Transacciones y Control de Concurrencia

Prof.: Héctor Gómez Gauchía

CONSULTA LOS DEMÁS MATERIALES DEL CAMPUS VIRTUAL

Transacciones

- Una transacción (T) es un conjunto de una o varias instrucciones SQL
  - cuya ejecución constituye una unidad lógica e indivisible
  - Es decir, que se ejecutan todas o ninguna.

- Específicamente, una transacción es :
  - Una o más instrucciones DML y DDL
  - Y después: o no hay más instrucciones o hay una COMMIT (confirmar)

- Hay dos tipos de T's, según las **operaciones** (ops.) que ejecute:
  - Solo de lectura: leer(A)
  - De actualización: leer(A), cambios(A), escribir(A)

Control de Concurrencia y Consistencia de Datos

- En un entorno multiproceso se ejecutan varias T's simultáneamente
- El Control de Concurrencia es la función del SGBD que
  - permite gestionar los cambios realizados por instrucciones SQL y
  - su agrupación en transacciones.

- Permite a los diseñadores de las aplicaciones la creación de unidades lógicas de instrucciones para mantener de la consistencia los datos.

- Para ello existe un conjunto de instrucciones SQL que permiten implementar el control de las transacciones

- **PROBLEMAS**: actualización sin tener en cuenta el control de concurrencia
  - produce inconsistencia de datos

Transacciones: Problemas en 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
- DEF: *Plan de ejecución*: el orden en que se ejecutarán la operaciones -> Ejemplos :

Algunos Problemas en Planes de Ejecución (sin Control de Concurrencia)

- PROBLEMA **a) Actualización perdida**

T_1	T_2
Leer(X)	
X := X - N	
	Leer(X)
	X := X + M
Escribir(X)	
Leer(Y)	
	Escribir(X)
Y := Y + K	

ejecución secuencial: una sola CPU

→ Error: pierde el X de T_1

Transacciones: Problemas en Multiprogramación

- PROBLEMA **b) Actualización Temporal (Lectura Sucia = no confirmada)**

T_1	T_2
Leer(X)	
X := X - N	
Escribir(X)	
	→Leer(X)
	X := X + M
	Escribir(X)
Rollback(X de T_1)	…….sigue
…… sigue	

Queda valor antigüo X →

→ Error: pierde el X de T_2

Transacciones: Problemas en Multiprogramación

- PROBLEMA **c) Resumen Incorrecto**

T_1	T_2
	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

Transacciones: Problemas en Multiprogramación

- PROBLEMA **d) Lectura no repetible**

T_1	T_2
Leer(X)	
	Leer(X)
	Escribir(X + 1)
Leer(X)	

Lee otro valor

- PROBLEMA **e) Lectura Fantasma**
  - Lectura de una fila que no existía cuando se inició la transacción
- PROBLEMA **X) Otros muchos casos . . .**

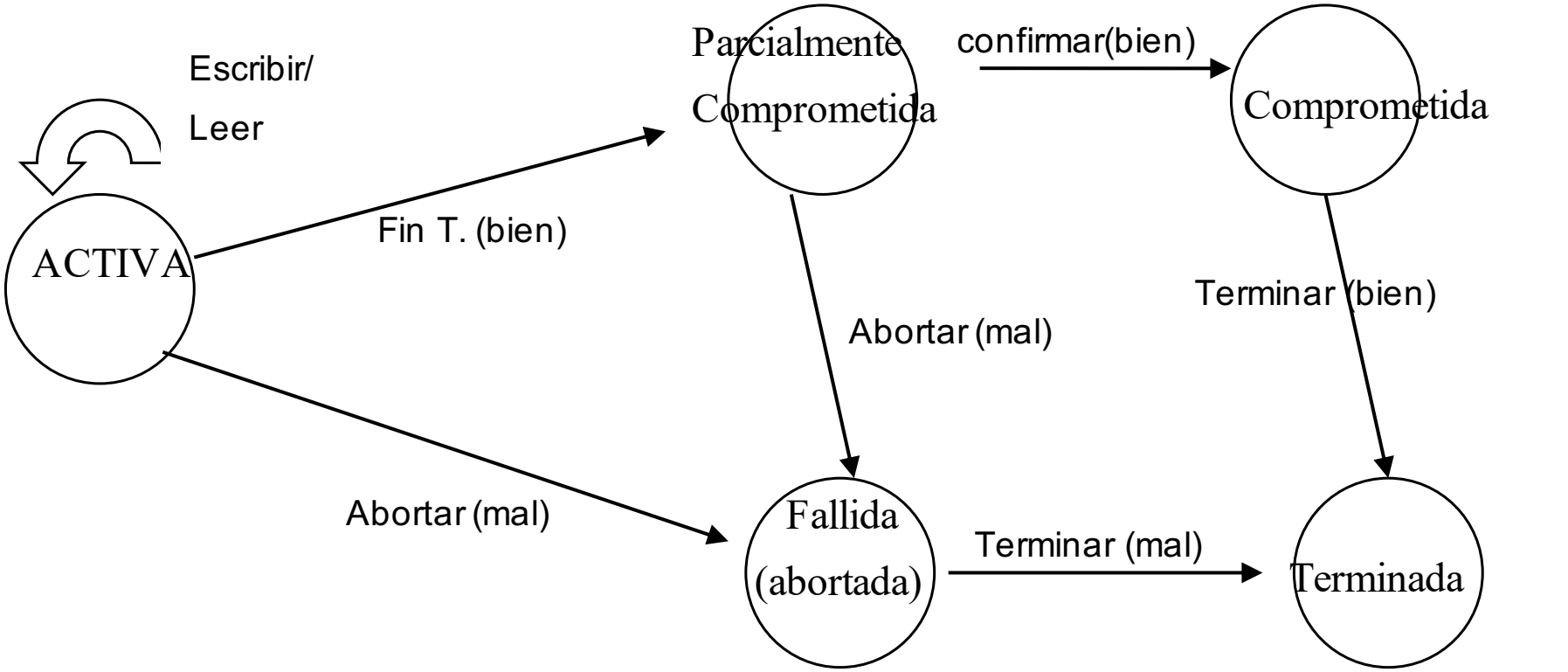
- Para conservar la integridad de los datos se han de cumplir propiedades A.C.I.D. (las BDs llamadas de tipo ACID)

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

Para que no haya inconsistencia de datos, las T's deben cumplir:

- ***A**tomicidad*: La transacción(T) funciona como unidad:
  - O se ejecutan todas sus operaciones o ninguna.
  - Se prohibe una ejecución parcial.
  - Responsable: el Método de Recuperación (Rollback)
- Conservar la ***C**onsistencia* de datos una vez ejecutada la transacción:
  - Responsable: los programadores mediante:
    - check, triggers, primary key, foreign key,…
- *Aislamiento* entre T's: una T en ejecución no permite acceso a sus
  - actualizaciones antes de que termine (sea confirmada)
  - Responsable: el *Método de Concurrencia* (→ varios niveles), decide programador
- ***D**urabilidad* 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.

Estados de una Transacción



- Hacer **Commit** = Confirmada = Comprometida => Marca un "punto de control".
- Fallos: caidas de sistema, físicos (eléctricos, disco), concurrencia(aborta T_1)
- Estado "Fallida": retrocede la T, rollback
  - Si fallo hardware, reinicia. Si fallo lógico, termina.
- para salvar puntos intermedios de una T: ***Savepoint NombrePunto***
- deja la BD como estaba antes del punto: ***Rollback[to savepoint NombrePunto]***

Terminación de Transacciones

- Las transacciones terminan **confirmadas** o **canceladas (abortadas).**
- Cuando una transacción se confirma,
  - *los cambios provocados por ella se hacen permanentes*.
- Las sentencias de una transacción pueden ejecutarse correctamente,
  - pero sus efectos podrán ser *cancelados*
  - mientras la transacción no sea *confirmada*
- La cancelación (*abortada*) de una T:
  - *Cambios hechos por instrucciones de una transacción son cancelados*

Aislamiento de Transacciones: Lectura Consistente

- **Lectura consistente** (L.C.): cuando durante la ejecución de toda la T,
  - las otras T's ven la BD congelada, con los valores al comienzo de la T,
  - para evitar el problema del *resumen incorrecto,* caso c)

- Dos niveles para conseguir la L.C. mediante dos comandos al empezar T :
  - A nivel de transacción: `SET TRANSACTON READ ONLY;`
    - esa T solo puede leer, no modifica; esto agiliza la BD
    - Solo ve las actualizaciones que estaban confirmadas antes de empezar T.
  - A nivel instrucción: *(por defecto en oracle)* `SET TRANSACTON READ WRITE;`
    - Solo ve actualizaciones confirmadas antes de ejecutar instrucción actual

- Da lugar a tres situaciones que vemos a continuación

Aislamiento de Transacciones: Lectura Consistente

- Tres situaciones sobre la L.C.:
  - **1.**- L.C. implícita (nivel instrucción): se mantiene dentro de la instrucción

    SELECT… → siempre en Oracle
  - **2.**- Se ve y pueden modificar la BD entre las dos consultas (sin L.C.)

    SELECT… → puede ser peligroso:

    SELECT… depende de lo que hagan las otras T's

    COMMIT; (fin de una T)
  - **3.**- L.C. explícita : → decide el programador cuando

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

Niveles Aislamiento

- (Oracle) Dos modos de poner el aislamiento para evitar interferencias entre T's
  - **`SET TRANSACTION ISOLATION LEVEL [SERIALIZABLE | READ COMMITED];`**
  - **`ALTER SESSION SET ISOLATION_LEVEL; ->`** para toda la sesión

HAY DOS NIVELES de Aislamiento según su visibilidad:

- **`SERIALIZABLE`** : "*secuenciable*". → puede relentizar mucho el resto de Ts
  - T's no pierden actualizaciones → ver problema a)
  - se garantizan lecturas repetibles (y L.C.) → ver problema d)
  - no hay registros fantasma (probl. e) y tampoco resumen incorrecto (probl. c)
  - Ninguna de las otras T's en el mismo nivel pueden ver las modificaciones hechas por un T, incluso después de que T haga commit.
  - Otras T's en el nivel Read commited siguen las normas de dicho nivel.
  - Si T_i actualiza algo que ya actualizó T_j y T_j está sin confirmar: T_i aborta y se encola
  - Vemos luego qué es un plan secuenciable

Niveles Aislamiento

- **`READ COMMITED`** : (por defecto en Oracle al empezar una ejecución)
  - Lectura Consistente. Y la T no tiene lecturas repetibles.
  - Modificaciones hechas por una T en este nivel
    - son visibles por otras T's en el mismo nivel solo si T ha hecho commit.
  - Otras Ts en nivel Serializable siguen sus normas.

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

Plan Secuenciable

Esos Niveles de Aislamiento se basan en estos conceptos:

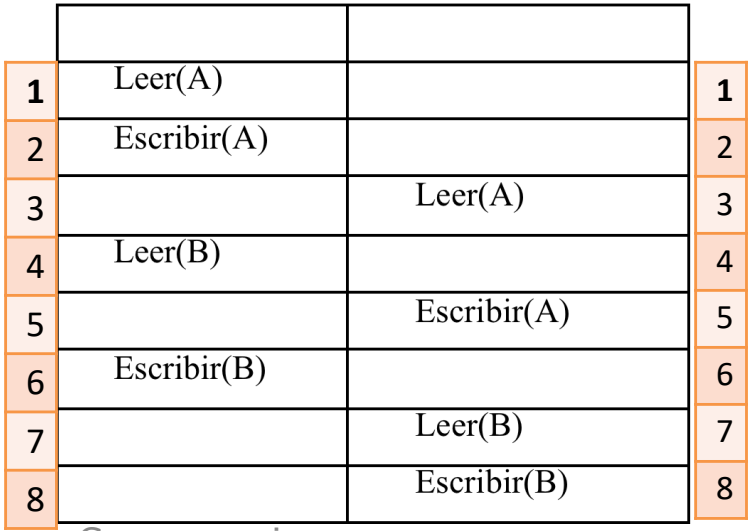
- Un *plan secuencial* (o en serie) de un conjunto de **T's** sucede:
  - Si todas las operaciones de cada $\mathbf{T_i}$ se ejecutan en secuencia
  - sin que se intercale ninguna operación de otra $\mathbf{T_j}$
- Todos los planes secuenciales son correctos: garantizan las propiedades A.C.I.D.
- **Problema:** T_2 pierde el tiempo esperando
- **Solución:** Reordenar para que no espere, siempre que no altere los conflictos.

→*Dos ops. seguidas de un plan están en* ***conflicto*** *si se dan las tres condiciones*:
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)"

	T_1	T_2	
1	Leer(A)		1
2	Escribir(A)		2
3	Leer(B)		3
4	Escribir(B)		4
5		Leer(A)	5
6		Escribir(A)	6
7		Leer(B)	7
8		Escribir(B)	8

Plan Secuenciable

- 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 el orden de ejecución entre ellas dos.
- Esto permite conseguir planes equivalentes más eficientes (intercalando todo lo posible).
- Un plan P es secuenciable en cuanto a conflictos si se puede encontrar un plan secuencial P' que sea equivalente por conflictos al P.
- Este plan es equivalente por conflictos al anterior y más eficiente, no deja esperando a T_2 todo el tiempo.



Plan Recuperable

- El beneficio de un plan Secuenciable es que pueda ser Recuperable
- Un plan es *recuperable* si
  - para todas sus T's que tienen Leer(X) no se confirman hasta que
  - todas las T's que tienen Escribir(X) se han confirmado.

Niveles Aislamiento: Cuál escojo?

- **Criterios**: **Rendimiento, necesidad de tipo de consistencia y requisitos del código.**

- Recomendado **`serializable`** cuando:
  - Grandes BDs con T's cortas y actualizan pocas filas.
  - Baja probabilidad de que dos T's modifiquen la misma fila y que
    - la mayoría de las T's, que se ejecutan mucho tiempo, son de lectura.
    - Se mejora si: se ponen primero las instrucciones que pueden colisionar
  - Protege mejor de lecturas fantasma e irrepetibles cuando:
    - una T lee dos veces la misma fila.
  - Garantiza la L.C. si una T tiene consultas con subconsultas
  - NO es recomendable para entornos con muchos deadlocks
    - que provoquen muchos rollbacks y reintentos.

Niveles Aislamiento: Cuál escojo?

- Recomendado **`READ COMMITED`** cuando:
  - Si muchos usuarios concurrentes y tiempo de respuesta crítico:
    - Compensar consistencia y concurrencia
  - Si pocos usuarios concurrentes y pocas posibilidades de
    - lecturas fantasmas y lecturas irrepetibles.
    - Y además pocas transacciones necesiten esos tipos de lecturas.
  - Permite
    - Respuesta más rápida pero más insegura
    - Más concurrencia con cierto riesgo de lectura fantasma y lecturas irrepetibles
- Al escribir el código:
  - las lecturas no bloquean las escritura en ningún nivel de Aislamiento

Niveles Aislamiento: ANSI vs Oracle

- Niveles de AISLAMIENTO (ANSI): "Sí" indica que es posible que suceda

Nivel de aislamiento	Lectura sucia	Lectura no repetible	Lectura fantasma
Lectura no comprometida *(read uncommited)*	Sí	Sí	Sí
Lectura comprometida *(read commited)*	No	Sí	Sí
Lectura repetible *(repeatable read)*	No	No	Sí
Secuenciable (*serializable*)	No	No	No

- Niveles de AISLAMIENTO (Oracle):
  - Read only
  - Isolation level read commited
  - Isolation level serializable

Control de transacciones

- La sentencia **`COMMIT`** (comprometer o confirmar) termina una transacción y hace permanentes y visibles los cambios al resto de los usuarios.

- La sentencia **`ROLLBACK`** (retroceder) descarta todos los cambios realizados durante la transacción actual, esto es, desde el último **`COMMIT o ROLLBACK`**.

- La instrucción **`SAVEPOINT <nombre>`** (punto de salvaguarda) establece un punto en una transacción hasta el cual se podrá cancelar la transacción mediante una sentencia **`ROLLBACK TO SAVEPOINT <nombre>.`**
  - Muy útil en Ts con muchas de actualizaciones en proceso por lotes: EJ: Generar la nómina de una empresa con 1.000.000 empleados

Programación de transacciones

- Una transacción comienza con
  - la primera instrucción SQL ejecutable: DML y DDL
    - Que modifique la BD
  - o bien en la instrucción **`SET TRANSACTION`**
- Una transacción finaliza cuando . . . : (siguiente transparencia)
- Tras la finalización de una transacción,
  - la siguiente instrucción SQL inicia automáticamente la siguiente transacción.
  - En algunos sistemas, como MS SQL Server,
    - el inicio de una transacción se hace explícito con
      - BEGIN TRAN[SACTION]

Programación de transacciones

- Hay operaciones implícitas: se realizan sin mediar el usuario
  - La op. explícitas se escriben en los scripts de SQL o PL/SQL
- Una transacción termina en las siguientes situaciones:
  - La ejecución explícita de la sentencias **COMMIT** o **ROLLBACK** sin la cláusula **SAVEPOINT**.
  - La ejecución de una instrucción DDL.
    - El gestor realiza de forma implícita un **COMMIT**
      - antes y después de cada sentencia DDL.
  - La ejecución implícita de un **COMMIT** cuando el usuario termina la sesión. Este comportamiento es configurable.
  - La ejecución implícita de un **ROLLBACK** debida a un error durante la ejecución de un proceso.

Programación de transacciones

- Autoconfirmación de transacciones: (Autocompromiso)
  - Se activa con `SET AUTOCOMMIT ON`
  - Aplica siempre una confirmación (**COMMIT**)
    - después de cada instrucción SQL
  - se desactiva con: `SET AUTOCOMMIT OFF`

- Uso con precaución:
  - puede no ser adecuado a la situación
  - Muy útil en pruebas o en tareas especiales de administrador

Ejemplo (autocommit off)

- UPDATE cuentas
  SET saldo= saldo- 500
  WHERE cuenta= 3209;
- UPDATE cuentas
  SET saldo = saldo + 500
  WHERE cuenta=3208;
- INSERT INTO movimientos
  VALUES (mov_seq.NEXTVAL,
  'TR' 3209, 3208, 500);
- COMMIT;

LA TRANSACCION COMIENZA AQUI

SENTENCIAS DE LA TRANSACCIÓN

LA TRANSACCION TERMINA AQUI

- SELECT …
- UPDATE …

UNA NUEVA TRANSACCION COMIENZA AQUI

Puntos de salvaguarda (Savepoints)

- Es un Punto intermedio (punto de control) en una T_x que
  - permiten guardar el trabajo realizado hasta él mismo
  - y a partir del cual es posible volver y continuar el proceso
- Un punto de control se establece mediante la instrucción

```
SAVEPOINT <nombre>
```

- Para retroceder una transacción hasta un punto de control:

```
ROLLBACK TO SAVEPOINT <nombre>
```

- Cuando se retrocede una transacción:
  - Solo deshace las sentencias realizadas después del punto de control
  - Conserva el savepoint correspondiente y los savepoints anteriores a este
  - Libera los bloqueos obtenidos tras el savepoint y conserva los obtenidos antes del mismo
- Una transacción que se retrocede a un punto de control
  - permanece *activa*
- El usuario puede asignar opcionalmente un nombre a una T:

```
SET TRANSACTION NAME <nombre>
```

Control de transacciones

- Ejemplo de ejecución:

```
CREATE TABLE EMPLEADO
 (NIF VARCHAR2(9) NOT NULL,
 NOMBRE VARCHAR2(20),
 SALARIO NUMBER(6,2),
 APELLIDOS VARCHAR2(40),
 CONSTRAINT EMP_PK PRIMARY KEY (NIF));

INSERT INTO empleado
VALUES('10000000A','Jorge',3000.11,'Perez Sala');

ROLLBACK;

INSERT INTO empleado
VALUES('30000000C','Javier',2000.22,'Sala Rodriguez');

INSERT INTO empleado
VALUES('30000000C','Soledad',2000.33,'Lopez J.');
```

Control de transacciones

```
INSERT INTO empleado
VALUES('40000000D','Sonia',1800.44,'Moldes R.');

INSERT INTO empleado
VALUES('50000000E','Antonio',1800.44,'Lopez A.');

COMMIT;
```

Control de transacciones

Resultado del Ejemplo:

- table EMPLEADO creado.
- 1 filas insertadas.
- rollback terminado.
- 1 filas insertadas.
- Error que empieza en la línea 18 del comando:

```
INSERT INTO empleado
VALUES('30000000C','Soledad',2000.33,'Lopez J.')
Informe de error:
Error SQL: ORA-00001: unique constraint (MGM.EMP_PK) violated
1 filas insertadas.
```

- 1 filas insertadas.
- confirmado.

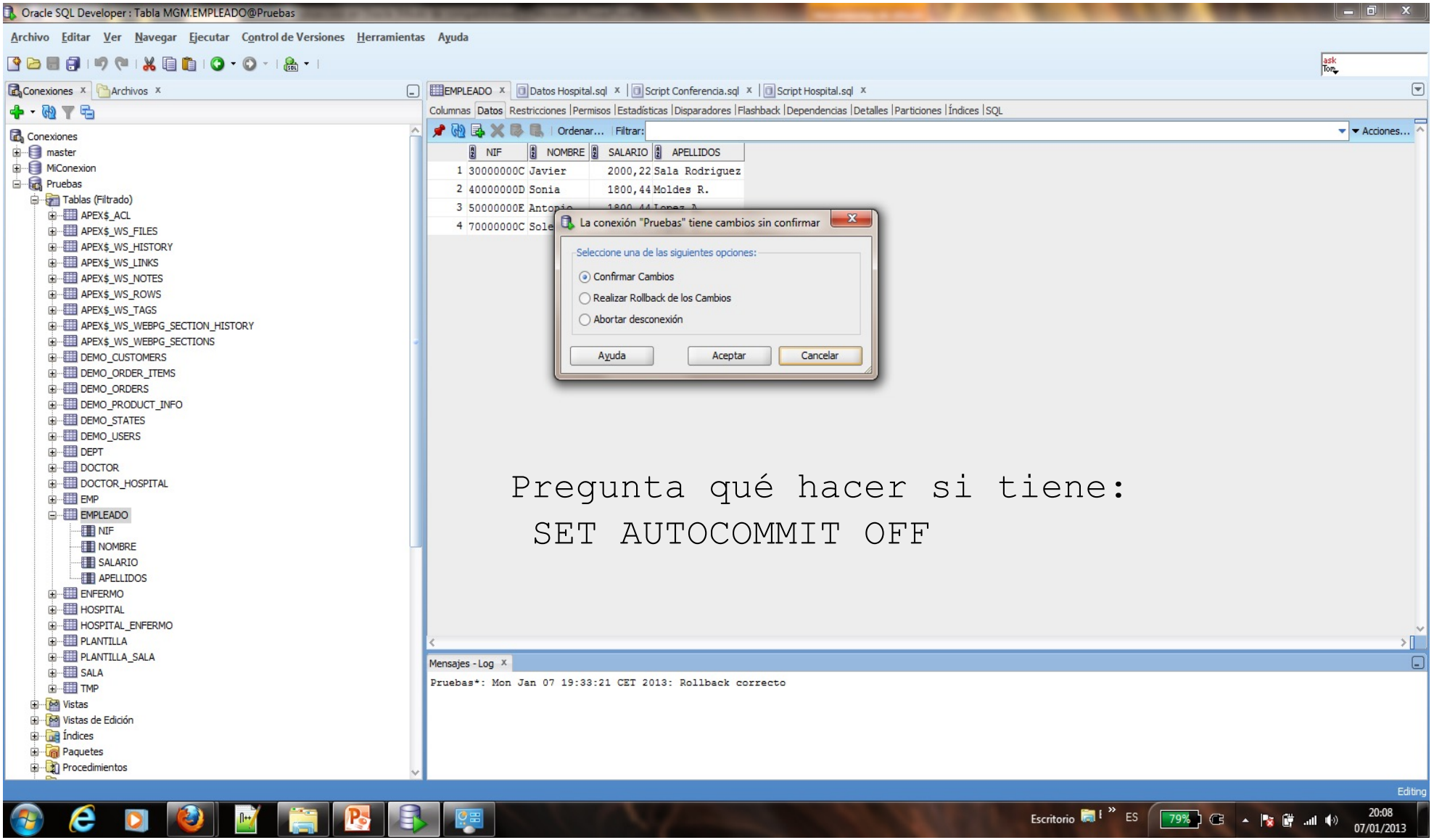
Control de transacciones

30000000C	Javier	2000,22	Sala Rodriguez
40000000D	Sonia	1800,44	Moldes R.
50000000E	Antonio	1800,44	Lopez A.

```
INSERT INTO empleado
VALUES('70000000C','Soledad',2000.33,'Lopez J.');
```

– Este registro solo será permanente en la base de datos y visible al resto de los usuarios si se confirma la transacción.

Control de transacciones



Pregunta qué hacer si tiene:
SET AUTOCOMMIT OFF

Control de Concurrencia Multiversión (Oracle)

- Cuando actualice un dato, no sobre-escribe el dato
  - Sino, que marca el valor actual como obsoleto usando el timestamp
  - Y añade una versión nueva del dato
  - Los demás siguen leyendo la version anterior. No deja update

- Se crean multiples versiones de un dato:
  - Cada transacción tiene su version, aunque obsoleta, no cambia:
    - Consistencia de datos “en el tiempo”
  - Solo una es la última version

- Así se consigue el aislamiento entre T’s sin bloqueos de tabla
  - Hace bloqueos de fila (aunque físicamente es del bloque entero)

Bloqueos de datos (LOCKS)

- Un Nivel de aislamiento es sobre toda la T.
  - El aislamiento puede ser de alto coste al relentizar otras T's
- Los LOCKs o bloqueos son sobre toda la tabla o sobre algunas filas
- Los LOCKs o bloqueos son de menor coste, son de grano más fino
- . . .Pero seguimos bajo un nivel de Aislamiento (escogido o por defecto)
- 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 (lecturas consistentes).

Tipos o Niveles de bloqueos

- Los niveles básicos de bloqueos son:
  - S o BS (shared), para lectura (SELECT)
  - U o BU (cursores FOR UPDATE), lectura para luego posible modificación
  - X o BX (Exclusivos), para modificación (INSERT, UPDATE, DELETE)

- La **T.1ª** tiene un tipo de bloqueo (columna izq de cada fila), entonces
  - La **T.2ª** pide un tipo de bloqueo (cabecera del resto de columnas)
  - Fila: dado el bloqueo de T.1ª: La casilla es si concede el bloqueo pedido por T.2ª
- La tabla de compatibilidad entre dos transacciones requiriendo bloqueos sobre el mismo elemento es:

T.1ª ↓ T.2ª → Bloqueo	S	U	X
S	Sí	Sí	No
U	Sí	No	No
X	No	No	No

Bloqueos de datos (LOCKS)

- Los bloqueos básicos son a dos niveles:
  - *Nivel de Tabla*, normalmente si hacemos:
    - utilidades de mantenimiento: RUNSTATS, BACKUP, RECOVER, etc.
    - Aplicaciones : DDL y bloqueos de intención antes de bloquear filas.
  - *Nivel de Fila*
    - DML en aplicaciones

- Oracle gestiona los *Bloqueos automáticos* (*implícitos*)
  - para garantizar los niveles de aislamiento
  - pueden ser a *nivel de fila* (los más estrictos) o a *nivel tabla*
  - Solucionan la mayoría de las situaciones

  - En cada instrucción DML se hace implícitamente:
    - Un BX a la fila
    - Un BS a la tabla (para que no la bloquee otra T)

Bloqueos de datos (LOCKS)

- Además, el programador puede usar los *Bloqueos explícitos*
  - Porque el nivel de aislamiento *serializable* puede ralentizar las transacciones
  - Sin olvidar que Oracle sigue haciendo
    - los *bloqueos implícitos*
    - y que estamos dentro de un nivel de aislamiento

- Instrucción: **`LOCK TABLE mitabla IN XXXX MODE [NOWAIT] ;`**

  donde **XXXX** puede ser:

  SHARE (S o BS) Lectura concurrente . Permite otros locks SHARE

  todos pueden leer, pero ninguno escribir

  EXCLUSIVE (X o BX) No permite ningún otro lock.

  otros solo pueden leer la versión anterior al lock

  donde **NOWAIT** es para que la T 2ª que lo pide aborte en vez de esperar

Duración de los bloqueos

- Los bloqueos exclusivos (INSERTs, UPDATEs, DELETEs) se liberan con
  - COMMIT o ROLLBACK

- Los bloqueos de lectura terminan
  - con la transacción o con la sesión
  - de acuerdo a cómo se estableció el nivel de aislamiento con
    - SET TRANSACTION ISOLATION . . .
    - ALTER SESSION SET ISOLATION_LEVEL . . .

Protocolos de Bloqueos

- Protocolo de bloqueo en dos fases básico (B2F): oracle
  - si todas las ops de bloqueo (BC, BX) preceden a la 1ª op de desbloqueo de T
  - Oracle usa este aplicando el Control de Concurrencia Multiversión

- Es decir, que se pueden dividir en dos fases:
  - 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 EJ-9 T_2 espera…
  - Mejoras con otros protocolos. . .

Protocolos de Bloqueos

- 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)

Transacciones Autónomas

- Un procedimiento o trigger puede ejecutarse como
  - una T independiente de la T principal desde donde se llama
  - No comparte ni locks, ni recursos ni dependencias de commit

- EJ uso: un log de actividades para registrar todas las operaciones
  - incluso las que abortan y se deshacen con rollback
- EJ uso: Oracle implícitamente si hay una DDL en un proc. o trigger

- Se define poniendo en la sección de variables:

  `pragma AUTONOMOUS_TRANSACTION` (orden para el compilador)
- Termina con un commit o rollback

- Si la T principal que tiene el nivel aislamiento:
  - READ COMMITTED : cuando commit la T autónoma,
    - la T principal ve lo modificado por ella
  - SERIALIZABLE : cuando commit la T autónoma,
    - la T principal NO ve lo modificado por ella

Transacciones Autónomas

- Si la T principal hace rollback
  - a un savepoint anterior a la llamada a la T autónoma:
  - a la T autónoma no le afecta, lo hecho queda hecho.

- Si la T autónoma aborta y hace rollback: no afecta a la T principal
  - Se deben capturar las excepciones que se han generado

- Si la T autónoma accede a un recurso bloqueado por la T principal:
  - Se produce un deadlock y una excepción en la T autónoma
    - Si no hay captura: hacer rollback de la T autónoma

- Si la T autónoma termina sin commit o rollback:
  - se genera una excepción y, si no se captura, la T autónoma hace rollback

- Los triggers y procs autónomos pueden:
  - Ejecutar DDL (create, drop)con EXECUTE IMMEDIATE
  - Tener COMMIT y ROLLBACK

Detalles Informativos Adicionales

Niveles de aislamiento ANSI: Más detalles

Repeatable reads

- Los bloqueos se mantienen hasta el final de la transacción.
- Protege la lectura física, protegiendo las filas leídas de UPDATEs y DELETEs
  - si se lee dos veces la misma fila, el resultado es el mismo,
  - pero permite el efecto de fila fantasma.
- Por ejemplo, si se pone la misma SELECT en la misma transacción,
  - las filas de la primera SELECT aparecen en la segunda
    - (están protegidas por un bloqueo S), pero puede aparecer alguna fila de más (fila fantasma).

Niveles de aislamiento ANSI: Más detalles

Read committed

- Los bloqueos de lectura se liberan en cuanto se termina de leer (con SELECT o FETCH).
- Si se repite una SELECT, la segunda puede ser completamente distinta (**lectura no repetible**). Al no mantener los bloqueos de lectura de las filas leídas, estas pueden ser modificadas o borradas por otras transacciones.
- Es el nivel de aislamiento más concurrente que garantiza la integridad de lo leído (que los datos son confirmados).

Niveles de aislamiento ANSI: Más detalles

Read uncommitted

- Ni bloquea las lecturas ni espera por bloqueos, ni siquiera por bloqueos exclusivos.
- Esto puede provocar que se lea un dato modificado por otra transacción no confirmada, y si dicha transacción realice un ROLLBACK, provocaría una lectura de un dato nunca confirmado (**lectura sucia**).
- Es el menor nivel de aislamiento y el más concurrente (las lecturas nunca esperan), pero no ofrece integridad de la lectura.
- Se usa en minería de datos (medias, etc.) o sobre datos históricos, Data Warehouses, (que no tienen modificaciones, sólo cargas) etc.
- ORACLE no lo soporta.

Niveles de aislamiento en ORACLE: Más detalles

- Oracle soporta los niveles de aislamiento READ COMMITTED y SERIALIZABLE.
  - No necesita lectura sucia (READ UNCOMMITTED) pues
  - cuando se intenta leer filas bloqueadas por modificaciones,
  - muestra lo último confirmado, sin esperar a que se libere el bloqueo exclusivo.

- Es decir, la lectura en ORACLE no espera por bloqueos
  - (equivalente a lectura sucia en otros gestores)
  - pero es consistente (lee, sin esperar, lo último confirmado,
  - pero no se da cuenta si otras transacciones están modificando los datos).

Niveles de aislamiento en ORACLE: : Más detalles

- Oracle Database añade un nivel de aislamiento propio:
  - READ ONLY, indicado para transacciones sólo de lectura.

- Una transacción con nivel de aislamiento READ ONLY
  - no puede realizar modificaciones y
  - sólo ve los cambios: confirmados antes de empiezar la transacción.