Cuenta
    WHERE número=7873698669 AND saldo_clp=225000;
COMMIT;
```

Por defecto, se aplica la restricción inmediatamente después de cada sentencia

# Transacciones / Restricciones: DEFERRABLE

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,94 |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | -175000 | -268,29 |

```
CREATE TABLE Cuenta (
  ...,
  CONSTRAINT Cuenta_PK
   PRIMARY KEY (número)
   DEFERRABLE
)
```

```
START TRANSACTION;
  SET CONSTRAINT Cuenta_PK DEFERRED;
  INSERT INTO Cuenta
    (7873698669,'32.000.273-K','Estacional',
       -175000,-268.29);
  DELETE FROM Cuenta
    WHERE número=7873698669 AND saldo_clp=225000;
COMMIT;
```

DEFERRABLE define una restricción que *se puede* diferir hasta un COMMIT

DEFERRED difiere la restricción hasta el COMMIT en la transacción actual

# Transacciones / Restricciones: DEFERRABLE

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | 225000 | 344,94 |

**Cuenta**

| número | rut | tipo | saldo_clp | saldo_usd |
|---|---|---|---|---|
| 7873698669 | 32.000.273-K | Estacional | -175000 | -268,29 |

```
CREATE TABLE Cuenta (
  ...,
  CONSTRAINT Cuenta_PK
   PRIMARY KEY (número)
   DEFERRABLE INITIALLY DEFERRED
)
```

DEFERRABLE INITIALLY DEFERRED define una restricción que sea diferido por defecto hasta un COMMIT

```
START TRANSACTION;
  INSERT INTO Cuenta
    (7873698669,'32.000.273-K','Estacional',
       -175000,-268.29);
  DELETE FROM Cuenta
    WHERE número=7873698669 AND saldo_clp=225000;
COMMIT;
```

# WARNING

Restricciones diferibles son estándares

Pero Postgres no permite diferir restricciones usando ni `CHECK` ni `NOT NULL`

Atomicidad, Consistencia, Aislamiento, Durabilidad

(*Atomicity, Consistency, Isolation, Durability*)

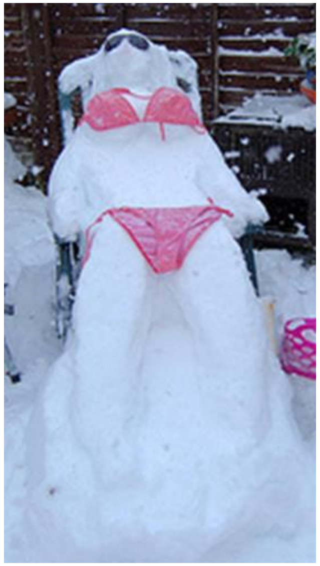
# Las garantías de ACID

Capítulo 8.1 | Ramakrishnan / Gehrke

# No hay un solo usuario …

… hay que tener cuidado con la concurrencia

# Una cuenta con varios usuarios



**Banco de Chilly**

| | | |
|---|---|---|
| N° Cuenta | : | 7873698669 |
| Saldo (CLP) | : | 225000 |
| Límite de crédito | : | 200000 |
| Disponible | : | 425000 |

```
CREATE TABLE Cuenta ( ..., CHECK ( saldo_clp > -200000 ) )
```

Compra(Islas de Caimán,300000)

```
INSERT INTO Gasto ...

UPDATE Cuenta ...
```

Compra(Noruega,400000)

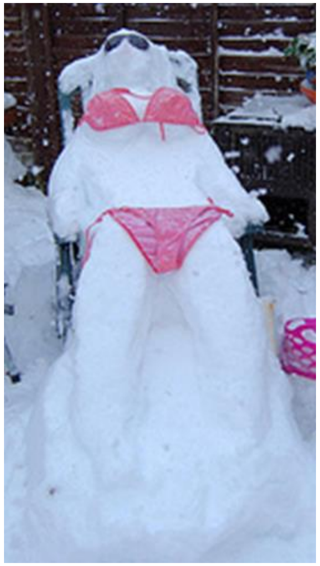
```
INSERT INTO Gasto ...

UPDATE Cuenta ...
```

*¿Qué será el resultado final? ...*

# *Caos*

# Esta vez con transacciones …



**Banco de Chilly**

| | | |
|---|---|---|
| N° Cuenta | : | 7873698669 |
| Saldo (CLP) | : | 225000 |
| Límite de crédito | : | 200000 |
| Disponible | : | 425000 |

```
CREATE TABLE Cuenta ( ..., CHECK ( saldo_clp > -200000 ) )
```

COMPRA(Islas de Caimán,300000)

```
START TRANSACTION
  INSERT INTO Gasto ...
  UPDATE Cuenta ...
COMMIT;
```

COMPRA(Noruega,400000)

```
START TRANSACTION
  INSERT INTO Gasto ...
  UPDATE Cuenta ...
COMMIT;
```

*¿Qué será el resultado final?*

Se rechazará una transacción.

# Garantías de ACID

- Atomicidad:
  - La ejecución de cada transacción es atómica:
    - Se realizan todas las acciones o no se realiza ninguna
- Consistencia:
  - Cada transacción debe preservar la integridad
    - La base de datos satisfacen todas las restricciones después de una transacción
- Aislamiento (*Isolation*):
  - Una transacción no puede afectar otra
- Durabilidad:
  - Una vez que haya un `COMMIT`, la base de datos debe persistir los cambios

# ACID: Un ejemplo más limpio

```
CREATE TABLE Balance (
    cuenta BIGINT PRIMARY KEY,
    total_gasto BIGINT,
    total_ingreso BIGINT,
    saldo BIGINT,
    CHECK (total_ingreso - total_gasto = saldo)
)
```

Usaré restricciones con CHECK porque dan ejemplos más claros pero es importante tener en cuenta que Postgres no suporte CHECKs diferibles.

# ACID: Atomicidad

```
CREATE TABLE Balance (
   cuenta BIGINT PRIMARY KEY,
   total_gasto BIGINT,
   total_ingreso BIGINT,
   saldo BIGINT,
   CHECK (total_ingreso - total_gasto = saldo)
)
```

```
START TRANSACTION
  UPDATE Balance SET saldo=saldo-10 WHERE Cuenta=7873698669 ;
  UPDATE Balance SET total_gasto=total_gasto+10 WHERE Cuenta=7873698669 ;
COMMIT;
```

**Atomicidad**

No se puede actualizar el saldo sin actualizar el gasto directamente después.

(Si alguna actualización falla, ambas fallan.)

# ACID: Consistencia

```
CREATE TABLE Balance (
   cuenta BIGINT PRIMARY KEY,
   total_gasto BIGINT,
   total_ingreso BIGINT,
   saldo BIGINT,
   CHECK (total_ingreso - total_gasto = saldo)
)
```

```
START TRANSACTION
  UPDATE Balance SET saldo=saldo-100 WHERE Cuenta=7873698669 ;
  UPDATE Balance SET total_gasto=total_gasto+10 WHERE Cuenta=7873698669 ;
COMMIT;
```

**Consistencia**

Si el resultado de la transacción no satisface todas las restricciones, fallará.

# ACID: Aislamiento (*Isolation*)

```
CREATE TABLE Balance (
   cuenta BIGINT PRIMARY KEY,
   total_gasto BIGINT,
   total_ingreso BIGINT,
   saldo BIGINT,
   CHECK (total_ingreso - total_gasto = saldo)
)
```

T1

```
START TRANSACTION
  UPDATE Balance
    SET saldo=saldo-10
    WHERE Cuenta=7873698669 ; (1)
  UPDATE Balance
    SET total_gasto=total_gasto+100
    WHERE Cuenta=7873698669 ; (3)
COMMIT;                 (4) ROLLBACK;
```

T2

```
START TRANSACTION
  UPDATE Balance
    SET saldo=saldo+100
    WHERE Cuenta=7873698669 ; (2)
  UPDATE Balance
    SET total_ingreso=total_ingreso+100
    WHERE Cuenta=7873698669 ; (5)
COMMIT; (6)
```

## Aislamiento

Una transacción no puede interferir con otra transacción.

En (4), hay que tener cuidado con el ROLLBACK: no se puede restaurar el valor de saldo antes del paso (1) porque el valor ya fue cambiado por (2).

# ACID: Durabilidad

```
CREATE TABLE Balance (
   cuenta BIGINT PRIMARY KEY,
   total_gasto BIGINT,
   total_ingreso BIGINT,
   saldo BIGINT,
   CHECK (total_ingreso - total_gasto = saldo)
)
```

```
START TRANSACTION
  UPDATE Balance SET saldo=saldo-10 WHERE Cuenta=7873698669 ;
  UPDATE Balance SET total_gasto=total_gasto+10 WHERE Cuenta=7873698669 ;
COMMIT;
```

## Durabilidad

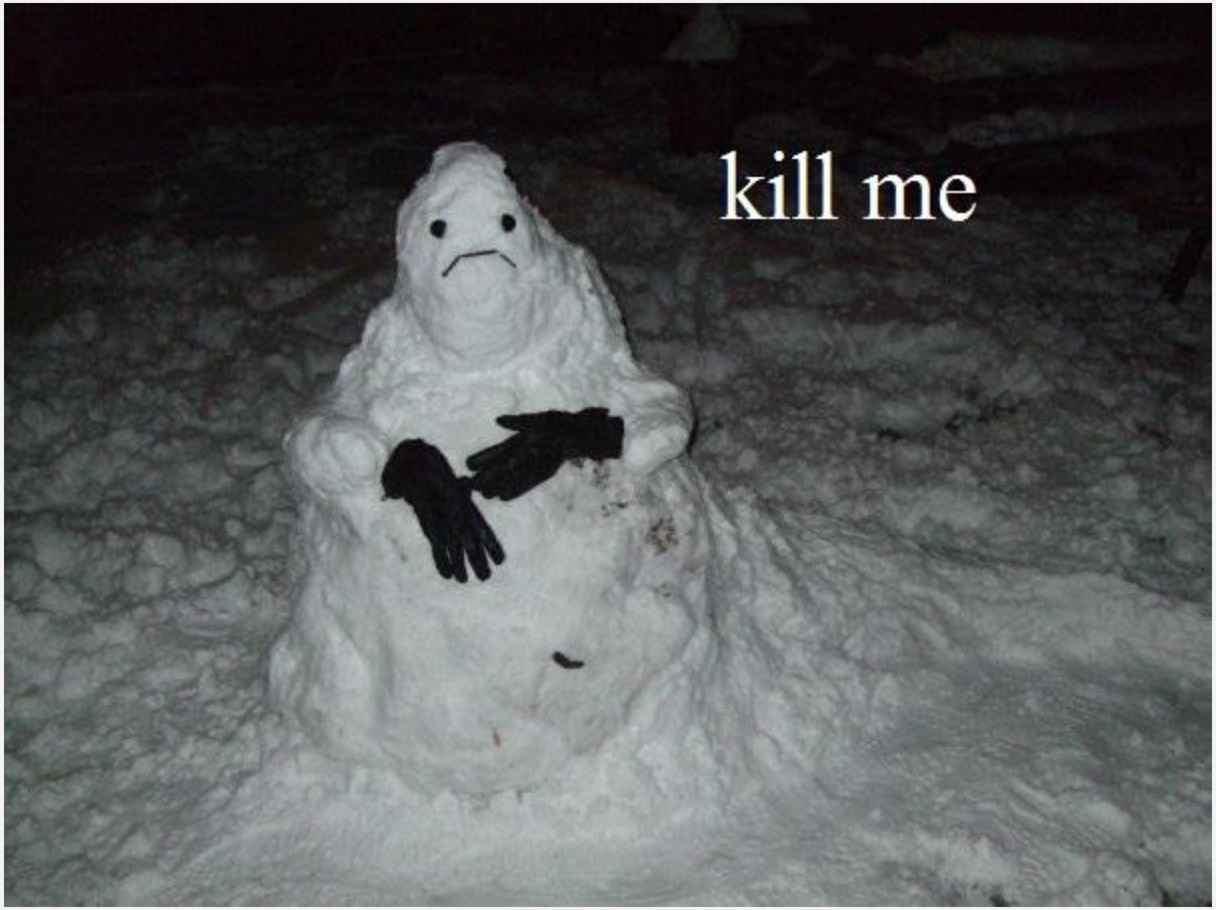
Una vez que haya un COMMIT exitoso, se persisten los cambios.

(Normalmente la persistencia aquí significa en el disco duro. Sin persistencia, en el caso de que la máquina falla y toda la evidencia de los cambios está en memoria principal, el sistema de base de datos olvidaré los cambios silenciosamente.)

# Entonces con las garantías de ACID …

… todo está tranquilo.

# … pero si uno tiene que *implementar* ACID




… es más difícil …

# ¿Cuándo **no** tenemos ACID?

- Atomicidad:
  - × Una transacción se ejecutan solamente a medias pero afecta el estado de la base de datos
- Consistencia:
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- Aislamiento (*Isolation*):
  - × El resultado final de dos transacciones no es equivalente a correr cada transacción en serie
- Durabilidad:
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.

# Modelando una transacción

$\textsc{Leer}(X)$: leer un objeto $X$ de la base de datos a memoria principal

$\textsc{Escribir}(X)$: escribir un objeto de memoria principal a la base de datos

Un objeto $X$ puede ser un valor, una fila, una tabla ...

*Ejemplo*: Una transferencia bancaria ...

T

$\textsc{Leer}(A)$
$A \leftarrow A - 100$
$\textsc{Escribir}(A)$
$\textsc{Leer}(B)$
$B \leftarrow B + 100$
$\textsc{Escribir}(B)$

$\leftarrow$ Dejamos el $\textsc{commit}$ implicito al fin de la transacción.

# Cuándo **no** tenemos ACID …

- Atomicidad:
  - × Una transacción se ejecutan sólo a medias pero afecta el estado de la base de datos
- Consistencia:
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- Durabilidad:
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.

*¿Cómo puede una implementación evitar estos tipos de problemas? …*

# Modelando una transacción

- Atomicidad:
  - × Una transacción se ejecutan solamente a medias pero afecta el estado de la base de datos

*Ejemplo*: Una transferencia bancaria ...

T

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

No se pueden realizar dos escrituras *exactamente* al mismo tiempo

# Implementando ACID:
# Registros (*Logging*)

# Mantener un registro de la transacción

*Ejemplo*: Una transferencia bancaria ...

T

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

```
T begin
T leer A 400
T escribir A 300
T leer B 200
T escribir B 300
T commit
```

./registro.log

# Si hay un problema, revertir el registro

La información en el registro debe bastar para revertir el estado de la base de datos sin ambigüedad

*Ejemplo*: Una transferencia bancaria ...

T

$\text{LEER}(A)$

$A \leftarrow A - 100$

$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$

$B \leftarrow B + 100$

$\text{ESCRIBIR}(B)$

```
T begin
T leer A 400
T escribir A 300
T leer B 200
T escribir B 300
T rollback
```

./registro.log

# Registros ayudan con ...

- Atomicidad:
  - × Una transacción se ejecutan sólo a medias pero afecta el estado de la base de datos
- Consistencia:
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- Aislamiento (*Isolation*):
  - × El resultado final de dos transacciones no es equivalente a correr cada transacción en serie
- Durabilidad:
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.

*¿Cuáles problemas podemos evitar con alguna forma de registro? ...*

# Concurrencia/Aislamiento: un problema abierto

```
CREATE TABLE Balance (
   cuenta INTEGER PRIMARY KEY,
   total_gasto BIGINT,
   total_ingreso BIGINT,
   saldo BIGINT,
   CHECK (total_ingreso - total_gasto = saldo)
)
```

T1

```
START TRANSACTION
  UPDATE Balance
    SET saldo=saldo-10
    WHERE Cuenta=7873698669 ;        (1)
  UPDATE Balance
    SET total_gasto=total_gasto+100
    WHERE Cuenta=7873698669 ;        (3)
COMMIT;                              (4) ROLLBACK;
```

T2

```
START TRANSACTION
  UPDATE Balance
    SET saldo=saldo+100
    WHERE Cuenta=7873698669 ;        (2)
  UPDATE Balance
    SET total_ingreso=total_ingreso+100
    WHERE Cuenta=7873698669 ;        (5)
COMMIT;  (6)
```

## Aislamiento

Una transacción no puede interferir con otra transacción.

En (4), hay que tener cuidado con el ROLLBACK: no se puede restaurar el valor de saldo antes del paso (1) porque el valor ya fue cambiado por (2).

# Implementando ACID:
# Secuenciabilidad (*Serializability*)

Capítulo 8.3 (*es*)/16.3 (*en*) | Ramakrishnan / Gehrke

# Registros no ayudan con …

- Atomicidad:
  - × Una transacción se ejecutan sólo a medias pero afecta el estado de la base de datos
- Consistencia:
  - × Al ejecutar la transacción, la base de datos no satisface las restricciones de integridad
- Aislamiento (*Isolation*):
  - × El resultado final de dos transacciones no es equivalente a correr cada transacción en serie
- Durabilidad:
  - × La base de datos se actualiza momentáneamente y luego vuelve al estado anterior.

*… entonces que podemos hacer con respecto a aislamiento/concurrencia? …*

# La solución más simple: ejecución serial

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

$T_1$

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

E
J
E
C
U
C
I
Ó
N

*¿Cuánto es A + B después?*

270 + 330 = 600

$T_2$

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

*¿Hay un problema aquí?*

¡Ejecución serial es lenta!

*¿Se pueden ejecutar partes de $T_1$ y $T_2$ en paralelo? …*

# ¿Ejecución paralela?

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

**T$_1$**

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

EJECUCIÓN

**T$_2$**

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

*¿Cuánto es A + B después?*

¯_(ツ)_/¯

Por simplicidad asumiremos un orden de ejecución en el caso paralelo ...

# ¿Ejecución paralela?

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

| $T_1$ | | $T_2$ |
|---|---|---|
| LEER$(A)$<br>$A \leftarrow A - 100$<br>ESCRIBIR$(A)$ | E | |
| | J<br>E<br>C | LEER$(A)$<br>$v \leftarrow A \times 0{,}1$<br>$A \leftarrow A - v$<br>ESCRIBIR$(A)$ |
| LEER$(B)$<br>$B \leftarrow B + 100$<br>ESCRIBIR$(B)$ | U<br>C<br>I<br>Ó | |
| | N | LEER$(B)$<br>$B \leftarrow B + v$<br>ESCRIBIR$(B)$ |

*¿Cuánto es A + B después?*

270 + 330 = 600

# ¿Ejecución paralela?

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

$T_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
ESCRIBIR$(A)$

$T_1$

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$

$T_2$

LEER$(B)$
$B \leftarrow B + v$
ESCRIBIR$(B)$

$T_1$

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

EJECUCIÓN

*¿Cuánto es A + B después?*

260 + 340 = 600

# ¡Cuidado cuando el orden importe!

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

*¿Cuánto es A + B después?*

260 + 340 = 600

EJECUCIÓN

T$_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
ESCRIBIR$(A)$
LEER$(B)$
$B \leftarrow B + v$
ESCRIBIR$(B)$

T$_1$

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$
LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

# ¡Cuidado cuando el orden importe!

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

T$_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
ESCRIBIR$(A)$

*¿Cuánto es A + B después?*

No tenemos ningún problema con la restricción, pero el orden de ejecución puede afectar el resultado.

*Dependiendo de la aplicación, a veces puede ser necesario preservar el orden de transacciones y/o aplicar ejecución serial.*

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$
LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

# ¿Ejecución paralela?

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

| $\mathrm{T}_1$ | EJECUCIÓN | $\mathrm{T}_2$ |
|---|---|---|
| $\mathrm{LEER}(A)$<br>$A \leftarrow A - 100$ | | |
| | | $\mathrm{LEER}(A)$<br>$v \leftarrow A \times 0{,}1$<br>$A \leftarrow A - v$<br>$\mathrm{ESCRIBIR}(A)$ |
| $\mathrm{ESCRIBIR}(A)$<br>$\mathrm{LEER}(B)$<br>$B \leftarrow B + 100$<br>$\mathrm{ESCRIBIR}(B)$ | | |
| | | $\mathrm{LEER}(B)$<br>$B \leftarrow B + v$<br>$\mathrm{ESCRIBIR}(B)$ |

*¿Cuánto es A + B después?*

300 + 340 = 640

*¿Dónde está el problema?*

# ¿Ejecución paralela?

*Ejemplo*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

| $\mathsf{T}_1$ | EJECUCIÓN | $\mathsf{T}_2$ |
|---|---|---|
| $\text{LEER}(A)$<br>$A \leftarrow A - 100$ | ↓ | |
| | ↓ | $\text{LEER}(A)$<br>$v \leftarrow A \times 0{,}1$<br>$A \leftarrow A - v$<br>$\text{ESCRIBIR}(A)$ |
| $\text{ESCRIBIR}(A)$ | ↓ | |
| $\text{LEER}(B)$<br>$B \leftarrow B + 100$<br>$\text{ESCRIBIR}(B)$ | ↓ | |
| | ↓ | $\text{LEER}(B)$<br>$B \leftarrow B + v$<br>$\text{ESCRIBIR}(B)$ |

*¿Cuánto es A + B después?*

300 + 340 = 640

*¿Dónde está el problema?*

# Planificaciones:

## Secuenciables vs. No Secuenciables

| $T_1$ | $T_2$ |
|---|---|
| LEER$(A)$<br>$A \leftarrow A - 100$<br>ESCRIBIR$(A)$ | |
| | LEER$(A)$<br>$v \leftarrow A \times 0{,}1$<br>$A \leftarrow A - v$<br>ESCRIBIR$(A)$ |
| LEER$(B)$<br>$B \leftarrow B + 100$<br>ESCRIBIR$(B)$ | |
| | LEER$(B)$<br>$B \leftarrow B + v$<br>ESCRIBIR$(B)$ |

"Secuenciable"

*Equivalente a ejecutar transacción $T_1$ y después transacción $T_2$ (serial)*

| $T_1$ | $T_2$ |
|---|---|
| LEER$(A)$<br>$A \leftarrow A - 100$ | |
| | LEER$(A)$<br>$v \leftarrow A \times 0{,}1$<br>$A \leftarrow A - v$<br>ESCRIBIR$(A)$ |
| ESCRIBIR$(A)$<br>LEER$(B)$<br>$B \leftarrow B + 100$<br>ESCRIBIR$(B)$ | |
| | LEER$(B)$<br>$B \leftarrow B + v$<br>ESCRIBIR$(B)$ |

"No secuenciable" ...

# Planificaciones:
## Secuenciables vs. No Secuenciables

Una **planificación** es una lista de acciones de un conjunto de transacciones en el orden de ejecución.

*Una **planificación secuenciable** tendrá el mismo efecto que una planificación serial (en algún orden de las transacciones).*

*¿Cómo se puede identificar una planificación secuenciable?*
*(sin ejecutarla y compararla con todas las posibles planificaciones seriales)? …*

*Seguiremos con la pregunta inversa … ¿cuándo no se puede garantizar que una planificación sea secuenciable?*

# (1) Conflicto de Lectura–Escritura

*Ejemplo 1*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

| $\mathrm{T}_1$ | EJECUCIÓN | $\mathrm{T}_2$ |
|---|---|---|
| LEER$(A)$<br>$A \leftarrow A - 100$ | | |
| | | LEER$(A)$<br>$v \leftarrow A \times 0{,}1$<br>$A \leftarrow A - v$<br>ESCRIBIR$(A)$ |
| ESCRIBIR$(A)$<br>LEER$(B)$<br>$B \leftarrow B + 100$<br>ESCRIBIR$(B)$ | | |
| | | LEER$(B)$<br>$B \leftarrow B + v$<br>ESCRIBIR$(B)$ |

*¿Cuánto es A + B después?*

300 + 340 = 640

# (1) Conflicto de Lectura–Escritura

*Ejemplo 1*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

| T$_1$ | EJECUCIÓN | T$_2$ |
|---|---|---|
| $\text{LEER}(A)$<br>$A \leftarrow A - 100$ | | |
| | | $\text{LEER}(A)$<br>$v \leftarrow A \times 0{,}1$<br>$A \leftarrow A - v$<br>$\text{ESCRIBIR}(A)$ |
| $\text{ESCRIBIR}(A)$<br>$\text{LEER}(B)$<br>$B \leftarrow B + 100$<br>$\text{ESCRIBIR}(B)$ | | |
| | | $\text{LEER}(B)$<br>$B \leftarrow B + v$<br>$\text{ESCRIBIR}(B)$ |

*¿Cuánto es A + B después?*

300 + 340 = 640

**Conflicto de Lectura–Escritura:**
**(Lecturas no repetibles)**

*T$_2$ cambia el valor de A que T$_1$ ha leído pero todavía no ha escrito*

# (2) Conflicto de Escritura–Lectura

*Ejemplo 2*: Una transferencia y un pago de interés
($A$ inicia con 400, $B$ con 200)

T$_1$

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$

T$_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A + v$
ESCRIBIR$(A)$
LEER$(B)$
$w \leftarrow B \times 0{,}1$
$B \leftarrow B + w$
ESCRIBIR$(B)$

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

EJECUCIÓN

*¿Cuántos serían A y B si se ejecutaran T$_1$ y después T$_2$?*

330 y 330

*¿… T$_2$ y después T$_1$?*

340 y 320

*¿Cuántos serán A y B aquí?*

330 y 320

# (2) Conflicto de Escritura–Lectura

*Ejemplo 2*: Una transferencia y un pago de interés
($A$ inicia con 400, $B$ con 200)

$T_1$

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$

¿Cuántos serían $A$ y $B$ si se ejecutaran $T_1$ y después $T_2$?

330 y 330

¿… $T_2$ y después $T_1$?

340 y 320

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

EJECUCIÓN

L

$v$

$A \leftarrow A + v$
ESCRIBIR$(A)$
LEER$(B)$
$w \leftarrow B \times 0{,}1$
$B \leftarrow B + w$
ESCRIBIR$(B)$

¿Cuántos serán $A$ y $B$ aquí?

330 y 320

**Conflicto de Escritura–Lectura:**

*El orden de cambio de A es $T_1$ / $T_2$ mientras el orden de B es $T_2$ / $T_1$, entonces el orden de ambos no es ni $T_1$ / $T_2$ ni $T_2$ / $T_1$*

# (3) Conflicto de Escritura–Escritura

*Ejemplo 3*: Fijando los dos valores de las cuentas $A$ y $B$ desde $C$ y $D$
($A$ inicia con 400, $B$ con 200, $C$ con 1000, $D$ con 1)

$T_1$

LEER($C$)
$A \leftarrow C$
ESCRIBIR($A$)

$T_2$

LEER($D$)
$B \leftarrow D$
ESCRIBIR($B$)

$B \leftarrow C$
ESCRIBIR($B$)

$A \leftarrow D$
ESCRIBIR($A$)

EJECUCIÓN

*¿Cuántos serán A y B aquí?*

1 y 1000

*¿Secuenciable?* ¡No!

# (3) Conflicto de Escritura–Escritura

*Ejemplo 3*: Fijando los dos valores de las cuentas $A$ y $B$ desde $C$ y $D$
($A$ inicia con 400, $B$ con 200, $C$ con 1000, $D$ con 1)

| $T_1$ | EJECUCIÓN | $T_2$ |
|---|---|---|
| LEER$(C)$<br>$A \leftarrow C$<br>ESCRIBIR$(A)$ | | |
| | | LEER$(D)$<br>$B \leftarrow D$<br>ESCRIBIR$(B)$ |
| $B \leftarrow C$<br>ESCRIBIR$(B)$ | | |
| | | $A \leftarrow D$<br>ESCRIBIR$(A)$ |

¿Cuántos serán A y B aquí?

1 y 1000

¿Secuenciable? ¡No!

**Conflicto de Escritura–Escritura:**

$T_2$ *sobrescribe* $T_1$ *y* $T_1$ *sobrescribe* $T_2$
(no hay problema con lecturas aquí)

**T$_1$**

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$

**T$_2$**

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A + v$
ESCRIBIR$(A)$
LEER$(B)$
$w \leftarrow B \times 0{,}1$
$B \leftarrow B + w$
ESCRIBIR$(B)$

**T$_1$**

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

*¿Qué tipo de conflicto hay aquí?*

**Conflicto de Escritura–Lectura:**

*A* cambia con T$_1$ / T$_2$

*B* cambia con T$_2$ / T$_1$

**T$_1$**

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$

**T$_2$**

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
ESCRIBIR$(A)$
LEER$(B)$
$B \leftarrow B + v$
ESCRIBIR$(B)$

**T$_1$**

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

*¿Qué tipo de conflicto hay aquí?*

**¡Es secuenciable!**

(No hay conflicto)

T$_1$

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$

T$_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A + v$
ESCRIBIR$(A)$
LEER$(B)$
$w \leftarrow B \times 0{,}1$
$B \leftarrow B + w$
ESCRIBIR$(B)$

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

*¿Qué tipo de conflicto hay aquí?*

Conflicto de Escritura–Lectura:

*A* cambia con T$_1$ / T$_2$

*B* cambia con T$_2$ / T$_1$

*¡Los casos son parecidos pero uno es secuenciable y el otro no!*

$A \leftarrow A - 100$
ESCRIBIR$(A)$

T$_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
ESCRIBIR$(A)$
LEER$(B)$
$B \leftarrow B + v$
ESCRIBIR$(B)$

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

*¿Qué tipo de conflicto hay aquí?*

¡Es secuenciable!

(No hay conflicto)

# (*) Conflictos parecidos con anulaciones

*Ejemplo 1*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

EJECUCIÓN

$T_1$

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$

$T_2$

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$

$\text{LEER}(B)$
$B \leftarrow B + 100$
ROLLBACK

$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$

*¿Cuánto es A + B después?*

270 + 230 = 500

# (*) Conflictos parecidos con anulaciones

*Ejemplo 1*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

- En una **planificación recuperable**, cada transacción T se compromete (COMMIT) solo después de que se comprometan a su vez todas las transacciones desde las cuáles T haya leído algo.

- Si cada transacción T solamente lee cambios de transacciones comprometidas, **se pueden evitar anulaciones en cascada** y mantener una planificación recuperable. De lo contrario, si T lee algo de una transacción no comprometida T’ y hay que anular T’, entonces puede ser que haya que anular T también (una anulación en casada) para mantener una planificación recuperable.

**Para cumplir con ACID, un sistema de bases de datos debe garantizar que solamente se permitan planificaciones recuperables.**


# SOLUCIÓN: BLOQUEOS (*LOCKS*)

# El problema

$T_1$

$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$

$T_2$

$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A + v$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$w \leftarrow B \times 0{,}1$
$B \leftarrow B + w$
$\text{ESCRIBIR}(B)$

$T_1$ (continúa)

$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$

Conflicto de Escritura–Lectura:

*¿Han visto un problema así antes de este curso?*

Multihilos (*Multithreading*)

*¿Y qué era la solución (en Java por ejemplo)?*

Bloques sincronizados

# Una solución en Java

Conflicto de Escritura–Lectura:

$T_1$

LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$

$T_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A + v$
ESCRIBIR$(A)$
LEER$(B)$
$w \leftarrow B \times 0{,}1$
$B \leftarrow B + w$
ESCRIBIR$(B)$

$T_1$

LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

```
public void trans(Cuenta A, Cuenta B, long val){
  synchronized(this.tablaCuentas) {
    long aAntiguo = tablaCuentas.leer(A);
    long aNuevo = aAntiguo - val;
    tablaCuentas.escribir(A,aNuevo);
    long bAntiguo = tablaCuentas.leer(B);
    long bNuevo = bAntiguo + val;
    tablaCuentas.escribir(B,bNuevo);
  }
}
```

```
public void interes(Cuenta[] Cs, float tasa){
  synchronized(this.tablaCuentas) {
    for(Cuenta C:Cs) {
      long cAntiguo = tablaCuentas.leer(C);
      long cNuevo = Math.round(cAntiguo*(1+tasa));
      tablaCuentas.escribir(C,cNuevo);
    }
  }
}
```

# Una solución en Java

*¿Cómo podrías mejorar el rendimiento del código abajo?*

Minimizar el código sincronizado tanto como sea posible

Usar objetos de bloqueos tan especifico que sea posible

(por ejemplo, usar un valor de la tabla o una fila de la tabla como el objeto de bloqueo, no la entera tabla, si es posible)

**¡Los mismos conceptos aplican a sistemas de bases de datos!**

Salvo que es el sistema, no el programador, que tiene que decidir (automáticamente) el nivel de sincronización, el nivel de bloqueo, etc.

Una transacción tiene que conseguir **bloqueos** para los objetos (valores, filas, vista, tablas) que quiere modificar.

Cuando haya terminado con el bloque, lo libera.

Un **protocolo de bloqueo** especifica las reglas que las transacciones tienen que seguir con respecto a bloqueos.

# Un protocolo: Bloqueo en 2 fases

- Si una transacción T quiere leer un objeto O, tiene que conseguir un bloqueo compartido sobre O
  - Varias transacciones pueden leer el mismo objeto al mismo tiempo
- Si una transacción T quiere modificar un objeto O, tiene que conseguir un bloqueo exclusivo sobre O
  - Un bloqueo exclusivo excluye bloqueos compartidos
  - T puede leer O (por supuesto)
  - Así nadie (aparte de T) puede ni leer ni modificar O mientras T tenga su bloqueo exclusivo sobre O

# Conflicto de Lectura–Escritura

*Ejemplo 1*: Dos transferencias bancarias entre cuentas $A$ y $B$
(donde el valor $A + B$ no debe cambiar, $A$ inicia con 400, $B$ con 200)

EJECUCIÓN

$T_1$

LEER$(A)$
$A \leftarrow A - 100$

$T_2$

LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
ESCRIBIR$(A)$

$T_1$

ESCRIBIR$(A)$
LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$

$T_2$

LEER$(B)$
$B \leftarrow B + v$
ESCRIBIR$(B)$

*¿Cuánto es A + B después?*

300 + 340 = 640

# … con bloqueos

| $T_1$ | $T_2$ |
|---|---|
| $\text{EXC}(A)$ | |
| $\text{LEER}(A)$ | |
| $A \leftarrow A - 100$ | |
| | $\text{EXC}(A)$ |
| | $\text{LEER}(A)$ |
| | $v \leftarrow A \times 0{,}1$ |
| | $A \leftarrow A - v$ |
| | $\text{ESCRIBIR}(A)$ |
| | $\text{REL}(A)$ |
| $\text{ESCRIBIR}(A)$ | |
| $\text{REL}(A)$ | |
| $\text{EXC}(B)$ | |
| $\text{LEER}(B)$ | |
| $B \leftarrow B + 100$ | |
| $\text{ESCRIBIR}(B)$ | |
| $\text{REL}(B)$ | |
| | $\text{EXC}(B)$ |
| | $\text{LEER}(B)$ |
| | $B \leftarrow B + v$ |
| | $\text{ESCRIBIR}(B)$ |
| | $\text{REL}(B)$ |

EJECUCIÓN

*¿Cómo sería la planificación final?*

# … con bloqueos

$T_1$

$\text{EXC}(A)$
$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{REL}(A)$

$T_2$

$\text{EXC}(A)$
$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$
$\text{REL}(A)$

$T_1$

$\text{EXC}(B)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$
$\text{REL}(B)$

$T_2$

$\text{EXC}(B)$
$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$
$\text{REL}(B)$

EJECUCIÓN

*¿A. P. A.?*

Todavía permite conflictos …

# … conflicto

**$T_1$**

$\text{EXC}(A)$
$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{REL}(A)$

EJECUCIÓN

**$T_2$**

$\text{EXC}(A)$
$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$
$\text{REL}(A)$
$\text{EXC}(B)$
$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$
$\text{REL}(B)$

*¿Qué tipo de conflicto hay?*

¡Escritura-lectura!

$\text{EXC}(B)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$
$\text{REL}(B)$

*¿Una solución?*

…

# … bloqueo estricto

Solución: Liberar los bloqueos solo cuando la transacción haya terminado (bloqueo estricto) …

$T_1$

$\text{EXC}(A)$
$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$
$\text{EXC}(B)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$
$\text{REL}(B)$
$\text{REL}(A)$

EJECUCIÓN

$T_2$

$\text{EXC}(A)$
$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$
$\text{EXC}(B)$
$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$
$\text{REL}(B)$
$\text{REL}(A)$

# Un protocolo: Bloqueo en 2 fases estricto

- Si una transacción T quiere leer un objeto O, tiene que conseguir un bloqueo compartido sobre O
  - Varias transacciones pueden leer el mismo objeto al mismo tiempo
- Si una transacción T quiere modificar un objeto O, tiene que conseguir un bloqueo exclusivo sobre O
  - Un bloqueo exclusivo excluye bloqueos compartidos
  - T puede leer O (por supuesto)
  - Así nadie (aparte de T) puede ni leer ni modificar O mientras T tenga su bloqueo exclusivo sobre O
- (Solo) cuando T haya terminado, libará sus bloqueos

# Un protocolo: Bloqueo en 2 fases estricto

- Si una transacción T quiere leer un objeto O, tiene que conseguir un bloqueo compartido sobre O
  - Varias transacciones pueden leer el mismo objeto al mismo tiempo
- Si una transacción T quiere modificar un objeto O, tiene que conseguir un bloqueo exclusivo sobre O
  - Un bloqueo exclusivo excluye bloqueos compartidos
  - T puede leer O (por supuesto)
  - Así nadie (aparte de T) puede ni leer ni modificar O mientras T tenga su bloqueo exclusivo sobre O
- (Solo) cuando T haya terminado, libará sus bloqueos

La forma más popular de garantizar secuenciabilidad.

# … pero cuidado!

*¿A. P. A.?*

Hay que evitar interbloqueos (*deadlocks*)

$T_1$

$\text{EXC}(A)$
$\text{LEER}(A)$
$A \leftarrow A - 100$
$\text{ESCRIBIR}(A)$

$T_2$

$\text{EXC}(B)$
$\text{EXC}(A)$
$\text{LEER}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\text{ESCRIBIR}(A)$
$\text{LEER}(B)$
$B \leftarrow B + v$
$\text{ESCRIBIR}(B)$
$\text{REL}(A)$
$\text{REL}(B)$

$\text{EXC}(B)$
$\text{LEER}(B)$
$B \leftarrow B + 100$
$\text{ESCRIBIR}(B)$
$\text{REL}(B)$
$\text{REL}(A)$

EJECUCIÓN

Solución 1: Plazos para terminar o …

Solución 2: Detectar ciclos en transacciones esperando o …

# Bloqueo en 2 fases conservativo

Solución 3: La transacción adquieren todos los bloqueos necesarios al inicio y de una forma atómica …

$T_1$

EXC$(A, B)$
LEER$(A)$
$A \leftarrow A - 100$
ESCRIBIR$(A)$
LEER$(B)$
$B \leftarrow B + 100$
ESCRIBIR$(B)$
REL$(A,B)$

EJECUCIÓN

$T_2$

EXC$(A, B)$
LEER$(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
ESCRIBIR$(A)$
REL$(A)$
LEER$(B)$
$B \leftarrow B + v$
ESCRIBIR$(B)$
REL$(B)$

*¿Es estricto?*

No …

# Bloqueo en 2 fases estricto/conservativo

Solo se pueden paralelizar transacciones que no compartan objetos

(lo más "fácil" pero lo menos paralelizable)

T$_1$

$\textsc{exc}(A, B)$
$\textsc{leer}(A)$
$A \leftarrow A - 100$
$\textsc{escribir}(A)$
$\textsc{leer}(B)$
$B \leftarrow B + 100$
$\textsc{escribir}(B)$
$\textsc{rel}(A,B)$

EJECUCIÓN

T$_2$

$\textsc{exc}(A, B)$
$\textsc{leer}(A)$
$v \leftarrow A \times 0{,}1$
$A \leftarrow A - v$
$\textsc{escribir}(A)$
$\textsc{leer}(B)$
$B \leftarrow B + v$
$\textsc{escribir}(B)$
$\textsc{rel}(A, B)$



# Pero para un usuario …

Hemos terminado con bases de datos relacionales

# Preguntas?