TEMA 8.
LENGUAJE DE MANIPULACION DE DATOS (DML) EN SQL

Permite agregar, modificar, o eliminar la información almacenada en una base de datos. Esta categoría del SQL está integrada por las siguientes instrucciones:

- `insert`
- `update`
- `delete`
- `merge`

8.1. SENTENCIA INSERT:

Como su nombre lo indica, la sentencia `insert` se emplea para agregar registros a una tabla en la base de datos.

Sintaxis SQL estándar (misma sintaxis para los demás manejadores):

```
insert into <table-or-view-name> [(<column_name1,column_name2, . . . )]
{
    { values(<literal> | <expression> | null | default), . . .}   |
   { <select-statement>}
}
```

Ejemplos:

La forma más simple de insertar datos en una tabla es la siguiente: (es válido, pero no se recomienda) omitir la lista de columnas ya que, por default, si no se especifica, se toma el orden en el que se definió la tabla.

EMPLEADO

EMPLEADO_ID	NUMERIC(10,0)	NOT NULL
NOMBRE	VARCHAR(40)	NOT NULL
APELLIDO_PATERNO	VARCHAR(40)	NOT NULL
APELLIDO_MATERNO	VARCHAR(40)	NULL
FECHA_NACIMIENTO	DATE	NOT NULL
CONYUGE_EMPLEADO_ID (FK)	NUMERIC(10,0)	NULL
JEFE_INMEDIATO (FK)	NUMERIC(10,0)	NULL
CONYUGE_ID (FK)	NUMERIC(10,0)	NULL
PUESTO_ID (FK)	NUMERIC(10,0)	NOT NULL

```
insert into empleado
values (2,'angela','ramirez','luna',
    to_date('1980/01/10 10:40:00','yyyy/mm/dd hh24:mi:ss'),1,1,null,1);
```

- Las cadenas siempre se especifican entre comillas simples (`''`)
- La función TO_DATE es particular de Oracle, y se emplea para generar objetos tipo DATE. El primer argumento indica el valor a procesar, y el segundo indica el formato, el cual debe corresponder con el formato del primer argumento (en temas posteriores se revisa a detalle las funciones para manejo de fechas).

Ejemplos:

- `to_date('2003/07/09', 'YYYY/MM/DD')`
- `to_date('070903', 'MMDDYY')`
- `to_date('20020315', 'YYYYMMDD')`

- La desventaja de la forma corta, es que se debe conocer de antemano el orden el en que se definieron los campos al crear la tabla, es propensa a errores ya que en la secuencia no se ve de forma clara a que campo pertenece cada valor.
- Adicionalmente, se deben especificar todos los valores de los campos, aunque estos sean nulos, por ejemplo, en el caso del campo `conyuge_id`, si no se cuenta con el valor, se tiene que escribir la palabra "`null`" para indicarle al manejador la ausencia de dicho valor.

Forma recomendada:

```
insert into empleado (empleado_id, nombre, apellido_paterno,
    apellido_materno, fecha_nacimiento, conyuge_empleado_id,
    jefe_inmediato_id,puesto_id)
values (2,'angela','ramirez','luna',
    to_date('1980/01/10 10:40:00','yyyy/mm/dd hh24:mi:ss'),1,1,1);
```

- Es importante mencionar que las columnas definidas como `not null`, es requerido especificar su valor.
- Observar que para el caso de valores nulos no es necesario incluir la palabra `null`. En estos casos, se omite el nombre del campo en la lista (`conyuge_id`).

8.1.1. Inserción valores seleccionados de otra tabla.

- Suponer que se desea copiar los datos insertados en una tabla hacia otra, por ejemplo, de `empleado` a `empleado_respaldo`

Ejemplo:

```
insert into empleado_respaldo
   select * from  empleado where empleado_id=1;
```

Ejemplo:

```
insert into empleado_respaldo (empleado_id, nombre,
    apellido_paterno, apellido_materno,fecha_nacimiento,puesto_id)
    select empleado_id, nombre, apellido_paterno, apellido_materno,
        fecha_nacimiento,puesto_id
    from empleado
    where empleado_id=2;
```

8.1.2. Insertando nuevos registros empleando una secuencia.

Esta estrategia evita realizar el control manual de los valores a asignar para la llave primaria de una tabla:

```
create sequence empleado_seq;
```

```
insert into empleado(empleado_id,nombre,apellido_paterno,apellido_materno,
     fecha_nacimiento,puesto_id)
values (empleado_seq.nextval,'angel', 'juarez','aguirre',
    to_date('1983/06/10 10:40:00','yyyy/mm/dd hh24:mi:ss'),1);
```

- Para generar un nuevo valor de la secuencia en Oracle, se emplea `nextval`.

8.2. LA SENTENCIA UPDATE

La sentencia `update` permite actualizar los valores de uno o más campos asociados a uno o más registros de una tabla.

Sintaxis SQL estándar:

```
update<table-or-view-name>set<column-name> =
{<literal> | <expression> | <single-row-select-statement>
   | null | default }, . .
[where<predicate>]
```

- Como se puede observar en la sintaxis anterior, el valor del campo a actualizar puede ser un valor fijo (literal), determinado a través de una expresión aritmética, booleana, a partir del resultado de la invocación de una función, o el resultado de un subquery. (single row select statement).
- El predicado o condición puede ser opcional.

Retomando el caso de estudio del sistema de nómina:

Ejemplo:
Modificar el sueldo del puesto Director General (DJ) a 75000

```
update puesto set sueldo_tabulador= 75000 where clave_puesto='dj';
```

Ejemplo:
Actualizar el puesto del empleado `ANGEL JUAREZ AGUIRRE` de director general (DG) a jefe de departamento (JD)

```
update empleado set puesto_id =
(select puesto_id
 from puesto
 where clave_puesto ='jd'
)
where nombre='angel'
    and apellido_paterno='juarez'
    and apellido_materno='aguirre';
```

Ejemplo:

El empleado `ANGELA RAMIREZ LUNA` se ha divorciado, se requiere reflejar los cambios en la base de datos. Adicionalmente se requiere corregir su fecha de nacimiento al 11-01-1980.

```
update empleado set conyuge_empleado_id =null,
    fecha_nacimiento  = to_date('11-01-1980','dd-mm-yyyy')
where  nombre='angela'
and apellido_paterno='ramirez'
and apellido_materno='luna';
```

Ejemplo:

Se desea agregar un registro en la tabla `pensionada` y asociarla con los datos del empleado, por default el porcentaje de pensión es del 10% del sueldo del empleado. Los datos son los siguientes:

Datos de la pensionada:

Nombre completo: `JUANA MENDOZA AGUILAR`

Fecha de nacimiento: `23/07/1964`

Datos del empleado asociado:

Nombre del empleado: `JUAN MARTINEZ LOPEZ (ID = 1)`

Observar la definición de la tabla `empleado_pensionada` y el uso de `default`, revisar el modelo relacional para mayor comprensión.

```
create table empleado_pensionada(
    empleado_id  number(10,0),
    pensionada_id number(10,0),
    porcentaje number(4,2) default 0.1,
    constraint emp_pensionada_pk primary key(empleado_id,pensionada_id),
    constraint emp_pensionada_fk foreign key(empleado_id) references
        empleado(empleado_id),
    constraint pensionada_fk foreign key(pensionada_id) references
        pensionada(pensionada_id)
);
```

Insertando los datos de la pensionada con un porcentaje del 34%:

```
insert into pensionada(pensionada_id,nombre,
    apellido_paterno,apellido_materno,fecha_nacimiento)
values (1,'juana','mendoza','aguilar',to_date('23/07/1964','dd/mm/yyyy'));

insert into empleado_pensionada(empleado_id,pensionada_id,porcentaje)
values (1,1,0.34);

select *
from empleado_pensionada
where empleado_id=1
and pensionada_id=1;

EMPLEADO_ID PENSIONADA_ID PORCENTAJE
=========== ============= ==========
1           1             .34
```

Se solicita actualizar el porcentaje de la pensionada registrada anteriormente a su valor por default:

```
update empleado_pensionada
set porcentaje = default
where pensionada_id = 1
and empleado_id=1;

select *
from empleado_pensionada
where empleado_id=1
and pensionada_id=1;


EMPLEADO_ID PENSIONADA_ID PORCENTAJE
=========== ============= ==========
1           1             .1
```

- Adicional a la sentencia `AND`, es posible utilizar la sentencia `OR`, existen más opciones, las cuales se revisarán más adelante, en especial en el tema de funciones de agregación.
- Pregunta: ¿Qué sucede si se omite la cláusula `WHERE` de la instrucción `UPDATE`?

8.3. LA SENTENCIA DELETE

- Como su nombre lo indica, `delete` se emplea para eliminar registros de una tabla.

Sintaxis SQL estándar:

```
delete
from <table-or-view-name>
[where <predicate>]
```

La única diferencia con la sintaxis en Oracle, es que la cláusula `from` es opcional.
`delete` borra únicamente los registros de una sola tabla a la vez a excepción de que exista la cláusula `on delete cascade` en la definición de las tablas involucradas

Ejemplo:
Eliminar todos los registros asociados a los hijos del empleado `JUAN MARTINEZ LOPEZ`.

```
delete from hijo_empleado
where  empleado_id = (
    select empleado_id
    from empleado
    where nombre='juan'
    and apellido_paterno='martinez'
    and apellido_materno='lopez'
);
```

Importante: tener cuidado con el uso de `delete`, ya que, si no se especifican las condiciones correctas en la cláusula `where`, pueden eliminarse registros no deseados, o inclusive el contenido completo de la tabla.

8.4. LA SENTENCIA MERGE

`MERGE` es una combinación de `insert`, `update` y `delete`, funciona de la siguiente forma:

MERGE inserta un registro si este no existe o actualiza determinadas columnas con base a un criterio si el registro ya fue insertado anteriormente.

Sintaxis SQL estándar.

```
merge into [<qualifier>.] <target-table>
using [<qualifier>.] <source-table> on (<condition>)
when matched then
update set {<column> = {<expression>|<default>},...}
when not matched then
insert [<column-name>,...]  values ({<expression>|<default>},...);
```

Ejemplo:

El uso común de esta instrucción es cuando se desea realizar una sincronización entre 2 tablas. Considerar los datos de las tablas producto y producto_respaldo, se desea que el contenido de producto sea actualizado en producto_respaldo.

Contenido de la tabla producto:

```
PRODUCTO_ID TIPO NOMBRE  FECHA_CREACION
=========== ==== ======= ===================
1           A    LAP-TOP 2010-10-27 23:18:21
2           B    MOUSE   2010-01-01 12:00:00
3           C    TECLADO 2010-01-02 12:00:00
4           D    MEMORIA 2010-01-05 11:00:00
5           E    TECLADO 2010-01-10 10:00:00
```

Contenido de la tabla producto_respaldo

```
PRODUCTO_ID TIPO NOMBRE    FECHA_CREACION
=========== ==== ========= ===================
2           X    MOUSE GEN 2010-11-15 15:23:26
3           T    TECHADO   2010-11-15 15:23:43
10          A    MOUSE PAD 2010-11-15 15:23:44
```

Instrucción merge:

```
merge into producto_respaldo r using producto p on
 (r.producto_id = p.producto_id)
when matched then update
 set r.tipo=p.tipo, r.nombre=p.nombre,r.fecha_creacion=p.fecha_creacion
when not matched  then insert
 (r.producto_id,r.tipo,r.nombre,r.fecha_creacion) values
 (p.producto_id,p.tipo,p.nombre,p.fecha_creacion);
```

Después de ejecutar MERGE, los datos en PRODUCTO_RESPALDO quedarán así:

```
select * from producto_respaldo;
```

```
PRODUCTO_ID TIPO NOMBRE    FECHA_CREACION
=========== ==== ========= ===================
1           A    LAP-TOP   2010-10-27 23:18:21
2           B    MOUSE     2010-01-01 12:00:00
3           C    TECLADO   2010-01-02 12:00:00
4           D    MEMORIA   2010-01-05 11:00:00
5           E    TECLADO   2010-01-10 10:00:00
10          A    MOUSE PAD 2010-11-15 15:23:44
```

Observar que registros existentes en `PRODUCTO_RESPALDO` no son registrados en `PRODUCTO`.

8.5. LA INSTRUCCIÓN TRUNCATE

La instrucción `truncate` elimina el contenido de una tabla de forma más eficiente que `delete`, emplea menos recursos del sistema para ejecutar la instrucción. Posibles Triggers asociados a la eliminación de un registro no es disparado al ejecutar `truncate`.
`truncate` no puede emplearse cuando existen referencias de los datos de la tabla hacia otras.

Sintaxis:

```
truncate table <nombre-tabla>;
```

Ejemplo:

```
truncate table producto_respaldo;
```

Otro aspecto importante a considerar es que en Oracle, `truncate` es irreversible, y si se ejecuta dentro de una transacción en la que pueden estar participando otras instrucciones por ejemplo, `insert`, `update` , al ejecutar `truncate`, se ejecuta `commit` de forma implícita haciendo permanentes los cambios aplicados por las demás instrucciones y termina la transacción.

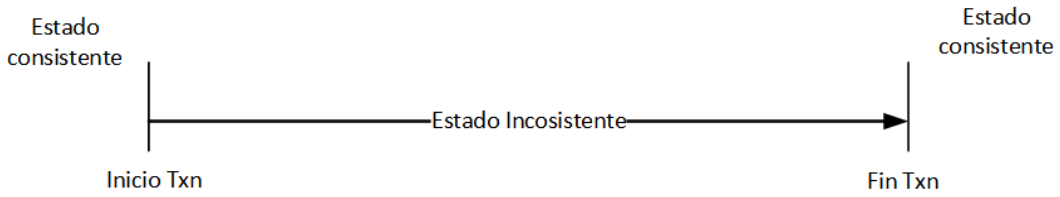
8.6. INTRODUCCIÓN AL CONCEPTO Y MANEJO DE TRANSACCIONES.

8.6.1. Definición de transacción.

- Unidad básica de procesamiento de datos que puede ejecutarse de forma confiable y consistente.
- Formada por una serie de operaciones de lectura y escritura.

8.6.2. Consistencia de una Base de datos.

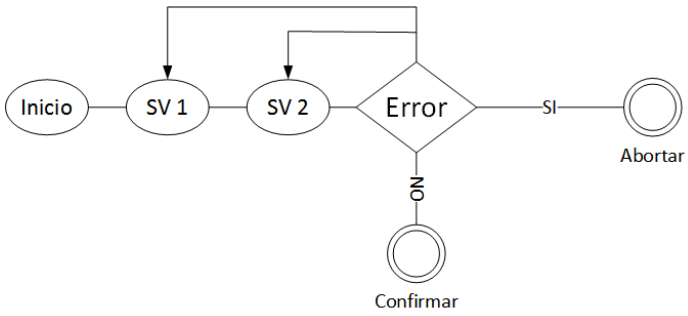
- Una BD se encuentra en un estado consistente si todas las restricciones y reglas de integridad definidas en ella se cumplen.
- El estado de una BD cambia al realizar operaciones de `insert, update, delete`.
- Para garantizar integridad, la BD no debería encontrarse en estado inconsistente.
- Sin embargo, la BD ***adquiere un estado de inconsistencia*** durante la ejecución de una Transacción, pero al terminarla debe recuperar el estado consistente:



8.6.3. Control transaccional.

- Se refiere a la administración de cambios realizados por instrucciones DML.
- Para realizar el control de transacciones se emplean las instrucciones `commit, rollback y savepoint`.
- A nivel de programación, el control transaccional debe contar con una estructura que garantice el estado consistente de la BD antes y después de la ejecución de una transacción.

```
begin transaction <txn_name>
   instrucción 1…
   instrucción 2…
   savepoint svp1;
   instrucción 3…
   instrucción 4…
   savepoint svp2;
   instrucción 5…
   commit;
exception
   --manejo del error
   rollback;
end;
```



Ejemplo:

```
connect medicos/medicos
SQL> update paciente set nombre ='Douglas' where paciente_id = 1;
1 row updated.

--En otra terminal como usuario sys
SQL> connect sys/system as sysdba
Connected.

SQL> select xid as  txn_id, xidusn as segmento_undo, status
     from v$transaction;

              TXN_ID     SEGMENTO_UNDO STATUS
---------------- ------------- ----------------
01001E0074070000       1 ACTIVE
```

- Cuando la transacción concluye, el registro se elimina de `v$transaction`
- Una transacción puede concluir de formas alternas, no únicamente a través de la instrucción `commit/rollback`;
  - Al salir de forma normal de sesión. Por ejemplo, al ejecutar el comando `disconnect` o `exit` de SQL *Plus, al cambiar de sesión, etc; Se aplica un `commit` implícito.
  - Al terminar una sesión de forma abrupta o anormal, sea aplica un `rollback`.

- Al ejecutar una instrucción DDL. Antes de ejecutarla, se aplica un `commit` implícito a la transacción en curso.

Ejemplo:

- Considerar la siguiente tabla de datos:

PROD

PROD_ID	CANTIDAD
1001	30
1002	20
1003	15
1004	5
1005	12

Revisar la siguiente secuencia de operaciones en la que se ilustra la forma en la que se realiza el control transaccional.

Tiempo	Operación	Descripción
0	`commit;`	Se termina la transacción que estaba en curso en la sesión actual.
1	`set transaction name 'T1';`	Se inicia una Txn con el nombre 'cambio_1001'. Notar que a pesar de no especificar la instrucción `set transaction`, se crea una nueva transacción al ejecutar la siguiente instrucción

2	`update prod`
	`set cantidad = 40`
	`where prod_id=1001;`	

3	`savepoint` `after_update_1001;`	Esta instrucción crea un "punto de guardado". La idea es que la Txn puede hacer `rollback` hasta este punto. Es decir, la Txn puede deshacer todos los cambios desde un Tiempo N sin deshacer los cambios previos a esta instrucción.

4	`update prod`
	`set cantidad = 45`
	`where prod_id=1002;`	

5	`savepoint` `after_update_1002;`	
6	`rollback to savepoint` `after_update_1001;`	La Txn hará `rollback` hasta el tiempo t3: • El update realizado en t4 se revierte. • Cuando se hace un `rollback` sobre un `save point` la Transacción no termina, sigue en curso. • Se conserva el save point en t3, pero los subsecuentes son eliminados. Por ejemplo, en este punto, al hacer rollback al savepoint `after_update_1001` el savepoint marcado en t5 se pierde.

		• Los bloqueos posteriores al savepoint `after_update_1001` se liberan, pero se conservan los bloqueos antes de dicho savepoint.
7	`select * from prod;`	Observar los valores actuales: PROD_ID CANTIDAD ---------- ---------- 1001 40 1002 20 1003 15 1004 5 1005 12

8	`update prod`
	`set cantidad = 49`
	`where prod_id =1003;`	

9	`select * from prod;`	Observar los valores actuals PROD_ID CANTIDAD ---------- ---------- 1001 40 1002 20 1003 49 1004 5 1005 12
10	`rollback;`	La transacción T1 revierte todos los cambios y termina.
11	`select * from prod;`	Observar los valores actuales. PROD_ID CANTIDAD ---------- ---------- 1001 30 1002 20 1003 15 1004 5 1005 12
12	`set transaction name 'T2'`	Inicia la transacción 2

13	`update prod`
	`set cantidad = 40`
	`where prod_id=1001;`	

14	`commit;`	La Txn 2 termina y el cambio se hace permanente.

8.6.4. Propiedades ACID de las transacciones.

- Atomicidad
- Consistencia
- Aislamiento
- Durabilidad.

8.6.4.1. Atomicidad.

- Una transacción se maneja como si se tratara de una sola operación. Por lo tanto, o todas las operaciones se concluyen o ninguna.
- Fundamental para garantizar la consistencia de los datos una vez que la Txn concluye.

8.6.4.2. Consistencia

- Capacidad para llevar a la BD de un estado consistente a otro.
- Para realizar esta tarea se requiere de la aplicación de los llamados ***Niveles de aislamiento***.

8.6.4.3. Aislamiento

- Es la propiedad que requiere cada Txn para ver a la BD consistente en cualquier instante de tiempo.
- Una Txn en ejecución no debe revelar sus cambios a otras Txn hasta confirmar dichos cambios.

8.6.4.4. Durabilidad.

- Asegura que una vez que la transacción se confirma, los datos son permanentes "***sin posibilidad"*** de pérdida sin importar fallas posteriores al ejecutar la instrucción `commit`.

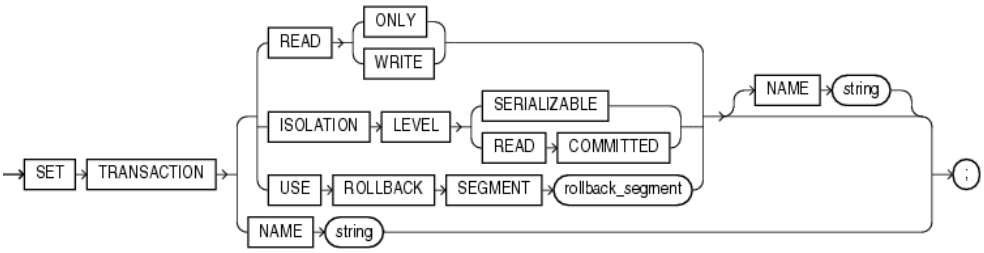
Los niveles de aislamiento representan una de las principales técnicas para implementar estas propiedades, los cuales se definen con base a problemas de consistencia que pueden existir en una base de datos cuando múltiples transacciones ***simultáneas*** intentan consultar o modificar un ***mismo*** dato.

8.6.5. Niveles de aislamiento

- Definidos en el estándar SQL-92
- Permiten implementar principalmente la propiedad de consistencia de una transacción.
- Los niveles de aislamiento se definen con base a 3 problemas que pueden ocurrir durante la ejecución de varias transacciones de forma concurrente:
    - Lecturas sucias
    - Lecturas no repetibles.
    - Lecturas fantasmas.

En Oracle, la instrucción `set transaction` se emplea para definir y configurar el comportamiento de una transacción:

```
set transaction
   { { read { only | write }
     | isolation level
       { serializable | read committed }
     | use rollback segment rollback_segment
     } [ name string ]
   | name string
   } ;
```



Existen diversas técnicas y algoritmos para implementar estos niveles de aislamiento en las bases de datos. El mecanismo más comúnmente empleado es el basado en ***bloqueos***.

8.6.5.1. Lecturas sucias.

- Se refiere a la posibilidad de leer o modificar datos de una Txn que aún no ha concluido.
- T1 modifica un dato el cual es leído por una Txn T2 antes de que T1 haga `commit` o `rollback`. Si Ti hace rollback, T2 tendrá un valor que ***nunca existió*** en la BD.
- Este problema se resuelve con el primer nivel de aislamiento llamado ***Lecturas confirmadas,*** sin embargo, permite la ocurrencia de lecturas no repetibles y lecturas fantasma.

Ejemplo:

- Oracle no permite lecturas sucias. El nivel de aislamiento por default que adquiere una transacción es *Lecturas confirmadas*.

  Asumir las siguientes condiciones iniciales y la siguiente secuencia de eventos generados por 2 transacciones concurrentes Txn1, Txn2 y Txn3

PROD

PROD_ID	CANTIDAD
1001	300
1002	500
1003	700

Tiempo	Txn1	Txn2

0	`update prod`
	`set cantidad = 100`
	`where prod_id = 1001;`	

1	`commit;`	

2		`update prod`
		`set cantidad = 100 + (`
		`  select cantidad`
		`  from prod`
		`  where prod_id=1001)`
		`where prod_id=1001;`
		¿qué valor actualizó esta instrucción?

3	inicia otra transacción: txn3	

	`update prod`
	`set cantidad = 150`
	`where prod_id = 1002;`	

4		update prod set cantidad = 100 + (select cantidad from prod where prod_id=1002) where prod_id=1003; ¿qué valor actualizó esta instrucción?
5	rollback;	
6		update prod set cantidad = 100 + (select cantidad from prod where prod_id=1002) where prod_id=1003; • ¿qué valor actualizó esta instrucción? notar que es la misma que en t4. • ¿qué valor actualizaría si txn3 hubiera hecho commit en lugar de haber hecho rollback ? • ¿qué problema ocurriría si oracle permitiera lecturas sucias?

Respuestas:

- En T2, la subconsulta regresará el valor 100 ya que txn1 ha hecho commit, por lo tanto Txn2 ya puede leer lo que realizó Txn1 , por lo tanto la sentencia update modificará el valor a **200**.
- En T4 la subconsulta regresará el valor 500. A pesar de que Txn3 lo ha modificado con el valor 150, Txn2 no puede ver lo que está haciendo Txn3 porque aún no termina la transacción (ya sea commit o rollback). Por lo tanto, la sentencia update modificará el valor a: 500+100 = **600**
- En T6 se vuelve a ejecutar la sentencia. En esta ocasión, Txn2 ya puede leer los cambios que hizo Txn3 ya que terminó en T5. La subconsulta obtendría nuevamente el valor 500 ya que Txn3 hizo rollback. Por lo tanto, la sentencia update modificará el valor a: 500+100 =**600**.
- Si Txn3 Hubiera hecho commit, la sentencia update modificaría el valor a: 150+100 =**250**.
- Si la BD permitiera lecturas sucias, en T4 se habría leído el valor 150. El problema aquí es que posteriormente Tx3 hace rollback y Txn2 ha leído un dato que ya no existe, provocando una lectura sucia, y por lo tanto inconsistencias graves.

8.6.5.2. Lecturas no repetibles

- T1 lee un dato, t2 modifica o elimina un dato y hace commit. SI T1 vuelve a leer, T1 va a leer un valor diferente o no va a encontrar el dato. Ambas lecturas regresan resultados diferentes cuando deberían regresar el mismo resultado.
- En resumen: Los datos se mueven mientras T1 se está ejecutando haciendo cálculos y "***asumiendo***" que los datos que se leyeron no han cambiado.
- Este problema se resuelve con el segundo nivel de aislamiento llamado ***Lecturas repetibles,*** sin embargo, permite la ocurrencia de lecturas fantasma.

Ejemplo:

Asumir las siguientes condiciones iniciales:

PROD

PROD_ID	CANTIDAD
1001	100

Tiempo	Txn1	Txn2
0	`set transaction isolation level serializable;`	
1	`select cantidad from prod where prod_id = 1001;` ¿Qué valor se obtendrá?	
2		`update prod set cantidad = 35 where prod_id = 1001;`
3	`select cantidad from prod where prod_id = 1001;` ¿Qué valor se obtendrá?	
4		`commit;`
5	`select cantidad from prod where prod_id = 1001;` ¿Qué valor se obtendrá?	
6	`commit;`	
7	`select cantidad from prod where prod_id = 1001;` ¿Qué valor se obtendrá?	

- En T1 se obtendrá el valor 100
- En T3 se obtendrá el valor 100, T2 aún no hace `commit`.
- En T5 se obtendrá el valor 100 a pesar de que T2 hizo `commit`. Esto se debe al nivel de aislamiento establecido en T0. En Oracle solo se soportan 2 niveles: `Read Commited` o `Serializable`.
- En T7 se obtendrá el valor 35 ya que la transacción serializable ha hecho `commit`, el nivel de aislamiento ha concluido.

8.6.5.3. Lecturas fantasmas.

- Una transacción Txn1 realiza una consulta con cierto predicado y obtiene N registros.
- Una transacción Txn2 inserta un nuevo registro que satisface el predicado de Txn1.
- Si Txn1 vuelve a ejecutar la consulta, existirán más registros de los leídos originalmente lo cual puede generar problemas. Ha ocurrido una lectura fantasma.
- Este problema se resuelve con el tercer y último nivel de aislamiento llamado ***Serializable.***

En Oracle se emplea la misma opción: `set transaction isolation level serializable`

Ejemplo:

Considerar las siguientes condiciones iniciales:

PROD

PROD_ID	CANTIDAD
1001	35

Tiempo	Txn1	Txn2
0	`set transaction isolation level serializable;`	
1	`select * from prod where cantidad = 35;` ¿cuántos registros se obtendrán?	
2		`insert into prod values(1006,35);`
3		`commit;`
4	`select * from prod where cantidad = 35;` ¿cuántos registros se obtendrán?	
5	`commit;`	
6	`select * from prod where cantidad = 35;` ¿Cuántos registros se obtendrán?	

- En T1 se obtiene un solo registro (condiciones iniciales).
- En T4 se obtiene nuevamente un solo registro a pesar de que Txn2 ha hecho `commit`, debido a que Txn1 fue marcada como 'serializable' en T0
- En T6 se obtendrán 2 registros ya que Txn1 ha hecho `commit` y por lo tanto ha terminado el nivel de aislamiento establecido.

8.7. CONTROL DE CONCURRENCIA.

A pesar de existir estos niveles de aislamiento pueden ocurrir otros problemas cuando 2 o más transacciones concurrentes intentan acceder a un mismo dato. Considerar los siguientes ejemplos que ilustran estas problemáticas.

Ejemplo 1:

Considerar las siguientes condiciones iniciales:

PROD

PROD_ID	CANTIDAD
1001	5

Suponer la siguiente secuencia de eventos que ocurren entre 2 transacciones Txn1 y Txn2, considerar el nivel de aislamiento por default en Oracle: Read Commited.

Tiempo	Txn1	Txn2
0	`select * from prod where prod_id =1001;`	
1		`select * from prod where prod_id = 1001;` ¿Qué se obtendrá en esta consulta? Seleccionar una respuesta: A. Se obtiene un registro. B. Habrá un error ya que Txn2 está consultando el mismo registro que Txn1 y aún no termina (`commit` o `rollback`).
2	`update prod set cantidad = 50 where prod_id = 1001;` ¿Qué sucederá al intentar ejecutar esta instrucción? A. Sin error, se actualizará un registro. B. Habrá problemas ya que el registro fue leído por Txn2 en T1.	
3		`update prod set cantidad = 20 where prod_id = 1001;` ¿Qué sucederá al intentar ejecutar esta instrucción? A. Sin error, se actualizará un registro. B. Habrá problemas ya que el registro fue actualizado por Txn1 en T2, y Txn1 no ha terminado.
4	`commit;`	
5	¿Qué evento ocurrirá en este tiempo, inmediatamente después de que Txn1 hizo `commit`?	
6		`commit;`
7	`select cantidad from prod where prod_id = 1001;` ¿Qué valor se obtendrá?	
8		`select cantidad from prod where prod_id = 1001;` ¿Qué valor se obtendrá?

- En T1 la consulta se ejecutará de manera normal, se obtendrá un registro. Esto significa que N transacciones pueden leer un mismo registro de manera concurrente. Ambas transacciones leerán el valor 5. Si una de las 2 transacciones modifica el registro, la otra seguirá leyendo el valor 5 mientras no se haga `commit`.

- En T2 la sentencia `update` se ejecuta sin problema alguno. A pesar de que ambas transacciones han leído el registro previamente, Txn1 puede modificar el registro. Txn2 no podrá ver los cambios hasta que Txn1 termine.
- En T3 ¡Habrá problemas! Txn2 intenta modificar el mismo registro que fue modificado por Txn1. Una transacción no puede modificar un dato mientras exista otra que lo esté modificando también. Txn2 podrá modificar hasta que Txn1 termine. No se provoca un error, pero Txn2 se ***bloqueará*** hasta que Txn1 termine. En resumen:
    - Lecturas no bloquean lecturas ni escrituras
    - Escrituras no bloquean lecturas
    - Escrituras bloquean escrituras.
    - En general, las bases de datos emplean el concepto de bloqueo para garantizar un comportamiento adecuado cuando 2 transacciones intentan modificar un mismo dato. Típicamente existen 2 tipos de bloqueos:
        - ***Compartido***: Permite que N transacciones puedan leer un mismo dato de forma concurrente.
        - ***Exclusivo***: Solo una Transacción puede modificar un dato a la vez.
- En T5 Txn2 reanuda su ejecución ya que Txn1 ha terminado haciendo `commit`. Justo después de haber terminado, Txn2 ya puede modificar el mismo registro, por lo tanto, la instrucción `update` que se intentó ejecutar en T3, se ejecuta en T5 sin problema alguno.
- En T7 se obtendrá el valor 20. Observar que el valor que actualizó Txn1 ¡se perdió! Esto podría representar un problema de inconsistencia el cual se revisará en el siguiente ejemplo llamado ***Lost Update***. A pesar de existir niveles de aislamiento, esta situación puede presentarse.
- En T8 se obtiene de manera similar el valor 20 ya que ambas transacciones han terminado.

Ejemplo 2:

Considerar la siguiente tabla que lleva el control de los asientos ocupados en un concierto. Cuando un asiento se ocupa, el valor del campo `ocupado` se establece en 1 y se guarda el nombre del ocupante. Considerar las siguientes condiciones iniciales.

CONCIERTO

NUM_ASIENTO	OCUPADO	NOMBRE
1	0	
2	0	
3	0	

Suponer la siguiente secuencia de eventos que ocurren entre 2 transacciones Txn1 y Txn2 que representan a 2 clientes que intentan comprar el asiento 1 desde un sitio Web que tiene problemas de concurrencia. Considerar el nivel de aislamiento por default en Oracle: Read Commited.

Tiempo	Txn1	Txn2

0	El cliente 1 consulta para verificar si el asiento 1 está disponible:
	`select ocupado`
	`from concierto`
	`where num_asiento=1;`
	¿Qué se obtendrá?	

1		El cliente 2 consulta para verificar si el asiento 1 está disponible:
		`select ocupado`
		`from concierto`
		`where num_asiento=1;`
		¿Qué se obtendrá?

2	El cliente 1 decide comprar el asiento por lo que se ejecuta la siguiente sentencia:
	`update concierto`
	`set ocupado = 1,`
	`nombre ='cliente1'`
	`where num_asiento=1;`	

3	La operación se confirma `commit;`	

4		El cliente 2 decide comprar el asiento por lo que se ejecuta la siguiente sentencia:
		`update concierto`
		`set ocupado = 1,`
		`nombre ='cliente2'`
		`where num_asiento=1;`
		¿Qué ocurrirá aquí?
		A. Habrá problema ya que este registro fue actualizado por Txn1.
		B. La sentencia se ejecutará sin problema alguno.

5	El cliente 1 consulta su compra para confirmar la operación.
	`select *`
	`from concierto`
	`where num_asiento = 1;`
	¿Qué se obtendrá?	

6		La operación se confirma `commit;`

7		El cliente 2 consulta su compra para confirmar la operación.
		`select *`
		`from concierto`
		`where num_asiento = 1;`
		¿Qué se obtendrá?

- Tanto en T1 como en T2 se obtendrá el valor 0. Esto significa que el asiento 1 está disponible.
- En T2 el cliente1 decide comprar el boleto, por lo que la sentencia `update` se ejecuta sin problema alguno.
- En T4 el cliente2 también decide comprar el mismo asiento ya que el asiento está disponible. Debido a que Txn1 hizo `commit` en T3, la instrucción `update` se ejecuta sin problema alguno.
- En T5 el cliente1 obtendrá el siguiente registro en el que se observa que el boleto ha sido vendido a él. Txn1 no puede ver el cambio que hizo Txn2 ya que Txn2 aún no hace `commit`.

```
NUM_ASIENTO OCUPADO    NOMBRE
----------- ---------- --------------------
1           1          cliente1
```

- En T7 el cliente2 consulta su compra y confirma que el asiento fue vendido a él ya que Txn2 hizo `commit` en T6 en el que se han sobrescrito los datos que había actualizado Txn1: ***Lost update***.

```
NUM_ASIENTO OCUPADO    NOMBRE
----------- ---------- --------------------
1           1          cliente2
```

¿Qué problemas existen en esta secuencia?

- Adicional al problema ***lost update*** descrito en el punto anterior, existe uno más grave: Ambos clientes fueron capaces de comprar el mismo asiento y ambos pudieron confirmar que les fue vendido de forma 'normal'. ¿Qué va a pasar cuando ambos clientes lleguen a ocupar su asiento el día del concierto?

¿Cómo solucionar este problema?

Existen 2 técnicas empleadas:
- Control de concurrencia pesimista.
- Control de concurrencia optimista.

8.7.1. Control de concurrencia pesimista.

En esta estrategia, la única forma de evitar el problema anterior es serializar por completo las transacciones anteriores, es decir, si una transacción comienza con el proceso de compra de un asiento en particular, ninguna otra transacción podrá iniciar el mismo proceso. Esto incluye tanto sentencias `select` como sentencias DML.

En Oracle se emplea la cláusula `select for update` para implementar este comportamiento. La solución al ejercicio anterior se muestra en la siguiente secuencia de eventos:

- Actualizar la tabla a su estado inicial para comprobar los resultados.

Tiempo	Txn1	Txn2

0	El cliente 1 consulta para verificar si el asiento 1 está disponible:
	`select ocupado`
	`from concierto`
	`where num_asiento=1`
	`for update;`
	¿Qué se obtendrá?	

1		El cliente 2 consulta para verificar si el asiento 1 está disponible:
		`select ocupado`
		`from concierto`
		`where num_asiento=1`
		`for update;`
		¿Qué se obtendrá?

2	El cliente 1 decide comprar el asiento por lo que se ejecuta la siguiente sentencia:	

	`update concierto`
	`set ocupado = 1,`
	`nombre ='cliente1'`
	`where num_asiento=1;`	

3	La operación se confirma `commit;`	
4.	¿Qué evento ocurrirá aquí?, justo después de que Txn1 ha hecho `commit`?	

- En T0 la Txn1 hace uso de la cláusula `select for update`. La BD establecerá un bloqueo exclusivo sobre el registro obtenido por la sentencia `select` de tal forma que ninguna otra transacción podrá ejecutar operaciones de lectura o escritura. En este punto se obtendrá un registro, indicando que el asiento 1 está disponible.
- En T1, el cliente2 intentará ejecutar la misma sentencia. Debido al uso de `select for update`, la sesión del cliente 2 se bloqueará. Existen las cláusulas `wait <seconds>` y `nowait` que indican el tiempo máximo que Txn2 va a esperar para que el registro sea liberado. Si se especifica `nowait`, Txn2 no se bloqueará, pero enviará un error indicando que el registro está bloqueado. Por default, la Txn2 se bloqueará hasta que Txn1 termine.

```
select * from concierto where num_asiento = 1 for update nowait;
select * from concierto where num_asiento = 1 for update nowait
*
ERROR at line 1:
ORA-00054: recurso ocupado y obtenido con NOWAIT especificado o timeout vencido
```

- En T3 Txn1 termina. Esto implica que en T4 Txn2 será desbloqueada y la instrucción `select` será ejecutada mostrando lo siguiente:

```
 NUM_ASIENTO OCUPADO    NOMBRE
----------- ---------- --------------------
1           1          cliente1
```

- Como se puede observar, aquí Txn2 se ha percatado que el asiento ha sido comprado por lo que el proceso con Txn2 ya no continuará.

8.7.1.1. Ventajas del control pesimista:

- Su implementación es sencilla, únicamente se tiene que hacer uso de la cláusula `select for update`.

8.7.1.2. Desventajas del control pesimista:

- Las transacciones se vuelven seriales. Esta condición puede representar un problema de desempeño ya que limita la capacidad de concurrencia que ofrecen las bases de datos. El control de concurrencia optimista permite resolver este problema.

8.7.2. Control de concurrencia optimista.

- Se basa sobre el principio que dice: “Vamos a confiar en que no van a existir situaciones como la del ejemplo de compra de asientos”. Se asume que esta situación tiene una probabilidad baja de ocurrir. Se decide no serializar.

- Para verificar que esta condición de error no ocurrió, se decide aplicar una validación justo antes de actualizar un registro al final de la transacción. Si se detecta que algo cambió, se lanza un error que deberá ser manejado por la aplicación que accede a la base de datos.

La solución se muestra en la siguiente secuencia de eventos:

Tiempo	Txn1	Txn2

0	El cliente 1 consulta para verificar si el asiento 1 está disponible:
	`select ocupado`
	`from concierto`
	`where num_asiento=1;`	

1		El cliente 2 consulta para verificar si el asiento 1 está disponible:
		`select ocupado`
		`from concierto`
		`where num_asiento=1;`

2	El cliente 1 decide comprar el asiento por lo que se ejecuta la siguiente sentencia:
	`update concierto`
	`set ocupado = 1,`
	`nombre ='cliente1'`
	`where num_asiento=1`
	`and ocupado = 0;`
	¿Qué sucederá aquí?	

3	La operación se confirma `commit;`	

4		El cliente 2 decide comprar el asiento por lo que se ejecuta la siguiente sentencia:
		`update concierto`
		`set ocupado = 1,`
		`nombre ='cliente2'`
		`where num_asiento=1`
		`and ocupado = 0;`
		¿Qué ocurrirá aquí?

5	El cliente 1 consulta su compra para confirmar la operación.
	`select *`
	`from concierto`
	`where num_asiento = 1;`
	¿Qué se obtendrá?	

- Observar que, con esta técnica, las transacciones no se bloquean, únicamente se emplea el nivel de aislamiento por default: Read Commited.
- Observar la condición agregada en T2, llamada sentencia ***update condicionada***. Justo antes de aplicar el cambio en la BD, se verifica que el asiento siga estando disponible. Solo en ese caso la sentencia `update` se ejecuta. En T2 se aplicará el cambio ya que ninguna otra Txn ha actualizado el valor del campo `ocupado`, sigue conservando su valor = 0 cuando este fue leído por primera vez por Txn1.

- En T4, Txn2 intenta hacer lo mismo. En esta ocasión la sentencia `update` no hará actualización alguna ya que la condición (ocupado =0) ya no se cumple debido a que Txn1 ha modificado su valor. Esta condición es detectada por la aplicación o software que accede a los datos y puede tomar ciertas acciones. En este caso, la aplicación detectaría que Txn2 no pudo actualizar el cambio solicitado y enviaría un mensaje de error al cliente 2 indicándole que alguien más le ha ganado el asiento.
- Esta técnica es la que comúnmente se emplea en sistemas que pueden tener competencia de transacciones para actualizar datos sobre un mismo conjunto de registros, como son: asignación de asientos, venta de boletos únicos, etc.

8.7.2.1. Ventajas del control de concurrencia optimista:

- Se elimina el problema de serialización de transacciones, no existe problema de desempeño.
- Representa la opción comúnmente empleada en sistemas de alta concurrencia.

8.7.2.2. Desventajas del control de concurrencia optimista:

- El software que accede a la BD tiene que incluir la programación necesaria para detectar el problema de concurrencia a través del uso de una sentencia `update` condicionada. Cada caso es particular por lo que la condición es diferente en cada caso. Se debe realizar un análisis para diseñar la condición de forma correcta.
- El software también debe incluir la lógica a ejecutar cuando la sentencia `update` condicionada no actualice registro alguno. En algunos casos, el sistema debe notificar el error al usuario final, en otros casos, el proceso se vuelve a ejecutar para refrescar o actualizar los valores leídos que fueron modificados por otras transacciones y se vuelve a ejecutar la sentencia `update` condicionada hasta que la sentencia sea exitosa.