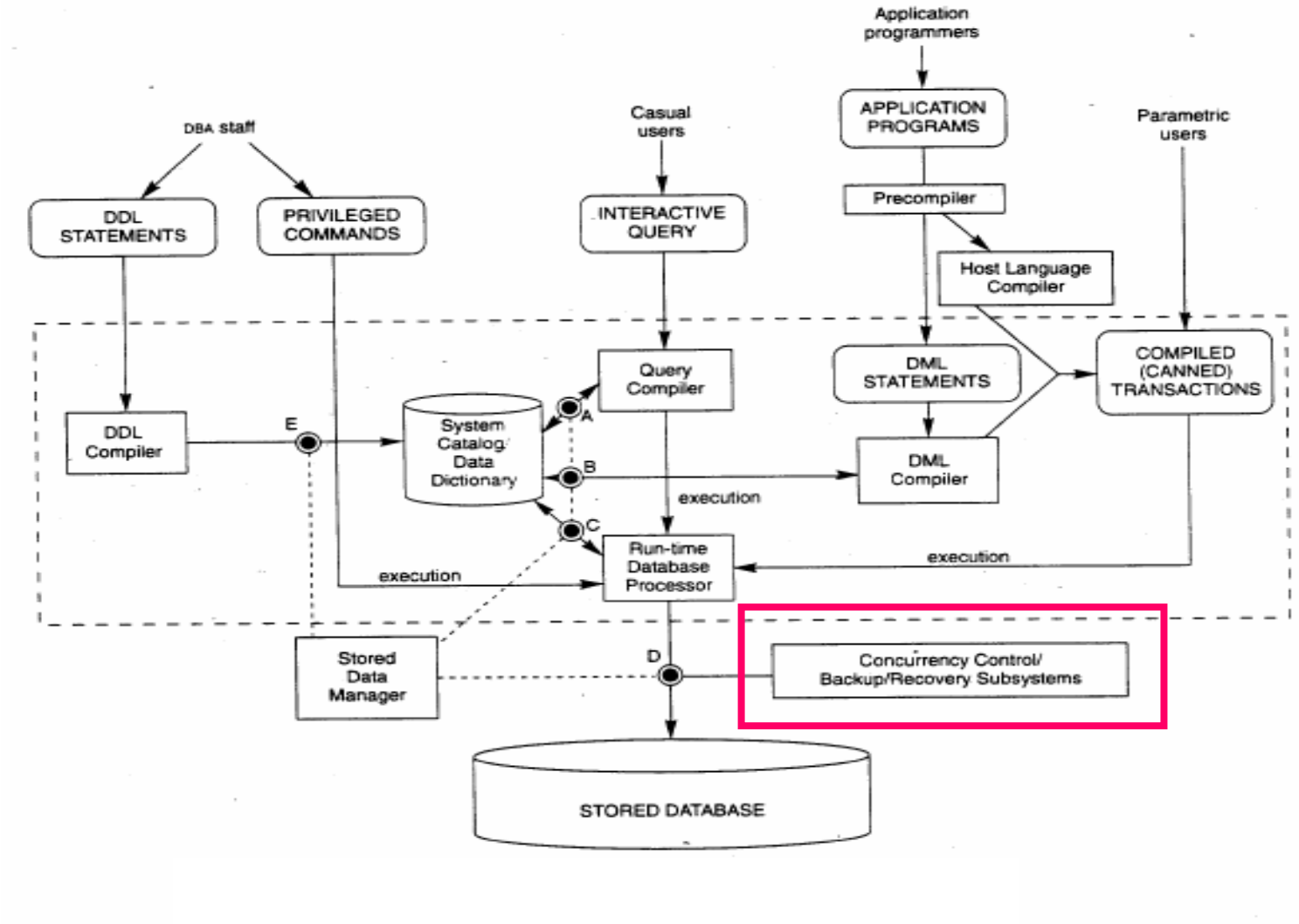
# Control de Concurrencia


Application programmers
APPLICATION PROGRAMS
Casual users
Parametric users
DBA staff
DDL STATEMENTS
PRIVILEGED COMMANDS
INTERACTIVE QUERY
Precompiler
Host Language Compiler
Query Compiler
DML STATEMENTS
COMPILED (CANNED) TRANSACTIONS
DDL Compiler
E
System Catalog Data Dictionary
A
B
C
DML Compiler
execution
execution
execution
Run-time Database Processor
Stored Data Manager
D
Concurrency Control/ Backup/Recovery Subsystems
STORED DATABASE

# Control de Concurrencia

Todo sistema de bases de datos debe brindar la posibilidad de que varios usuarios puedan acceder la data de forma simultánea. Cada usuario ejecuta su programa como su fuera el único que consulta la base de datos, es decir, que la concurrencia de otros programas de otros usuarios es transparente.

En los ambientes donde ocurre la concurrencia de varios usuarios a la base de datos surgen problemas cuando dos o más usuarios se interesan por el mismo objeto a la vez, llámese a éste: relación, tupla, atributo. Un SGBD debe ser capaz de controlar los conflictos que esta situación produce con la finalidad de mantener la consistencia e integridad de la data.

# DOS TÉRMINOS BÁSICOS PARA ABORDAR EL ESTUDIO DE LA CONCURRENCIA:

## Transacción e Itinerario

TRANSACCIÓN: se define como la ejecución de un programa que accesa la base de datos que es compartida por varios usuarios en forma simultánea.

Formalmente se define como una sucesión finita de operaciones:

**Oi (O1 → O2 → .... → On)**

que se realizan sobre un conjunto de objetos de la base de datos.

DOS TÉRMINOS BÁSICOS PARA ABORDAR EL ESTUDIO DE LA CONCURRENCIA: Transacción e Itinerario

**OPERACIONES EN UNA TRANSACCIÓN:**

| | | |
|---|---|---|
| Begin transaction | : | inicio de la transacción |
| Read a | : | lectura del objeto a |
| Write a | : | escritura del objeto a |
| Rollback | : | anulación de la transacción |
| Commit | : | fin de la transacción |

DOS TÉRMINOS BÁSICOS PARA ABORDAR EL ESTUDIO DE LA CONCURRENCIA: Transacción e Itinerario

De manera que el cuerpo de una transacción puede verse como:

**OPERACIONES EN UNA TRANSACCIÓN:**

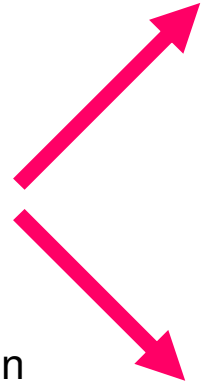
Begin transaction : inicio de la transacción

Read a : lectura del objeto a

Write a : escritura del objeto a

Rollback : anulación de la transacción

Commit : fin de la transacción



**Begin transaction T**
**O1**
**O2**
**.**
**.**
**.**
**On**
**Commit T.**

**Begin transaction T**
**O1**
**O2**
**.**
**.**
**.**
**On**
**Rollback.**

DOS TÉRMINOS BÁSICOS PARA ABORDAR EL ESTUDIO DE LA CONCURRENCIA: Transacción e Itinerario

ITINERARIO: es un conjunto de transacciones que se realizan en forma concurrente. Definido formalmente sería:

Un itinerario I sobre n transacciones T1, T2,...,Tn es una sucesión de operaciones:

$$I = ( x1\ y1\ ....\ u1)\ (x2\ y2\ ...\ u2)\ ....\ (xp\ yp\ ...\ up)$$

En donde:

xi es un conjunto de 0 o más operaciones de T1
yi es un conjunto de 0 o más operaciones de T2
...
ui es un conjunto de 0 o más operaciones de Tn

## PROPIEDADES DE LAS TRANSACCIONES

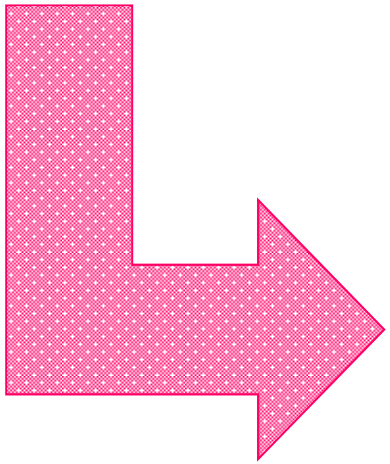
- Atomicidad: una transacción T es una unidad atómica de ejecución, es decir, o se ejecuta toda o no se ejecuta nada. Dicho de otra manera, los efectos de la T son reflejados en la BD o son completamente ignorados.
- Consistencia: los efectos de una transacción T trasladan la BD de un estado consistente a otro estado consistente. Una BD es consistente si satisface todas las consideraciones iniciales de estructura e integridad.
- Independencia: una transacción T se realiza como si fuera la única en hacerlo.
- Durabilidad: los efectos de una transacción T que ha alcanzado su punto de validación (ha llegado al ´COMMIT´) deben ser permanentes en la BD.

**CLIENTE**

| c.i. | nombre | monto |
|---|---|---|
| 111 | María | 500.000 |
| 222 | Pedro | 1.000.000 |
| 333 | Jesús | 700.000 |

**Se desea trasladar un 20% (10% y 10%) de los ahorros de Pedro a María y a Jesús.**

**Begin transaction** Transferencia
read a
read b
read c
a ← a + b*0.1
c ← c + b*0.1
b ← b - b*0.2
write a
write b
write c
**Commit** Transferencia



**CLIENTE**

| c.i. | nombre | monto |
|---|---|---|
| 111 | María | 600.000 |
| 222 | Pedro | 800.000 |
| 333 | Jesús | 800.000 |

## CONFLICTOS ENTRE TRANSACCIONES CONCURRENTES

Cuando se realiza una transacción T ésta puede interesarse en un objeto de dos maneras: para leerlo o para escribirlo. Por otro lado, si dos transacciones se interesan en forma simultánea por un objeto a, se tendrán cuatro posibilidades:

1. read1 a, read2 a
2. write1 a, write2 a
3. write1 a, read2 a
4. read1 a, write2 a

# CONFLICTOS ENTRE TRANSACCIONES CONCURRENTES

**1. Lectura-Lectura**

En este caso no se presentan conflictos, ya que el valor de a sólo será leído por ambas transacciones. Tampoco importa el orden en el que se lean, es igual "read1 a, read2 a" que "read2 a, read1 a".

# CONFLICTOS ENTRE TRANSACCIONES CONCURRENTES

**1. Lectura-Lectura**

En este caso no se presentan conflictos, ya que el valor de a sólo será leído por ambas transacciones. Tampoco importa el orden en el que se lean, es igual "read1 a, read2 a" que "read2 a, read1 a".

**2. Escritura-Escritura**

Este conflicto ocurre cuando dos transacciones T1 y T2 hacen una escritura sobre el mismo objeto. En este caso se puede perder la actualización hecha por alguna de las transacciones.

# CONFLICTOS ENTRE TRANSACCIONES CONCURRENTES

**1. Lectura-Lectura**

En este caso no se presentan conflictos, ya que el valor de a sólo será leído por ambas transacciones. Tampoco importa el orden en el que se lean, es igual "read1 a, read2 a" que "read2 a, read1 a".

**2. Escritura-Escritura**

Este conflicto ocurre cuando dos transacciones T1 y T2 hacen una escritura sobre el mismo objeto. En este caso se puede perder la actualización hecha por alguna de las transacciones.

**3. Escritura-Lectura**

Es cuando una transacción T1 se interesa en escribir sobre un objeto en el que T2 está interesada en leer. El problema ocurre cuando T1 actualiza el objeto y luego aborta (no llega al ´COMMIT´). En este caso se puede presentar una lectura indebida del objeto.

# CONFLICTOS ENTRE TRANSACCIONES CONCURRENTES

**1. Lectura-Lectura**

En este caso no se presentan conflictos, ya que el valor de a sólo será leído por ambas transacciones. Tampoco importa el orden en el que se lean, es igual "read1 a, read2 a" que "read2 a, read1 a".

**2. Escritura-Escritura**

Este conflicto ocurre cuando dos transacciones T1 y T2 hacen una escritura sobre el mismo objeto. En este caso se puede perder la actualización hecha por alguna de las transacciones.

**3. Escritura-Lectura**

Es cuando una transacción T1 se interesa en escribir sobre un objeto en el que T2 está interesada en leer. El problema ocurre cuando T1 actualiza el objeto y luego aborta (no llega al ´COMMIT´). En este caso se puede presentar una lectura indebida del objeto.

**4. Lectura-Escritura**

Una transacción T1 se interesa en leer un objeto y una transacción T2 se interesa en escribir sobre el mismo. Si T1 hace dos lecturas, ya que ella no ha escrito sobre el objeto, esperaría encontrar el mismo valor. Sin embargo, este sería diferente si T2 ha escrito sobre el objeto antes de la segunda lectura de T1.

## Ejemplo: sean las siguientes transacciones

T1: read a
write a

T2: read b
write b

entonces, para este caso, se tienen seis posibles itinerarios para las transacciones T1 y T2, que son:

| I1 | I2 | I3 | I4 | I5 | I6 |
|---|---|---|---|---|---|
| read a | read b | read a | read b | read b | read a |
| write a | write b | read b | read a | read a | read b |
| read b | read a | write a | write a | write b | write b |
| write b | write a | write b | write b | write a | write a |

| I1 | I2 | I3 | I4 | I5 | I6 |
|---|---|---|---|---|---|
| read a | read b | read a | read b | read b | read a |
| write a | write b | read b | read a | read a | read b |
| read b | read a | write a | write a | write b | write b |
| write b | write a | write b | write b | write a | write a |

Los itinerarios I1 y I2 son secuenciales, mientras que los demás no lo son.

Por otra parte, es evidente que los itinerarios secuenciales darán resultados consistentes, tanto I1 como I2 darán los mismos resultados.

A su vez, dentro de los no secuenciales, algunos itinerarios son serializables o son consistentes. Por consistentes se entiende a aquellos itinerarios que no siendo secuenciales son equivalentes a uno secuencial pues dan los mismos resultados.

Supongamos el siguiente caso: se tienen dos transacciones que descuentan 1000 unidades de un objeto y lo adicionan a otro.

Las transacciones son:

```
Begin transaction T1
  read a
  read b
  a=a-1000
  b=b+1000
  write a
  write b
Commit T1;
```

```
Begin transaction T2
  read c
  read b
  c=c-1000
  b=b+1000
  write c
  write b
Commit T2;
```

Digamos que existen tres Itinerarios: I1, I2 e I3:

| I1 | Valor Real | I2 | Valor Real | I3 | Valor Real |
|---|---|---|---|---|---|
| T1: Begin | | T1: Begin | | T1: Begin | |
| T1: read a | a=2000 | T1: read a | a=2000 | T1: read a | a=2000 |
| T1: read b | b=3000 | T2: Begin | | T2 : Begin | |
| T1: a=a-1000 | | T2 : read b | b=3000 | T1 : read b | b=3000 |
| T1 : b=b+1000 | | T2 : read c | c=4000 | T2 : read c | c=4000 |
| T1 : write a | a=1000 | T1 : a=a-1000 | | T1 : a=a-1000 | |
| T1 : write b | b=4000 | T1 : write a | a=1000 | T2 : read b | b =3000 |
| T1 : Commit | | T2 : b=b+1000 | | T1 : b=b+1000 | |
| T2 : Begin | | T2 : c=c-1000 | | T2 : c=c-1000 | |
| T2 : read c | c=4000 | T2 : write b | b=4000 | T1 : write a | a=1000 |
| T2 : read b | b=4000 | T2 : write c | c=3000 | T2: b =b+1000 | |
| T2 : c=c-1000 | | T1 read b | b=4000 | T1 : write b | b=4000 |
| T2 : b=b+1000 | | T2 : Commit | | T2 : write c | c=3000 |
| T2 : write c | c=3000 | T1 : b=b+1000 | | T1 : Commit | |
| T2. write b<br>T2 : Commit | b=5000 | T1 : write b<br>T1 : Commit | b=5000 | T2 : write b<br>T2 : Commit | b=4000 |

En este caso I1 es un itinerario secuencial pues primero se realiza T1 y luego T2, por lo tanto es consistente. Por otro lado, I2 no es secuencial pero sí es consistente pues da lo mismos resultados que I1. El itinerario I3 no es ni secuencial ni consistente, este itinerario no se debe ejecutar.

# Fuentes consultadas:

**[1] Prof. Elsa Liliana Tovar.**
Notas de clase compiladas.

**[2] Ramakrishnan Raghu.** ,
“Database Management Systems".

**[3] Navathe, Elsamri,**
“Fundamentals of DataBase System”.

**[4]** http://www.slideshare.net/koolkampus/ch15