BASES DE DATOS

Guión Tema 6:

Transacciones y Control de Concurrencia

Profesor : Héctor Gómez Gauchía

1. Transacciones

1.1 Definición de Transacción

Una transacción es la ejecución de un programa (o parte de él), que es visto como una unidad en sí. Hay dos tipos:

- Solo de lectura: leer(A)
- De actualización: leer(A), cambios(A), escribir(A)

En programas con SQL integrado, se marca por instrucciones de "inicio de T" y "fin de T

1.2 Problemas de la multiprogramación

DEF: Dos transacciones T_1 y T_2 son *concurrentes* cuando se ejecutan simultaneamente en una sola CPU, que solo puede ejecutar una por una.

TIPOS DE PROBLEMAS: actualización sin tener en cuenta el control de la concurrencia produce inconsistencia de datos. http://www.wisc.edu/drmt/oratips/sess001.html

a) Actualización perdida:

CPU ejecuta ⇩

T_1	T_2	
Leer(X)		
X := X - N		
	Leer(X)	
	X := X + M	
Escribir(X)		
Leer(Y)		
	Escribir(X)	→Error:se pierde el X de T_1
Y := Y + K		

b) Actualización temporal (Lectura sucia):

Leer(X)		
X := X - N		
Escribir(X)		
	Leer(X)	
	X := X + M	
	Escribir(X)	
Rollback(X de T_1)	…….sigue	
… sigue		Error:se pierde el X de T_2←

c) Resumen incorrecto:

	Suma := 0
	Leer(A)
	Suma := Suma + A
Leer(X)	
X := X - N	
Escribir(X)	
	Leer(X)
	Suma := Suma + X
	Leer(Y)
	Suma := Suma + Y
Leer(Y)	
Y := Y + M	
Escribir(Y)	

Bien:acumula X nuevo ←

Error: acumula Y viejo←

d) Lectura no repetible:

Leer(X)	
	Leer(X)
	Escribir(X + 1)
Leer(X)	

e) Lectura Fantasma: Una lectura de una fila que no existía cuando se inició la transacción

…hay muchos otros casos. Para evitar estas situaciones y conservar la integridad de datos necesitamos estas

1.3 Propiedades de las Transacciones: A.C.I.D.

1. *Atomicidad*: La transacción(T) funciona como unidad: o se ejecutan todas sus operaciones o ninguna. Se prohibe una ejecución parcial.

   Responsable: es el Método de Recuperación el que, si no puede completar todas las operaciones, devuelve la BD a su estado anterior a empezar esa T (rollback).

2. Conservar la *Consistencia* de datos una vez ejecutada la transacción.

   DEF: *Estado Consistente* de los datos en una BD es cuando satisfacen las restricciones definidas sobre ellos.

   Responsable: los programadores son responsables de la consistencia mediante la definición adecuada de: check, triggers, primary key, foreign key,…

3. *Aislamiento* entre T's: garantiza que una T en ejecución no permite acceso a sus actualizaciones antes de que termine (sea confirmada)

   Responsable: lo mantiene el Método de Concurrencia: mecanismos, reglas, protocolos

4. *Durabilidad* o permanencia de los datos después de terminar una T con éxito y que ningún error posterior provoque la pérdida de los datos actualizados por T.

   Responsable: la mantiene el Método o gestor de Recuperación.

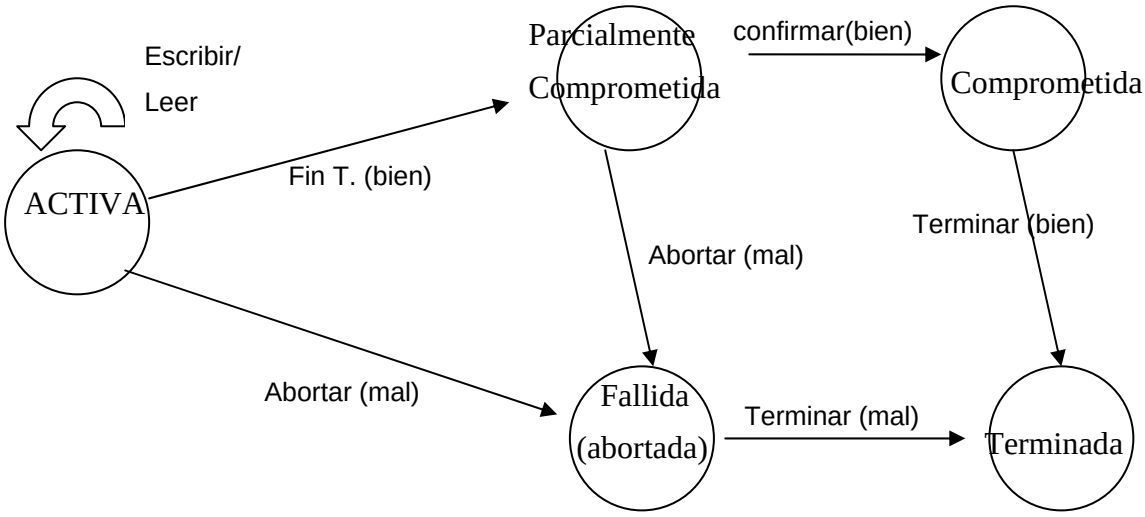
Comportamiento permitido

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura fantasma
Lectura no comprometida	Sí	Sí	Sí
Lectura comprometida	No	Sí	Sí
Lectura repetible	No	No	Sí
Secuenciable	No	No	No

SQL Server permite todos estos niveles, Oracle sólo permite la lectura comprometida y secuenciable. Los niveles se pueden establecer en ambos SGBD para cada transacción.

1.4 Estados de una Transacción

Diagrama de Estados



Comprometida = Confirmada. => Marca un "punto de control".

Fallos: caidas de sistema, físicos (eléctricos, disco), concurrencia(aborta T_1 por violación)

Estado "Fallida": retrocede la T, rollback. Si fallo hardware, reinicia. Si fallo lógico, termina.

`Savepoint NombrePunto`: para salvar puntos intermedios de una T.

`Rollback[to savepoint NombrePunto]` deja la BD como estaba antes del NombrePunto

2 Planes / Planificaciones

Para mantener esas propiedades se necesita saber en *qué* orden se ejecutan las transacciones concurrentes,… necesitamos un plan que lo decida.

2.1 Definiciones

-Un *plan/planificación* **P** de **n** transacciones $\mathbf{T_i}$, es la descripción del orden en que se ejecutan las operaciones de dichas $\mathbf{T_i}$, manteniendo, para cada $\mathbf{T_i}$, el orden original de sus operaciones.

-Un *plan secuencial* (o en serie) de un conjunto de **T** es aquel en el que todas las operaciones de cada $\mathbf{T_i}$ se ejecutan en secuencia sin que se intercale ninguna operación de otra $\mathbf{T_j}$.

-Todos los planes secuenciales son correctos porque garantizan las propiedades A.C.I.D.

-Los planes secuenciales pueden ser ineficientes (ver ejemplo pag. siguiente): conviene intercalar operaciones siempre que mantengan las propiedades ACID.

-Un *plan equivalente*: aquel que produce los mismos efectos en la BD.

-*Plan secuenciable* : aquel que es equivalente a otro secuencial (también es correcto)

EJEMPLO. Plan secuencial

- $\mathbf{T_2}$ pierde el tiempo esperando.

-*Orden del plan*: eje vertical

-solo se ejecuta una op./línea

-Dos op. tienen el mismo *orden* en dos planes si se ejecutan en el mismo orden en ambos planes.

$\mathbf{T_1}$	$\mathbf{T_2}$
Leer(A)	
Escribir(A)	
Leer(B)	
Escribir(B)	
	Leer(A)
	Escribir(A)
	Leer(B)
	Escribir(B)

-Dos ops. Seguidas de un plan estan en conflicto si:

1.- Son de diferentes T's

2.- Acceden al mismo elemento X (p.e.: un atrib. de una tabla)

3.- Al menos una de las ops es "escribir(X)".

…esto último obliga a que una T debe ejecutarse completamente antes que la otra.

2.2 Secuenciabilidad

Hay dos modos de que dos planes P_1 y P_2 sean equivalentes y secuenciables:

2.1.1 Equivalentes por conflictos:

Dos planes P_1 y P_2 son equivalentes *por conflictos* si, para todo par de ops *en conflicto*, ambas ops se ejecutan en el mismo orden en P_1 y P_2

→ Si dos ops (de dos Ts) no estan en conflicto dentro de un plan, se pueden cambiar de orden.

Esto permite conseguir planes equivalentes más eficientes (intercalando todo lo posible).

→ Un plan P es <u>secuenciable en cuanto a conflictos</u> si se puede encontrar un plan secuencial P' que sea equivalente por conflictos al P.

EJEMPLO:

Equivalente por conflictos al ejemplo anterior y más eficiente intercalando.

Leer(A)	
Escribir(A)	
	Leer(A)
Leer(B)	
	Escribir(A)
Escribir(B)	
	Leer(B)
	Escribir(B)

Más adelante usamos:

L_j se refiere a una Leer(Algo)

E_i se refiere a una Escribir(Algo)

<u>Prueba de secuenciabilidad: Grafo de Precedencia(GP)</u>

Dado un plan P , el GP es un grafo dirigido G=(N, A) con nodos N = { T_1 ,…, T_n } y arcos dirigidos A = { a_1 ,…, a_n } . Cada nodo es una T_i del plan. Cada arco a_i es T_j → T_k , donde se da $1 <= i <= n$, $1 <= j <= n$, y una de las ops en conflicto de T_j aparece antes en el plan de alguna op en conflicto de T_k

<u>Algoritmo para generar el GP</u>

1- Para cada T_i de P crear un nodo T_i
2- Para cada caso en el P que L_j (X) está después de E_i (X): crear un T_i → T_j
3- Para cada caso en el P que E_j (X) está después de L_i (X): crear un T_i → T_j
4- Para cada caso en el P que E_j (X) está después de E_i (X): crear un T_i → T_j
5- El plan P es secuenciable sii el GP *no tiene ciclos*

(es decir: se puede crear un P' secuencial equivalente por conflicto, donde T_i está antes que T_j)

EJ: hacer EJ-1,2 y 3: Son secuenciables estos planes?

Aplicabilidad:

Es dificil de comprobar a mano la secuenciabilidad porque intervienene otros factores (carga sistema, momento arranque de cada T, prioridades dadas a cada T, cuando empieza un plan)

Solución: Crear *protocolos* o conjuntos de reglas que, si todas las Ts las siguen, garantizan la secuenciabilidada de los planes hechos para esas Ts. S basa en la teoría de secuenciabilidad para el control de concurrencia (se ve más tarde).

2.1.2 Equivalentes por vistas

Es menos restrictiva que la anterior.

Dos planes P y P' son equivalentes si ambos tienen las mismas Ts y en los dos se cumplen las siguientes condiciones para los mismos *valores, operaciones y transacciones*:

1.- Si el valor leído por cada L_i (X) – de T_i – cumple que:

a) Fue producido por E_j (X), ó

b) X era el valor inicial del plan P,

entonces X ha de cumplir lo mismo, a) ó b), en P' (asegura que ambos Ps leen los mismos valores).

2.- La operación E_k (Y) es la última que escribe Y en el plan P también lo es en P' (asegura que ambos Ps dejan el mismo estado final en la BD).

EJ: hacer EJ-4

La equivalencia por conflictos y por vistas son similares cuando se cumple, en todas las transacciones, la siguiente condición:

Escritura restringida : Toda operación E_i (X) va precedida de una L_i (X) y el valor de E_i (X) solo depende del valor leído por L_i (X).

Si no sucede esto se habla de *escrituras ciegas* independientes a los valores anteriores en la BD, como en T_1 y T_3 del EJ-4

→ Un plan P es secuenciable en cuanto a vistas si se puede encontrar un plan secuencial P' que sea equivalente por vistas al P.

Prueba de secuenciabilidad por Vistas: Es secuenciable si el grafo es sin ciclos. Algoritmo:

Crear dos transacciones ficticias T_C (comienzo) y T_{FIN} , con 1ª op: E_C y última op.: L_F

1.- Añadir arco $T_i \rightarrow T_j$ de valor 0 si T_j lee un Q escrito por T_i

2.- Borrar arcos que inciden en T_{inutil} (no exite camino $T_{inutil} \rightarrow T_F$

3.- Para todo dato Q y arco generado en 1.-, examinar toda E_k (Q), donde k <> c

a)- Si es el comienzo (i = c, j <> fin), añade a continuación K : crea $T_j \rightarrow T_k$ de valor 0

b)- Si es el final (i <> c, j = fin), añade antes K : crea $T_k \rightarrow T_i$ de valor 0

c)- Si es en medio (i <> c, j <> fin), añade ambas : $T_k \rightarrow T_i$ y $T_j \rightarrow T_k$ de valor p

donde p es un entero único

Para crear el P' secuencial se quitan T_C y T_{FIN} y se sigue el grafo.

EJ-5

2.3 Recuperabilidad de los planes

DEF: *Recuperabilidad* : Un plan es *recuperable* si para todas sus T's que tienen L(X) no se confirman hasta que todas las Tx que tienen E(X) se han confirmado.

→ Toda transacción T puede terminar: Confirmandola (commit) o abortandola (abort) en caso de fallo, en cuyo caso hay que deshacer los cambios (rollback, retroceso) de la BD.

→ El fallo de una T puede ser en cualquier momento (EJ-6 fallo en cascada)

→ Lo importante es cuando hacer la confirmación (C) ? (EJ-7b)

→ En un plan recuperable *nunca* se deshace una T confirmada…pero puede que se haga…

Retroceso en cascada: Por el fallo de una sola T (T_{10}) puede que haya que retroceder varias T's

Solución: Toda T solo lee un elemento cuando fue escrito por una T ya confirmada. (EJ-6)

Plan estricto: Aquel en la que todas T no puede ni L(X) ni E(X) hasta que se confirme la última T que E(X). Así reducen la posible recuperación a la última imagen de X en la BD.

→ La garantía de consistencia en la BD la dan planes secuenciables por conflictos o por vistas y que sean recuperables, sin cascadas.

… Encontrar planes secuenciables puede ser Np-completo,… , además

… El usuario no sabe el orden en el que se ejecutan las Ts, así que, para tener control…

→ Para ser eficientes necesitan refinar la concurrencia mediante el control de concurrencia

… Si bloqueamos toda la BD seguro que termina consistente, pero…

3 Control de Concurrencia

- Otro modo, además de los planes secuenciables, de mantener el aislamiento al ejecutar las Ts.
- Se pueden basar en el mismo criterio de secuenciabilidad o en otros.
- Los sistemas de Control de Concurrencia (CC) se basan en *protocolos*: conjuntos de reglas.

3.1 Protocolos basados en el bloqueo (Locks, BL)

- Asegurar la secuenciabilidad: acceso mutuamente excluyente a los datos
- Se hace poniendo bloqueos al elemento de datos accedido.
- Dos modos de bloqueo (BL): Dada una T_i que va a acceder al elemento Q
  a)- Compartido (ó de lectura): BC_i (Q), permite a T_i leer Q, pero no escribir Q
  b)- Exclusivo (ó de escritura): BX_i (Q), permite a T_i leer Q y escribir Q
- Cada T_i debe "solicitar" el BC ó BX y queda en espera hasta que el CC se lo conceda
- La función de compatibilidad entre bloqueos es esta:

T_2 PIDE \ T_1 TIENE	**BC**	**BX**
BC	SI	NO
BX	NO	NO

Que pasa si Q tiene un BL de un tipo concedido a T_1 y T_2 pide otro BL? Se le concede?:

- Se permiten varios BC, solo un BX

En los ejemplos incluimos tres operaciones: BC, BX y DES (desbloquear) EJ-8

Dos acciones: promover ó subir (pasar un BC a BX) y degradar o bajar (pasar un BX a BC)

Protocolo de bloqueo en dos fases básico (B2F): oracle

Una T sigue el B2F si todas las ops de bloqueo (BC, BX) preceden a la 1ª op de desbloqueo de T

Es decir, que se pueden dividir en dos fases: EJ-9

- Fase de expansión (crecimiento): adquiere nuevos BLs, no libera BLs
- Fase de contracción (decrecimiento): libera BLs, no puede adquirir BLs nuevos

→ Si todas las T_i de un plan siguen B2F, el plan es secuenciable respecto a conflictos.

Problema: limitan mucho el acceso… Mejoras con

Protocolo de bloqueo estricto (BE2F): Todas las T_i de un plan mantienen todos los BX hasta comprometerse (ó abortar). El más usado.

Protocolo de bloqueo riguroso (BR2F): Todas las T_i de un plan mantienen todos los BX y BC hasta comprometerse (ó abortar).

Protocolo de bloqueo conservador o estático (BC2F): Todas las T_i de un plan bloquean todos los BX y BC antes de empezar a ejecutarse. Y si le falta alguno espera. Evita el bloqueo mortal (deadlock)

EJ-10

→ Podemos hacer protocolos en una sola fase… Usando más información: orden fijo de las Ts

3.2 Protocolos basados en Marcas Temporales (timestamp)

- Basados en predeterminar el orden de ejecución de las Ts

- Una *marca temporal* MT(T_i) es un identificador único que el SGBD asigna a cada T_i al entrar en el sistema, bien con la hora de entrada o con un contador secuencial.

- A cada elemento X de la BD se le asigna dos valores de MT:

  - MTL(X): La mayor MT de todas la T_i que han leido con éxito X hasta ahora.

  - MTE(X): La mayor MT de todas la T_i que han escrito con éxito X hasta ahora.

Algoritmo de ordenamiento básico por marcas de tiempo (OMT):

Comprueba el orden de las operaciones para cada petición de un $L_i(Q)$ o $E_i(Q)$ se compara con el MTL(Q) ó MTE(Q) : si es correcto ejecuta y si no aborta T_i y todas las T_j que hayan *leido algo escrito por T_i* (retroceso en cascada). Situaciones que se pueden dar:

1.- T_i que intenta $L_i(Q)$ (leer Q)

a).- Si $MT(T_i) < MTE(Q)$: rechazar $L_i(Q)$ y retroceder T_i

intento de leer algo escrito por una T posterior

b).- Si $MT(T_i) >= MTE(Q)$: ejecutar $L_i(Q)$ y asignar nueva MTL(Q): max (MTL(Q),$MT(T_i)$)

2.- T_i que intenta $E_i(Q)$ (escribir Q)

a).- Si $MT(T_i) < MTL(Q)$: rechazar $E_i(Q)$ y retroceder T_i

intento de escribir algo sobre algo que ha leído ya una T posterior

b).- Si $MT(T_i) < MTE(Q)$: rechazar $E_i(Q)$ y retroceder T_i

intento de escribir algo sobre algo que otra T posterior ha escrito

c).- En caso contrario: ejecutar $E_i(Q)$ y asignar nueva MTE(Q) con el valor de $MT(T_i)$

Si T_i ha sido retrocedida: se aborta y se inicia de nuevo con MT nueva que será mayor que todas (incremento secuencial) EJ-11

Características del OMT:

-Asegura secuenciabilidad según conflictos

-Asegura que no hay bloqueos (interbloqueos) mortales –deadlocks–

-Pero se puede dar *reinicio cíclico* (inanición)

-Pero no garantiza que sean recuperables: confirma T_i de $L_i(Q)$ después de confirmar T_k de $E_k(Q)$

-Entonces…genera retrocesos en cascada.

OMT Modificado: Regla de Escritura de Thomas:

-Mejora concurrencia evitando algunos retrocesos(rechaza menos operaciones)

-No impone secuenciabilidad por conflicto

-Se basa en ignorar una $E_i(Q)$ si viene después de otra $E_j(Q)$ sin lectura intermedia: cambia "2.-"

2.-NUEVO T_i que intenta $E_i(Q)$ (escribir Q)

a).- Si $MT(T_i) < MTL(Q)$: rechazar $E_i(Q)$ y retroceder T_i

b).- Si $MT(T_i) < MTE(Q)$*: ignorar $E_i(Q)$ --no hacer $E_i(Q)$-- y seguir el plan con resto de T_i*

c).- En caso contrario: ejecutar $E_i(Q)$ y asignar nueva MTE(Q) con el valor de $MT(T_i)$

3.3 Otras cuestiones en el Control de Concurrencia

Bloqueos mortales o interbloqueos: (deadlock)

Un sistema está en estado de interbloqueo cuando toda T_i del conjunto de T's que están en ejecución, está esperando por algún recurso que tiene bloqueado otra T_i.

Suceden cuanto más alto es el grado de Aislamiento.

Hay dos estrategias: prevenirlos o detectarlos+recuperación.

Prevención:

1.- Usando bloqueos solamente:

- BSF conservador: o todos los elementos bloqueados o ninguno
  - Dificil predecir qué bloqueos necesito antes de empezar
  - Se mantienen bloqueos innecesarios
- Fijar un orden parcial preestablecido de ejecución de las T's: protocolo de árbol
  - Se necesita conocer el orden para programar: dificil.

2.- Usando MTs y bloqueos: cada T_i. se pone su MT, y el CC usa bloqueos

Dos estrategias para decidir quien tiene que esperar o retroceder si T_i solicita un elemento que tiene bloqueado T_j :

- Esperar-morir: - T_i espera solo si $MT(T_i) < MT(T_j)$ -- T_i anterior a T_j –
  - En otro caso T_i muere
- Herir-esperar: - T_i espera solo si $MT(T_i) > MT(T_j)$
  - En otro caso T_j se retrocede --T_i "hiere" a T_j --
    y T_j expropia el elemento en conflicto

No se producen esperas indefinidas (inanición):si T_i retrocede conserva su MT antigua

Diferencias de funcionamiento en las dos estrategias:

- Esperar-morir: -Cuanto más antigua es T, más espera
  -La T_i que muere, puede morir varias veces si el recurso sigue bloqueado
- Herir-esperar: -Una T antigua no espera a una T reciente
  -Entre T_i y T_j se alternan la espera.

3.4 Control de Concurrencia en Oracle

Lectura consistente (L.C.): Durante la ejecución de toda la T, las otras T's ven el contenido de la BD congelado, con los valores que tenía al comienzo de la T, para evitar el problema del *resumen incorrecto* –ver el caso c) de sección 1.2

- A nivel de transacción: `SET TRANSACTON READ ONLY;` esa T solo lee, no modifica, agiliza BD

  Solo ve las actualizaciones que estaban confirmadas cuando empezó T.

- A nivel instrucción: (por defecto en oracle) `SET TRANSACTON READ WRITE;`

Consistencia de Lectura Tres situaciones:

1.- L.C**. i**mplícita (nivel instrucción): se mantiene dentro de la instrucción `SELECT…`

2.- Sin L.C.: `SELECT…` (se ve y modifica entre las dos consultas)
`SELECT…`

3.- L.C. explícita : `COMMIT;` (inicia T)

```
SET TRANSACTON READ ONLY; (debe ser 1ª instr en T)
SELECT count (*) from cliente;
SELECT count (*) from invierte;
COMMIT;                    (termina la T de read only)
```

Control de Concurrencia: AISLAMIENTO automático (iso/ansi sql3)

Manejo de las actualizaciones evitando interferencias entre T's

```
SET TRANSACTON ISOLATION LEVEL [SERIAZABLE | READ COMMITED];
```

Dos Modos:

- `SERIAZABLE` : *secuenciable*. T's no pierden actualizaciones --ver a) en sección 1.2--, se garantizan lecturas repetibles --ver d) en sección 1.2--, no hay registros fantasma y tampoco resumen incorrecto.

  Porque las modificaciones hechas por T solo la ve T.

  Si T_i actualiza algún recurso que actualizó T_j y está sin confirmar, entonces T_i aborta.

- `READ COMMITED` : (por defecto) Lectura Consistente. La T no tiene lecturas repetibles. Modificaciones hechas por T y otras T's son visibles por T y por otras T's, solo si las otras han hecho commit.

  Si T_i tiene DML que necesita bloquear filas que tiene otra T, la T_i espera.

```
Explicación de niveles de aislamiento en Oracle:
http://www.oracle.com/technology/oramag/oracle/05-nov/o65asktom.html
```

… Este recurso puede relentizar las transacciones, por eso se usan *bloqueos explícitos*.

Bloqueos de datos (locks)

-Garantizan consistencia de datos (no permite cambios por otras T's mientras se lee/actualiza)

-Garantiza integridad (datos y estructuras correctas)

-Se mantienen hasta un commit/rollback o desbloqueo explícito

-Usos:

-Para controlar los accesos y actualizaciones concurrentes

-Reservar un tabla para toda la transacción

-Lecturas repetidas en bucles.

Bloqueos automáticos: los pone el SGBD a nivel de fila

Bloqueos explícitos: los pone el programador

```
LOCK TABLE mitabla IN XXXX MODE [NOWAIT] ;
```

XXXX puede ser:

`SHARE (S)` Lectura concurrente . Permite otros locks SHARE

`EXCLUSIVE (X)` No permite ningún otro lock.

EJ-12

Atomicidad: oracle usa un protocolo en dos fases

Lectura Fantasma: La T_1 crea un registro en la BD de un elemento creado por otra T_2 (está acumulando). Pero el orden es $MT(T_1) > MT(T_2)$

http://www.itu.dk/people/pagh/IDB05/Transactions-examples-run.html

User pagh

```
SQL> SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE;

Transaction set.

SQL> CREATE TABLE Primes (p INT);

Table created.

SQL> GRANT SELECT, UPDATE, INSERT ON
Primes to pagh2;

Grant succeeded.

SQL> SELECT * FROM Primes;

no rows selected
```

User pagh2

```
SQL> SET TRANSACTION ISOLATION LEVEL
SERIALIZABLE;

Transaction set.
```

```
SQL> SELECT * FROM pagh.Primes;

no rows selected
```

```
SQL> INSERT INTO Primes VALUES (41);

1 row created.
```

```
SQL> INSERT INTO pagh.Primes VALUES
(43);

1 row created.
```

```
SQL> SELECT * FROM Primes;

         P
----------
        41
```

```
SQL> SELECT * FROM pagh.Primes;

         P
----------
        43
```

```
SQL> COMMIT;

Commit complete.
```

```
SQL> SELECT * FROM pagh.Primes;

         P
----------
        43
```

```
SQL> COMMIT;

Commit complete.
```

```
SQL> SELECT * FROM pagh.Primes;

         P
----------
        41
        43
```

```
SQL> SELECT * FROM pagh.Primes;

         P
----------
        41
        43
```

```
SQL> SET TRANSACTION ISOLATION LEVEL
READ COMMITTED;

Transaction set.

SQL>  INSERT INTO Primes VALUES (2);

1 row created.
```

```
SQL> SET TRANSACTION ISOLATION LEVEL
READ COMMITTED;

Transaction set.
```

```
SQL> SELECT * FROM pagh.Primes;

         P
----------
        41
        43

SQL> INSERT INTO pagh.Primes VALUES
(2003);

1 row created.

SQL> SELECT * FROM pagh.Primes;

         P
----------
        41
        43
      2003
```

```
SQL> COMMIT;

Commit complete.
```

```
SQL> SELECT * FROM pagh.Primes;

         P
----------
        41
        43
         2
      2003
```

```
SQL> SELECT * FROM Primes;
```

```
         P
----------
        41
        43
         2
                                        SQL> COMMIT;

                                        Commit complete.

SQL> SELECT * FROM Primes;

         P
----------
        41
        43
         2
      2003

SQL> INSERT INTO Primes VALUES (3);

1 row created.

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT * FROM Primes;

         P
----------
        41
        43
         2
      2003
```

User pagh

```
SQL> CREATE TABLE doedlaas (
   pk INT PRIMARY KEY
);

Table created.

SQL> INSERT INTO doedlaas VALUES
(1);

1 row created.
```

```
commit;
```

```
SQL> INSERT INTO doedlaas VALUES
(2);

1 row created.
```

User pagh, transaction 2

```
                                        INSERT INTO doedlaas VALUES (1);


                                        INSERT INTO doedlaas VALUES (1)
                                        *
                                        ERROR at line 1:
                                        ORA-00001: unique constraint
                                        (PAGH.SYS_C0030509) violated
```

```
                                    SQL> INSERT INTO doedlaas VALUES (3);

                                    1 row created.

                                    SQL> INSERT INTO doedlaas VALUES (2);



SQL> INSERT INTO doedlaas VALUES
(3);

INSERT INTO doedlaas VALUES (3)
            *
ERROR at line 1:
ORA-00060: deadlock detected while
waiting for resource

SQL> rollback;

Rollback complete.                  1 row created.
```