ICS 321 Fall 2009

Overview of Transaction Management (ii)

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

Transactions in SQL

- After connection to a database, a transaction is automatically started
  - Different connections -> different transactions
- Within a connection, a transaction is ended by
  - **COMMIT** or **COMMIT WORK**
  - **ROLLBACK** (= “abort”)
- **SAVEPOINT** <savepoint name>
- **ROLLBACK TO SAVEPOINT** <savepoint name>
  - Locks obtained after savepoint can be released after rollback to that savepoint
- Using savepoints vs sequence of transactions
  - Transaction rollback is to last transaction only

Lock Granularity

- What should the DBMS lock ?
  - Row ?
  - Page ?
  - A Table ?

```
UPDATE  Sailors
SET     rating=0
WHERE   rating>9
```

```
SELECT  *
FROM    Sailors
```

```
SELECT  *
FROM    Sailors
WHERE rating < 2
```

```
UPDATE  Boats
SET     color='red'
WHERE   bid=13
```

```
UPDATE  Boats
SET     color='blue'
WHERE   bid=100
```

Isolation levels in SQL

- SQL supports 4 isolation levels

SQL Isolation Levels	DB2 Isolation Levels	Dirty read	Unrepeatable Read	Phantom
READ UNCOMMITTED	UNCOMMITTED READ (**UR**)	Maybe	Maybe	Maybe
READ COMMITTED	CURSOR STABILITY * (**CS**)	No	Maybe	Maybe
REPEATABLE READ	READ STABILITY (**RS**)	No	No	Maybe
SERIALIZABLE	REPEATABLE READ (**RR**)	No	No	No

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

```
SELECT *
FROM Reserves
WHERE SID=100
WITH UR
```

Crash Recovery

- **Transaction Manager**: DBMS component that controls execution (eg. managing locks).
- **Recovery Manager**: DBMS component for ensuring
  - Atomicity: undo actions of transactions that do not commit
  - Durability: committed transactions survive system crashed and media failures
- Assume atomic writes to disk.

The Log

- The following actions are recorded in the log:
  - *Ti writes an object*: the old value and the new value.
    - Log record must go to disk *before* the changed page! (Write Ahead Log property)
  - *Ti commits/aborts*: a log record indicating this action.
- Log records are chained together by Xact id, so it's easy to undo a specific Xact.
- Log is often *duplexed* and *archived* on stable storage.
- All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Stealing Frames & Forcing Pages

- **Stealing Frames**: writing a modified page to disk before transaction commits.
  - T1 updates row r
  - T2 needs to fetch a page, but bufferpool is full
  - The page containing r is chosen for eviction
  - Write page containing r back to disk (optimistic)
  - What happens if T1 aborts ?
- **Forcing Pages**: All modified pages written back to disk when transaction commits.
  - If no-force is used, what happens after a crash ?

Recovering from a Crash

- There are 3 phases in the *Aries* recovery algorithm:
  - *Analysis*: Scan the log forward (from the most recent *checkpoint*) to identify all Xacts that were active, and all dirty pages in the buffer pool at the time of the crash.
  - *Redo*: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk.
  - *Undo*: The writes of all Xacts that were active at the crash are undone (by restoring the *before value* of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process!)