Transacciones en SQL

Bases de Datos

2017

Conceptos Generales

Transacciones Implícitas/Explicitas

- ISO SQL: cualquier comando SQL al comienzo de una sesión o inmediato posterior al fin de una transacción comienza automáticamente una nueva transacción (DB2 y Oracle)

Transacciones Implícitas/Explicitas

- ISO SQL: cualquier comando SQL al comienzo de una sesión o inmediato posterior al fin de una transacción comienza automáticamente una nueva transacción (DB2 y Oracle)
- SQL Server, MySQL/InnoDB, PostgreSQL funcionan por defecto en modo AUTOCOMMIT
  - MySQL/InnoDB: SET AUTOCOMMIT = 0|1
  - SQL Server: SET IMPLICIT_TRANSACTIONS [*ON*|*OFF*]
  - PostgreSQL: SET AUTOCOMMIT = [*ON*|*OFF*]

Transacciones

```
BEGIN/START TRANSACTION;
COMMIT;
ROLLBACK;
```

Transacciones

```
BEGIN/START TRANSACTION;
COMMIT;
ROLLBACK;
```

```
INSERT INTO Tabla (id, s) VALUES (1, 'primero');
INSERT INTO Tabla (id, s) VALUES (2, 'segundo');
INSERT INTO Tabla (id, s) VALUES (3, 'tercero');
SELECT * FROM Tabla ;

ROLLBACK;
SELECT * FROM Tabla ;
```

Transacciones

```
BEGIN/START TRANSACTION;
COMMIT;
ROLLBACK;
```

```
INSERT INTO Tabla (id, s) VALUES (1, 'primero');
INSERT INTO Tabla (id, s) VALUES (2, 'segundo');
INSERT INTO Tabla (id, s) VALUES (3, 'tercero');
SELECT * FROM Tabla ;

ROLLBACK;
SELECT * FROM Tabla ;
```

AUTOCOMMIT

Una sentencia SQL enviada a la base de datos en modo AUTOCOMMIT no puede ser desecha (rollback)

Precaución

Supongamos lo siguiente:

```
CREATE TABLE Cuentas(
    [ctaID] [int] NOT NULL PRIMARY KEY,
    [balance] [decimal](11, 2) NULL
    )

ALTER TABLE Cuentas WITH CHECK ADD CHECK  (([balance]>=(0)))
```

Precaución

Supongamos lo siguiente:

```
CREATE TABLE Cuentas(
    [ctaID] [int] NOT NULL PRIMARY KEY,
    [balance] [decimal](11, 2) NULL
    )

ALTER TABLE Cuentas WITH CHECK ADD CHECK  (([balance]>=(0)))

BEGIN TRANSACTION;
UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
COMMIT;
```

Precaución

Supongamos lo siguiente:

```
CREATE TABLE Cuentas(
    [ctaID] [int] NOT NULL PRIMARY KEY,
    [balance] [decimal](11, 2) NULL
    )

ALTER TABLE Cuentas WITH CHECK ADD CHECK  (([balance]>=(0)))

BEGIN TRANSACTION;
UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
COMMIT;
```

SQL Server

```
SET XACT_ABORT { ON | OFF }
```

PostgreSQL

Siempre aborta ante un error.

Precaución

```
BEGIN TRANSACTION;
UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
IF @@error <> 0 ROLLBACK
ELSE
BEGIN
  UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
  COMMIT;
END
```

Precaución

```
BEGIN TRANSACTION;
UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
IF @@error <> 0 ROLLBACK
ELSE
BEGIN
  UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
  COMMIT;
END
```

```
SET XACT_ABORT ON
BEGIN TRANSACTION;
   UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
   UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
COMMIT;
```

Precaución

```
BEGIN TRANSACTION;
UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
IF @@error <> 0 ROLLBACK
ELSE
BEGIN
  UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
  COMMIT;
END
```

```
SET XACT_ABORT ON
BEGIN TRANSACTION;
   UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
   UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
COMMIT;
```

```
BEGIN TRY
   BEGIN TRAN
        UPDATE Cuentas SET balance = balance - 100 WHERE ctaID = 101;
        UPDATE Cuentas SET balance = balance + 100 WHERE ctaID = 102;
   COMMIT TRAN
END TRY
BEGIN CATCH
   ROLLBACK TRAN
   RAISERROR('Se que se ha producido un error.', 16, 1);
END CATCH
```

- (ISO-89)SQLCODE $\Rightarrow$(ISO-92)SQLSTATE
- 5 Caracteres :
  - Clase 2 Caracteres
    - 0000 Exito
    - 01 Warning
    - 02 No Data
    - 07 Dynamic SQL ERROR
    - 08 Connection Exception
    - 22 Data Exception
    - 40 Transaction Rollback
  - Subclase 3 Caracteres

Niveles de Aislamiento

Niveles de aislamiento

El nivel de aislamiento controla el grado en que una transacción dada está expuesta a la acciones de otras transacciones ejecutándose simultáneamente.

Problemas:

- Lost Update
- Dirty Read
- Non-Repeatable Read (Fuzzy Read)
- Phantom Read

Niveles de Aislamiento

Isolation level	Dirty read	Nonrepeatable read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed *RCSI (SQLServer)*	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable *Snapshot (SQLServer)*	No	No	No

- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED;

Niveles de Aislamiento

Isolation level	Dirty read	Nonrepeatable read	Phantom
Read uncommitted	Yes	Yes	Yes
Read committed *RCSI (SQLServer)*	No	Yes	Yes
Repeatable read	No	No	Yes
Serializable *Snapshot (SQLServer)*	No	No	No

SqlServer PostgreSQL Oracle

MySQL InnoDB

- SET TRANSACTION ISOLATION LEVEL READ UNCOMMITED;

PostgreSQL

PostreSQL: MVCC

MVCC

PostgreSQL usa una implementación particular de control de concurrencia multiversión y locks

- Cada transacción ve un *snapshot* a partir de su comienzo (*timestamp*). No importa lo que otras transacciones hagan mientras esta en ejecución.
- Las lecturas no bloquean a las escrituras ni las escrituras bloquean a las lecturas

Niveles de Aislamiento

MVCC

En PostgreSQL se pueden definir los 4 niveles del standard SQL pero internamente los soportados son 3: *Read Committed, Repeatable Read y Serializable*.

- *Read uncommitted* en realidad se comporta como *Read Committed*
- *Repeatable Read*. En el ISO SQL se permite *Phantom Read* pero PG no lo permite.
- *Serializable*: utiliza una técnica denomiada Serializable Snapshot Isolation. Es similar *Repeatable Read* pero monitorea si se produce algún comportamiento que pueda violar la *serializabilidad*.

MVCC

XID: identificador único de cada transacción, equivale al *timestamp*.
Tuple headers:

- xmin: XID de la transacción que inserto la fila
- xmax: XID de la transacción que borro o actualizo la fila
- forward link: link a la nueva versión de la misma fila lógica si existiera.

Visibilidad

Para un insert

Si el XID de la transacción es mayor que el xmin de una fila *commited*, se permite leer.
Si el XID de la transacción es menor que el xmin de una fila *commited* entonces dependerá del nivel de aislamiento.

- Para READ COMMITED toma el *timestamp* del comienzo de la sentencia.
- Para REPEATABLE READ o SERIALIZABLE todas las lecturas son relativas al comienzo de la transacción

Visibilidad

Para un delete

Si el XID de la transacción es mayor que el xmax de una fila *commited*, **no se permite leer**.
Si el XID de la transacción es menor que el xmax de una fila *commited* entonces dependerá del nivel de aislamiento.

- Para READ COMMITED toma el *timestamp* del comienzo de la sentencia.
- Para REPEATABLE READ o SERIALIZABLE todas las lecturas son relativas al comienzo de la transacción

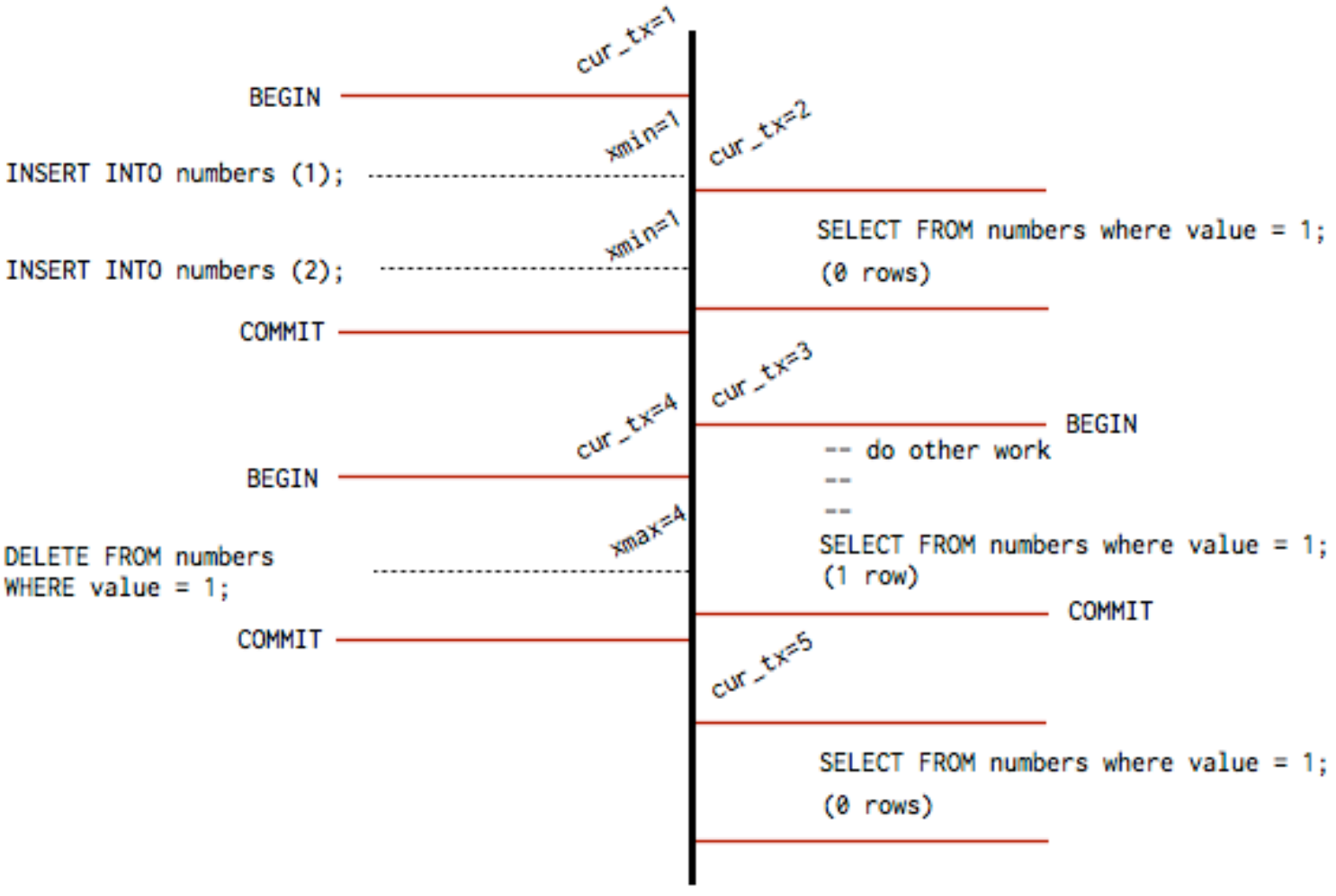
Visibilidad

Para un update

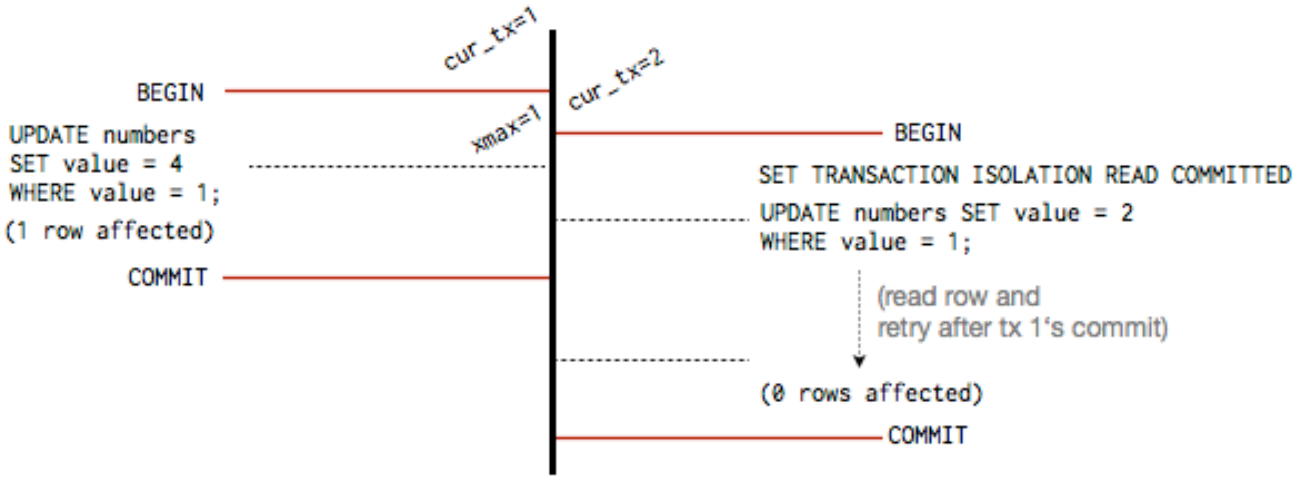
- Se crea una nueva fila y se pone el XID de la transacción en xmax
- Si no esta *committed* entonces una transacción con XID mayor a xmin va a leer la versión dependiendo del grado de aislamiento.
  - Si es READ COMMITED leera la versión vieja hasta que la transacción que actualizo haga un commit.
  - Si es REPEATABLE READ o SERIALIZABLE vera la versión vieja.

Ejemplo PostgreSQL

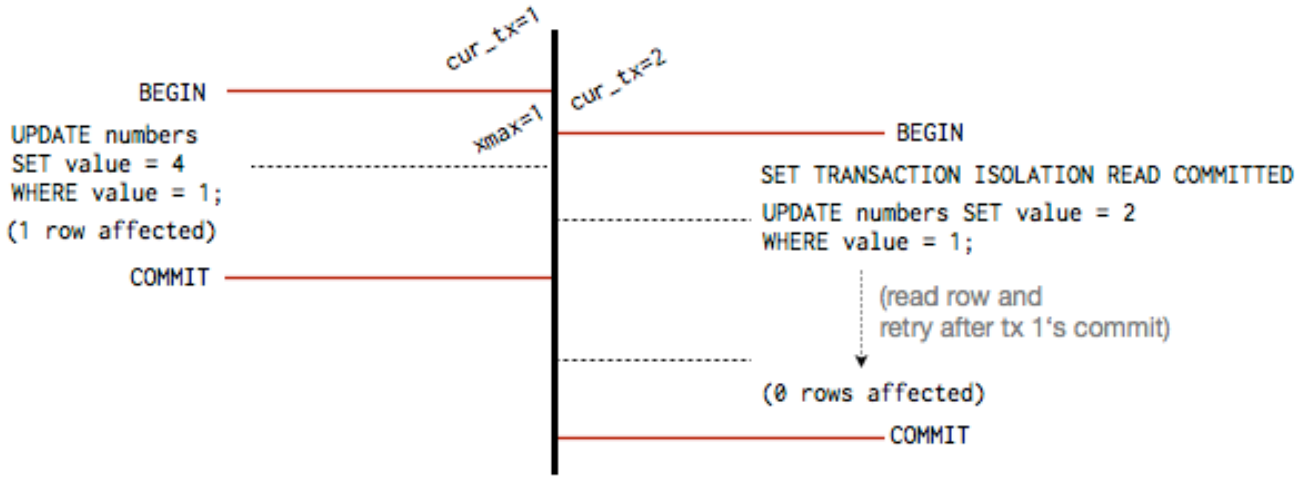
```
CREATE TABLE numbers (value int);
```

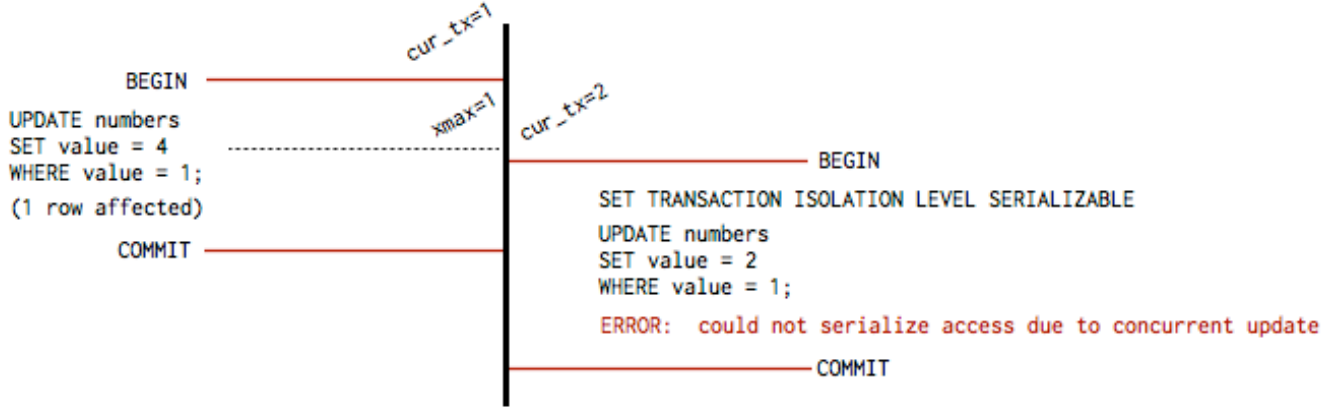


PostgreSQL MVCC Update sobre la misma fila



PostgreSQL MVCC Update sobre la misma fila





SQL Server

Niveles de Aislamiento SQLServer

- Read Commited (Optimista o Pesimista, RCSI o RCI):
  - SET READ_COMMITTED_SNAPSHOT ON (multi-versión)
    - Versiones de filas (row versioning)
  - SET READ_COMMITTED_SNAPSHOT OFF (default)
    - Shared locks (read lock) son levantados inmediatamente.

Time	Transaction 1	Transaction 2
1	BEGIN TRAN UPDATE Production.Product SET ListPrice = 10.00 WHERE ProductID = 922;	BEGIN TRAN
2		SELECT ListPrice FROM Production.Product WHERE ProductID = 922; -- SQL Server returns 8.89
3	COMMIT TRAN	
4		SELECT ListPrice FROM Production.Product WHERE ProductID = 922; -- SQL Server returns 10.00
5		COMMIT TRAN

Repeatable Read

Locks en repeateable red

Shared locks (read lock) se mantienen durante toda la transacción

Repeatable Read

Locks en repeateable red

Shared locks (read lock) se mantienen durante toda la transacción

Write Locks

Los **locks exclusivos** siempre deben mantenerse hasta el final de una transacción, sin importar el nivel de aislamiento o modelo de concurrencia, de modo que una transacción pueda revertirse si es necesario

Niveles de Aislamiento SQLServer: Snapshot (SI)

Optimista

Permite a los procesos leer viejas versiones de los datos que fueron *commited* si la versión actual esta *locked*.

```
ALTER DATABASE databasename SET ALLOW_SNAPSHOT_ISOLATION ON;
```

Niveles de Aislamiento SQLServer: Snapshot (SI)

Optimista

Permite a los procesos leer viejas versiones de los datos que fueron *commited* si la versión actual esta *locked*.

```
ALTER DATABASE databasename SET ALLOW_SNAPSHOT_ISOLATION ON;
```

Atención

No es equivalente a serializable. Pueden haber dos transacciones que se ejecuten simultáneamente y produzcan un resultado imposible en una ejecución serial. Aunque evita los mismos problemas

Niveles de Aislamiento SQLServer: Snapshot (SI)

Optimista

Permite a los procesos leer viejas versiones de los datos que fueron *commited* si la versión actual esta *locked*.

ALTER DATABASE databasename SET ALLOW_SNAPSHOT_ISOLATION ON;

Atención

No es equivalente a serializable. Pueden haber dos transacciones que se ejecuten simultáneamente y produzcan un resultado imposible en una ejecución serial. Aunque evita los mismos problemas

Time	Transaction 1	Transaction 2
1	*USE pubs;* *SET TRANSACTION ISOLATION LEVEL SNAPSHOT;* *DECLARE @price money;* *BEGIN TRAN*	*USE pubs;* *SET TRANSACTION ISOLATION LEVEL SNAPSHOT;* *DECLARE @price money;* *BEGIN TRAN*
2	*SELECT @price = price* *FROM titles* *WHERE title_id = 'BU1032';*	*SELECT @price = price* *FROM titles* *WHERE title_id = 'PS7777';*
3	*UPDATE titles* *SET price = @price* *WHERE title_id = 'PS7777';*	*UPDATE titles* *SET price = @price* *WHERE title_id = 'BU1032';*
4	*COMMIT TRAN*	*COMMIT TRAN*

Snapshot (SI)

Time	Transaction 1	Transaction 2
1	BEGIN TRAN	
2	UPDATE Production.Product SET ListPrice = 12.00 WHERE ProductID = 922;	SET TRANSACTION ISOLATION LEVEL SNAPSHOT
3		BEGIN TRAN
4		SELECT ListPrice FROM Production.Product WHERE ProductID = 922; -- SQL Server returns 10.00 -- This is the beginning of -- the transaction
5	COMMIT TRAN	
6		SELECT ListPrice FROM Production.Product WHERE ProductID = 922; -- SQL Server returns 10.00 -- Return the committed -- value as of the beginning -- of the transaction
7		COMMIT TRAN
		SELECT ListPrice FROM Production.Product WHERE ProductID = 922; -- SQL Server returns 12.00

Figura tomada de: "*Microsoft SQL 2012 Internals*" -Kalen Delaney et. al.

Niveles de Aislamiento SQLServer

```
set transaction isolation level SERIALIZABLE/SNAPSHOT

begin tran

update marbles set color = 'White' where color = 'Black'
```

Niveles de Aislamiento SQLServer

```
set transaction isolation level SERIALIZABLE/SNAPSHOT

begin tran

update marbles set color = 'White' where color = 'Black'
```

```
set transaction isolation level SERIALIZABLE/SNAPSHOT

begin tran

update marbles set color = 'Black' where color = 'White'

commit tran
```

Niveles de Aislamiento SQLServer

```
set transaction isolation level SERIALIZABLE/SNAPSHOT

begin tran

update marbles set color = 'White' where color = 'Black'
```

```
set transaction isolation level SERIALIZABLE/SNAPSHOT

begin tran

update marbles set color = 'Black' where color = 'White'

commit tran
```

```
commit tran

select * from marbles
```

Niveles de Aislamiento SQLServer: Serializable

Pesimista

Requiere que se realice *lock* sobre datos que han sido leídos y también sobre datos que no existen

- Key-range lock: requiere índice sobre la columna.
- Table-lock

```
SELECT * FROM Personas WHERE CodigoPostal BETWEEN 'C1000AAA' AND  'C1000ZZZ'
```

SQL server Niveles de Aislamiento

Isolation level	Dirty read	Nonrepeatable read	Phantom	Concurrency control
Read Uncommitted	Yes	Yes	Yes	Pessimistic
Read Committed (locking)	No	Yes	Yes	Pessimistic
Read Committed (snapshot)	No	Yes	Yes	Optimistic
Repeatable Read	No	No	Yes	Pessimistic
Snapshot	No	No	No	Optimistic
Serializable	No	No	No	Pessimistic

Figura tomada de: "*Microsoft SQL 2012 Internals*" -Kalen Delaney et. al.

Bibliografía

- ***Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery*** (The Morgan Kaufmann Series in Data Management Systems) Gerhard Weikum y Gottfried Vossen
- ***Microsoft SQL 2012 Internals***. Kalen Delaney y otros
- ***Fundamentos de Bases de Datos*** Abraham Silberschatz, Henry F. Korth y S Sudarshan
- ***Serializable Isolation for Snapshot Databases*** Michael J. Cahill, Uwe Röhm, and Alan D. Fekete. 2008. ACM SIGMOD international conference on Management of data (SIGMOD '08)
- Mimer SQL 10.0 Technical Description
- ***A critique of ANSI SQL isolation levels*** H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, , and P. O'Neil Proceedings of ACM SIGMOD International Conference on Management of Data,1995