

Database Systems

Concurrency

H. Turgut Uyar Şule Öğüdücü

2002-2014

1 / 45

License



© 2002-2014 T. Uyar, Ş. Öğüdücü

You are free to:

- ▶ Share – copy and redistribute the material in any medium or format
- ▶ Adapt – remix, transform, and build upon the material

Under the following terms:

- ▶ Attribution – You must give appropriate credit, provide a link to the license, and indicate if changes were made.
- ▶ NonCommercial – You may not use the material for commercial purposes.
- ▶ ShareAlike – If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

For more information:

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Read the full license:

<https://creativecommons.org/licenses/by-nc-sa/4.0/legalcode>

2 / 45

Topics

Transactions

Introduction
Recovery
Two-Phase Commit

Concurrency

Introduction
Locking
Isolation Levels
Intent Locks

3 / 45

Transactions

- ▶ a group of operations may have to be carried out together
 - ▶ finishing some operations while failing on others might cause inconsistency
- ▶ **transaction**: a logical unit of work
- ▶ no guarantee that multiple operations will all be finished
- ▶ at least return to the state before the changes

4 / 45

Transaction Example

- ▶ transfer money from one bank account to another

```
UPDATE ACCOUNTS SET BALANCE = BALANCE - 100
WHERE ACCOUNTID = 123
```

```
UPDATE ACCOUNTS SET BALANCE = BALANCE + 100
WHERE ACCOUNTID = 456
```

5 / 45

Transaction Properties

- ▶ **A**tomicity: all or nothing
- ▶ **C**onsistency: move from one consistent state to another
- ▶ **I**solation: whether operations of an unfinished transaction affect other transactions or not
- ▶ **D**urability: when a transaction is finished, its changes are permanent even if there is a system failure

6 / 45

SQL Transactions

- ▶ starting a transaction

```
BEGIN [ WORK | TRANSACTION ]
```

- ▶ finishing a transaction

```
COMMIT [ WORK | TRANSACTION ]
```

- ▶ cancelling a transaction

```
ROLLBACK [ WORK | TRANSACTION ]
```

7 / 45

Transaction Example

```
BEGIN TRANSACTION
ON ERROR GOTO UNDO
UPDATE ACCOUNTS SET BALANCE = BALANCE - 100
  WHERE (ACCOUNTID = 123)
UPDATE ACCOUNTS SET BALANCE = BALANCE + 100
  WHERE (ACCOUNTID = 456)
COMMIT
...
```

```
UNDO:
  ROLLBACK
```

8 / 45

Recovery

- ▶ consider a system failure during a transaction
- ▶ buffer cache has not been flushed to the disk
- ▶ how to guarantee durability?
- ▶ derive the data from other sources in the system

9 / 45

Transaction Log

- ▶ **log**: values of every affected tuple before and after every operation
- ▶ **write-ahead log rule**:
log must be flushed before the transaction is committed
- ▶ accessing records in the log is sequential by nature

10 / 45

Checkpoints

- ▶ create **checkpoints** at certain intervals:
- ▶ flush buffer cache to the physical medium
- ▶ note the checkpoint
- ▶ note the continuing transactions

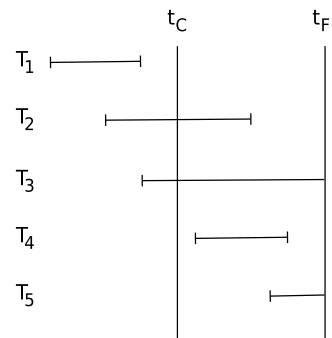
11 / 45

Recovery Lists

- ▶ after the failure: which transactions will be undone, which transactions will be made permanent?
- ▶ create two lists: *undo* (U), *redo* (R)
- ▶ t_C : last checkpoint in the log
 - ▶ add the transactions which are active at t_C to U
- ▶ scan records from t_C to end of log
 - ▶ add any starting transaction to U
 - ▶ move any finishing transaction to R

12 / 45

Recovery Example



- ▶ t_C :
 $U = [T_2, T_3]$ $R = []$
- ▶ T_4 started:
 $U = [T_2, T_3, T_4]$ $R = []$
- ▶ T_2 finished:
 $U = [T_3, T_4]$ $R = [T_2]$
- ▶ T_5 started:
 $U = [T_3, T_4, T_5]$ $R = [T_2]$
- ▶ T_4 finished:
 $U = [T_3, T_5]$ $R = [T_2, T_4]$

13 / 45

Recovery Process

- ▶ scan records backwards from end of log
- ▶ undo the changes made by the transactions in U
- ▶ scan records forwards
- ▶ redo the changes made by the transactions in R

14 / 45

Two-Phase Commit

- ▶ different source managers
- ▶ different undo / redo mechanisms
- ▶ modifications on data on different source managers
- ▶ either commit in all or rollback in all
- ▶ coordinator

15 / 45

Protocol

- ▶ coordinator → participants:
flush all data regarding the transaction
- ▶ coordinator → participants:
start transaction and report back the result
- ▶ all participants succeeded: success
- ▶ otherwise: failure
- ▶ if success, coordinator → participants: commit
- ▶ if failure, coordinator → participants: rollback

16 / 45

References

Required Reading: Date

- ▶ Chapter 15: **Recovery**

17 / 45

Concurrency

- ▶ problems that might arise due to simultaneous transactions:
- ▶ lost update
- ▶ uncommitted dependency
- ▶ inconsistent analysis

18 / 45

Lost Update

| Transaction A | Transaction B |
|---------------|---------------|
| ... | ... |
| RETRIEVE p | ... |
| ... | ... |
| ... | RETRIEVE p |
| ... | ... |
| UPDATE p | ... |
| ... | ... |
| ... | UPDATE p |
| ... | ... |

19 / 45

Uncommitted Dependency

| Transaction A | Transaction B |
|---------------|---------------|
| ... | ... |
| ... | UPDATE p |
| ... | ... |
| RETRIEVE p | ... |
| ... | ... |
| ... | ROLLBACK |
| ... | ... |

20 / 45

Inconsistent Analysis

- compute sum of accounts: acc1=40, acc2=50, acc3=30

| Transaction A | Transaction B |
|---------------------|-----------------------|
| ... | ... |
| RETRIEVE acc1 (40) | ... |
| RETRIEVE acc2 (90) | ... |
| ... | ... |
| ... | UPDATE acc3 (30 → 20) |
| ... | UPDATE acc1 (40 → 50) |
| ... | COMMIT |
| ... | ... |
| RETRIEVE acc3 (110) | ... |
| ... | ... |

21 / 45

Conflicts

- A reads, B reads: no problem
- A reads, B writes: non-repeatable read (inconsistent analysis)
- A writes, B reads: dirty read (uncommitted dependency)
- A writes, B writes: dirty write (lost update)

22 / 45

Locking

- let transactions lock the tuples they work on
- **shared** lock (S)
- **exclusive** lock (X)

23 / 45

Lock Requests

lock type compatibility matrix

| | | |
|---|---|---|
| | S | X |
| - | Y | Y |
| S | Y | N |
| X | N | N |

- existing shared lock: only shared lock requests are allowed
- existing exclusive lock: all lock requests are denied

24 / 45

Locking Protocol

- ▶ transaction requests lock depending on operation
- ▶ promote a shared lock to an exclusive lock
- ▶ if request denied, it starts waiting
- ▶ it continues when the transaction that holds the lock releases it
- ▶ **starvation**

25 / 45

Lost Update

| Transaction A | Transaction B |
|-----------------|-----------------|
| ... | ... |
| RETRIEVE p (S+) | ... |
| ... | ... |
| ... | RETRIEVE p (S+) |
| ... | ... |
| UPDATE p (X-) | ... |
| wait | ... |
| wait | UPDATE p (X-) |
| wait | wait |

26 / 45

Uncommitted Dependency

| Transaction A | Transaction B |
|-----------------|---------------|
| ... | ... |
| ... | UPDATE p (X+) |
| ... | ... |
| RETRIEVE p (S-) | ... |
| wait | ... |
| wait | ROLLBACK |
| RETRIEVE p (S+) | |
| ... | |

27 / 45

Inconsistent Analysis

| Transaction A | Transaction B |
|--------------------|------------------|
| ... | ... |
| RETRIEVE acc1 (S+) | ... |
| RETRIEVE acc2 (S+) | ... |
| ... | ... |
| ... | UPDATE acc3 (X+) |
| ... | UPDATE acc1 (X-) |
| ... | wait |
| RETRIEVE acc3 (S-) | wait |
| wait | wait |

28 / 45

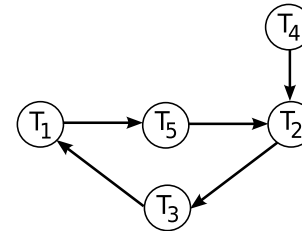
Deadlock

- ▶ **deadlock**: transactions waiting for each other to release the locks
- ▶ almost always between two transactions
- ▶ detect and solve
- ▶ prevent

29 / 45

Solving Deadlocks

Example



- ▶ wait graph
- ▶ choose a **victim** and kill it

30 / 45

Preventing Deadlocks

- ▶ every transaction has a starting timestamp
- ▶ if A's request conflicts with B's lock:
- ▶ **wait-die**: A waits if older than B, otherwise it dies
A is rolled back and restarted
- ▶ **wound-wait**: A waits if younger than B, otherwise it wounds B
B is rolled back and restarted
- ▶ timestamp of restarted transaction is not changed

31 / 45

Lock Statements

- ▶ shared lock
`SELECT query FOR SHARE`
- ▶ exclusive lock
`SELECT query FOR UPDATE`

32 / 45

Isolation Levels

- ▶ if isolation is decreased, concurrency can be increased
- ▶ various isolation levels
- ▶ serializable
- ▶ repeatable read
- ▶ read committed
- ▶ read uncommitted

33 / 45

Serializability

- ▶ *serial execution*: a transaction starts only after another is finished
- ▶ **serializable**: result of concurrent execution is always the same as one of the serial executions

Example

- ▶ $x = 10$
- ▶ transaction A: $x = x + 1$
- ▶ transaction B: $x = 2 * x$
- ▶ first A, then B: $x = 22$
- ▶ first B, then A: $x = 21$

34 / 45

Two-Phase Locking Protocol

- ▶ **two-phase locking**:
after any lock is released, no more new lock requests
- ▶ expansion phase: gather locks
- ▶ contraction phase: release locks
- ▶ **two-phase strict locking**:
all locks are released at the end of the transaction
- ▶ *If all transactions obey the two-phase locking protocol, all concurrent executions are serializable.*

35 / 45

Read Committed

- ▶ read committed: only X locks are held until end of transaction

| Transaction A | Transaction B |
|-----------------|---------------|
| ... | ... |
| RETRIEVE p (S+) | ... |
| ... | ... |
| release lock | ... |
| ... | ... |
| ... | UPDATE p (X+) |
| ... | COMMIT |
| RETRIEVE p (S+) | |

36 / 45

Phantoms

- ▶ **phantom**: when the query is executed again, new tuples appear

Example

- ▶ A computes the average of a customer's account balances:

$$\frac{100+100+100}{3} = 100$$

- ▶ B creates new account (200) for the same customer

- ▶ A computes again:

$$\frac{100+100+100+200}{4} = 125$$

37 / 45

Isolation Levels

- ▶ setting an isolation level

```
SET TRANSACTION ISOLATION LEVEL
[ SERIALIZABLE | REPEATABLE READ |
  READ COMMITTED | READ UNCOMMITTED ]
```

38 / 45

Isolation Level Problems

| | dirty read | non-repeatable read | phantom |
|------------------|------------|---------------------|---------|
| READ UNCOMMITTED | Y | Y | Y |
| READ COMMITTED | N | Y | Y |
| REPEATABLE READ | N | N | Y |
| SERIALIZABLE | N | N | N |

39 / 45

Locking Granularity

- ▶ lock relations instead of tuples
- ▶ even the entire database
- ▶ granularity: unit of locking
- ▶ if granularity is increased, concurrency is decreased
- ▶ hard to find locks on tuples
→ first, get **intent locks** on relation variables

40 / 45

Intent Locks

- ▶ Intent Shared (IS):
transaction intends to read some tuples
- ▶ Intent Exclusive (IX):
IS + transaction intends to write some tuples
- ▶ Shared (S):
concurrent readers are allowed but no concurrent writers
- ▶ Shared + Intent Exclusive (SIX)
- ▶ Exclusive (X):
no concurrency allowed on this relation

41 / 45

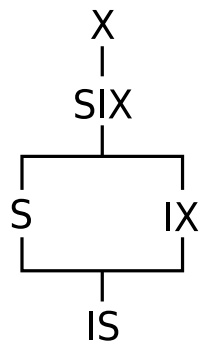
Lock Requests

lock compatibility matrix

| | IS | IX | S | SIX | X |
|-----|----|----|---|-----|---|
| - | Y | Y | Y | Y | Y |
| IS | Y | Y | Y | Y | N |
| IX | Y | Y | N | N | N |
| S | Y | N | Y | N | N |
| SIX | Y | N | N | N | N |
| X | N | N | N | N | N |

42 / 45

Lock Precedence



- ▶ for an S on a tuple,
at least an IS on the relation
- ▶ for an X on a tuple,
at least an IX on the relation

43 / 45

Locking Statements

- ▶ lock a table


```
LOCK [ TABLE ] table_name
      [ IN lock_mode MODE ]
```
- ▶ lock modes:
 - ▶ ACCESS SHARE
 - ▶ ROW SHARE
 - ▶ ROW EXCLUSIVE
 - ▶ SHARE UPDATE EXCLUSIVE
 - ▶ SHARE
 - ▶ SHARE ROW EXCLUSIVE
 - ▶ EXCLUSIVE
 - ▶ ACCESS EXCLUSIVE

44 / 45

References

Required Reading: Date

- ▶ Chapter 16: **Concurrency**