Lecture 15: Concurrency Control in SQL

EGCI 321: DATABASE SYSTEM (WEEK 10)

Outline

1.Concurrency Control

- Abort and Commit
- Isolation Levels

2.Locking

- Shared Lock
- Updated Lock
- Exclusive Lock

Problems Caused by Failures

Update all account balance at a bank branch

Accounts(Anum, Cid, Branchld, Balance)

update Accounts

set Balance = Balance * 1.05

where Branchld = 12345

Problem

If the system crashes while processing this update, some, but, not all, tuples with **BranchID = 12345** may have been updated

EGCI321

Another Failure-Related Problem

Transfer money between accounts:

update Accounts

set Balance = Balance – 100

where Anum = 8888

update Accounts

set Balance = Balance + 100

where Anum = 9999

Problem

If the system fails between these updates, money may be withdrawn but not redeposited.

Abort and COMMIT

A transaction may terminate in one of two ways:

- When a transaction commits, any updates it made become durable, and they become visible to other transactions. A COMMIT it the "all" in "all-ornothing" execution.
- When a transaction aborts, any updates it may have made are undone (erased), as if the transaction never ran at all. An abort is the "nothing" in "all-or-nothing" execution.

A transaction that has started but has not yet aborted or committed is said to be *active*

Abort and COMMIT

Example: Abort (SQL called ROLLBACK)

(At 2nd statement, try again without "with (Nolock)")

1st Query Statement START TRANSACTION;

USE concurrency
UPDATE Balance
SET Balance = 2000
WHERE AccountID = 2

SELECT SLEEP(10);

ROLLBACK; SELECT * FROM Balance 2nd Query Statement
USE concurrency
SELECT *
FROM Balance

	AccountID	Name	Balance
1	1	Bob	1000
2	2	Mary	800
3	3	Sam	800
4	4	Jame	1200
5	5	June	1400

	AccountID	Name	Balance
1	1	Bob	1000
2	2	Mary	2000
3	3	Sam	800
4	4	Jame	1200
5	5	June	1400

Abort and COMMIT

Example: COMMIT;

(At 2nd statement, try again without "with (Nolock)")

1st Query Statement COMMIT;;

USE concurrency
UPDATE Balance
SET Balance = 2000
WHERE AccountID = 2

SELECT SLEEP(10);

COMMIT; SELECT * FROM Balance **2nd Query Statement USE** concurrency

SELECT *

FROM Balance

	AccountID	Name	Balance
1	1	Bob	1000
2	2	Mary	2000
3	3	Sam	800
4	4	Jame	1200
5	5	June	1400

Another Concurrency Problem

Application 1:

select balance into: balance

from Accounts

where Anum = 8888

compute :newbalance using :balance

update Accounts

set Balance = :newbalance

where Anum = 8888

Application 2: same as Application 1

Problem

If the applications run concurrently, one of the updates may be "lost"

EGCI321

SQL Isolation Levels

- SQL allows the serializability guarantee to be relaxed, if necessary
- For each transaction, it is possible to specify an isolation level.
- For isolation levels are supported, with the highest being serializability:

Level 0 (Read UnCOMMITTED): transaction may see UNCOMMITTED updates **Level 1 (Read COMMITTED):** transaction sees only COMMITRED changes, but non-repeatable reads are possible

Level 2 (Repeatable Read): reads are repeatable, but "phantoms" are possible Level 3 (Serializability)

Non-Repeatable Reads

Application 1:

update Employee

set Salary = Salary +1000

where WorkDept = 'D11'

Application 2:

select *

from Employee

where WorkDept = 'D11'

select *

from Employee

where Lastname Like 'A%'

Problem

If there are employees in D11 with surnames that begin with "A", Application 2's queries may see them with different salaries.

Phantoms

```
Application 1:
insert into
                  Employee
        ('000123', 'Sheldon', 'Q', 'Jetstream', 'D11', '7777', '05/01/00', 520000.00)
Application 2
select
from
        Employee
        WorkDept = 'D11'
where
select
from
        Employee
        Salary > 5000
where
```

Problem

Application 2's second query may see Sheldon Jetstream, even though its first query does not.

Isolation level	Dirty Reads/ Lost-Update	Non-repeatable reads	Phantom reads
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SNAPSHOT	No	No	No
SERIALIZABLE	No	No	No

Isolation Level 0: Read UnCOMMITTED (Dirty Read)

1st Query Statement

SELECT val FROM tbl;

USE concurrency;
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
START TRANSACTION;
UPDATE tbl SET val = -1;
SELECT SLEEP(5);
ROLLBACK;

2nd Query Statement

USE concurrency;
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
START TRANSACTION;
SELECT val FROM tbl
COMMIT;

Result:

val

1

Run 2nd Query Statement again

val

-1

Isolation Level 1: READ COMMITTED

1st Query Statement

USE concurrency;
SET SESSION TRANSACTION ISOLATION LEVEL
READ COMMITTED;
START TRANSACTION;
UPDATE tbl SET val = -1;
SELECT SLEEP(5);
ROLLBACK;
SELECT val FROM tbl;

2nd Query Statement

USE concurrency;
SET SESSION TRANSACTION ISOLATION LEVEL READ
COMMITTED;
START TRANSACTION;
SELECT val FROM tbl
COMMIT;

Result:

val

1

Result:

val

1

Run 2nd Query Statement again

val

1

Problem: NON-REPEATABLE READ

-1

```
1<sup>st</sup> Query Statement
                                                           2<sup>nd</sup> Query Statement
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
                                                           START TRANSACTION;
                                                           UPDATE tbl set val = -1
GO
START TRANSACTION;
                                                           COMMIT;
SELECT * FROM tbl
SELECT SLEEP(5);
SELECT * FROM tbl
COMMIT; TRAN repeatRead
Result:
       val
      1
      val
```

Isolation Level 2: REPEATABLE READ

1

```
1<sup>st</sup> Query Statement
                                                              2<sup>nd</sup> Query Statement
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
                                                              START TRANSACTION;
START TRANSACTION;
                                                              UPDATE tbl set val = -1;
SELECT * FROM tbl;
                                                              COMMIT;
                                                              SELECT * FROM tbl;
SELECT SLEEP(5);
SELECT * FROM tbl;
COMMIT;
SELECT * FROM tbl;
Result:
                                                              id
                                                                      val
```

Isolation Level 3: SERIALIZABLE

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
START TRANSACTION; pessimisticTran
SELECT * FROM tbl;

SELECT SLEEP(5);
UPDATE tbl
SET val = 3
WHERE id = 1;
COMMIT;
SELECT * FROM tbl;
```

```
2<sup>nd</sup> Query Statement
```

```
START TRANSACTION;

UPDATE tbl

SET val = 4

WHERE id = 1;

COMMIT;

SELECT * FROM tbl;
```

```
Result:
id val
------
1 4
```

Strict Two-Phase Locking

The rules:

- 1. Before a transaction may read or write an object, it must have a lock on the object.
 - A shared lock is required to read an object
 - An exclusive lock is required to write an object
- 2. Two or more transactions may not hold locks on the same object unless all hold shared locks.
- 3. A transaction may not release any locks until it COMMIT;s (or aborts).

If all transactions use strict two-phase locking, the execution history is guaranteed to be serializable.

	Existing granted mode					
Requested mode	IS	S	U	IX	SIX	X
Intent shared (IS)	Yes	Yes	Yes	Yes	Yes	No
Shared (S)	Yes	Yes	Yes	No	No	No
Update (U)	Yes	Yes	No	No	No	No
Intent exclusive (IX)	Yes	No	No	Yes	No	No
Shared with intent exclusive (SIX)	Yes	No	No	No	No	No
Exclusive (X)	No	No	No	No	No	No

Key

,			
N	No Conflict	SIU	Share with Intent Update
I	Illegal	SIX	Shared with Intent Exclusive
C	Conflict	UIX	Update with Intent Exclusive
		BU	Bulk Update
NL	No Lock	RS-S	Shared Range-Shared
SCH-S	Schema Stability Locks	RS-U	Shared Range-Update
SCH-M	Schema Modification Locks	RI-N	Insert Range-Null
S	Shared	RI-S	Insert Range-Shared
U	Update	RI-U	Insert Range-Update
X	Exclusive	RI-X	Insert Range-Exclusive
IS	Intent Shared	RX-S	Exclusive Range-Shared
IU	Intent Update	RX-U	Exclusive Range-Update
IX	Intent Exclusive	RX-X	Exclusive Range-Exclusive

Shared Locks (s)

1st Query Statement

START TRANSACTION; USE concurrency SELECT SLEEP(10);

SELECT AddressID, AddressLine1, AddressLine2, City **FROM** dbo.Address WITH (HOLDLOCK) **WHERE** AddressID = 1

SELECT resource_type, request_mode, resource_description **FROM** sys.dm_tran_locks **WHERE** resource_type <> 'DATABASE'

ROLLBACK;

	AddressID	AddressLine1	AddressLine2	City
1	2	9833 Mt. Dias Blv.	NULL	Bothell
	resource_typ	e request_mode	resource_description	
1	OBJECT	IS		
2	PAGE	IS	1:9312	
3	KEY	S	(61a06abd401c	:)

2nd Query Statement

START TRANSACTION;

USE concurrency

UPDATE dbo.Address

SET AddressLine2 = 'Test Address 2'

WHERE AddressID = 1

SELECT AddressID, AddressLine1,

AddressLine2, City

FROM dbo.Address

WHERE AddressID < 2

ROLLBACK;

	AddressID	AddressLine1	AddressLine2	City
1	1	1970 Napa Ct.	Test Address 2	Bothell

While the 1st Statement is running, the 2nd statement can be updated immediately

Update Locks (u)

1st Query Statement

START TRANSACTION;

USE concurrency

SELECT AddressID, AddressLine1, AddressLine2, City

FROM dbo.Address WITH (UPDLOCK)

WHERE AddressID < 2

SELECT resource_type, request_mode,resource_description

FROM sys.dm tran locks

WHERE resource type <> 'DATABASE'

SELECT SLEEP(10);

ROLLBACK;

	AddressID	Add	AddressLine1		ldressLine2	City
1	1	1970 Napa Ct.		N	ULL	Bothell
	resource_type		request_mod	е	resource_de	scription
1	KEY		U		(819444328	34a0)
2	OBJECT		IX			
3	PAGE		IU		1:9312	

2nd Query Statement

START TRANSACTION;

USE concurrency

UPDATE dbo.Address

SET AddressLine2 = 'Test Address 2'

WHERE AddressID = 1

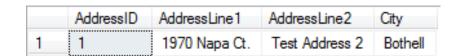
SELECT AddressID, AddressLine1,

AddressLine2, City

FROM dbo.Address

WHERE AddressID < 2

ROLLBACK;



Wait until 1st query statement done, then 2nd query statement can begin

Exclusive Locks (x)

1st Query Statement

START TRANSACTION;

USE concurrency

UPDATE dbo.Address

SET AddressLine2 = 'Test Address 2'

WHERE AddressID = 5

SELECT resource_type, request_mode, resource_description
FROM sys.dm_tran_locks
WHERE resource type <> 'DATABASE'

SELECT SLEEP(10);

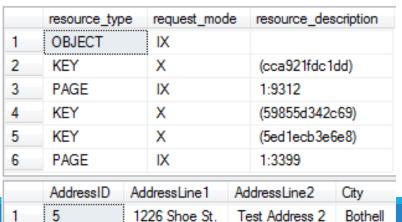
SELECT AddressID, AddressLine1, AddressLine2, City

FROM dbo.Address

WHERE AddressID = 5

ROLLBACK;

EGCl321



2nd Query Statement

SELECT *AddressID, AddressLine1, AddressLine2, City

FROM dbo.Address **WHERE** AddressID = 5

	AddressID	AddressLine1	AddressLine2	City
1	5	1226 Shoe St.	NULL	Bothell

22

Try it with "NOLOCK"
FROM dbo.Address with (NOLOCK)

Reference

1. Ramakrishnan R, Gehrke J., Database management systems, 3rd ed., New York (NY): McGraw-Hill, 2003.