Ch 10: Transaction Management and Concurrent Control

Learning Objectives

- A transaction *represents a real-world event* such as the sale of a product.
- A transaction must be a *logical unit of work*. That is, no portion of a transaction **stands by itself**. For example, the product sale has an effect on inventory and, if it is a credit sale, it has an effect on customer balances.
- A transaction must *take a database from one consistent state to another*. Therefore, all parts of a transaction must be executed or the transaction must be aborted. (A consistent state of the database is one in which all data integrity constraints are satisfied.)

Each SQL transaction is composed of several *database requests*, each one of which yields I/O operations.

A *transaction log* keeps track of all transactions that modify the database.

The transaction log data may be used for recovery (ROLLBACK) purposes.

Transaction

A sequence of logical steps that will accomplish a single task (or what seems like a single task)

ex:

- add an employee
- enter an order
- enroll a student in a course

- A single task may require MANY changes to the database.
- If all changes are NOT made database integrity will be lost

Enroll a student

- MARY to INSS651

steps:

start

- 1.check to see if student "MARY" exists… read only
- 2.check to see if class "INSS651" exists.. read only
- 3. access enrollment table .. read only
- update enrollment table ..make changes (update)
- 4. access student record/table ..read only
- update student table ..make changes (update)
- 5. access class record/table
- update class table
- 6.commit transaction

end

Transaction Prop. (p 401)

ATOMICITY:

- all phases (steps) must be completed, if not abort the trans.

DURABILITY:

- permanence of DB consistent state achieved only when transaction is complete

SERIALIZABILITY:

- Be able to serialize concurrent trans.

ISOLATION:

- Be able to isolate data and can not be used by other trans.

All transactions have FOUR properties

Single-user databases

Atomicity: Unless all parts of the transactions are executed, the transaction is aborted

Durability: Once a transaction is committed, it cannot be rolled back

Multi-user databases

Serializability: The result of the concurrent execution of transactions is the same as though the transactions were executed in serial order.

Isolation: data used by one transaction can not be used by another transaction until the first transaction is complete

Begin transaction:

- step 1
- step 2
- .
- .
- .

END Transaction

- (COMMIT)
- if aborted for any reasons, ROLLBACK,
- i.e., change back to previous commit

SQL provides transaction support through

- **COMMIT** (permanently saves changes to disk)

and

- **ROLLBACK** (restores the previous database state)

Transaction Log(Journal): (p 402)

Keeps track of all transactions that update the DB

Info kept in a typical log:

- Trans. ID
- time of trans.
- type of trans.
- object of action
- BEFORE image
- AFTER image

this allows for FORWARD & BACKWARD recovery

Ex:add student xyz to inss651

transaction ID	transaction description
ST1	get student table (check to see if student xyz exists
	get class table (check to see if inss651 exists)
	get enrollment table
	enroll student XYZ in inss651
	commit
	update student record (ie total number of hours)
	commit
	update class record (i.e., total number of students)
	commit

Transaction log

TRANS ID	TIME	ACTION	OBJECT OF ACTION	BEFORE IMAGE	AFTER IMAGE
ST1	8:00	START			
ST1	8:04	INSERT	ENROLLMENT(XYZ,INSS651..)		NEW
ST1	8:16	MODIFY	STUDENT (XYZ,..)	OLD	NEW
ST1	8:20	MODIFY	CLASS(INSS651..)	OLD	NEW
ST1	8:30	COMMIT			

Recovery Management

Restores the database to a consistent state

Two types:

Forward Recovery

Backward recovery

Forward Recovery

FORWARD recovery:

- if ALL or PART of the database has been destroyed then start with most recent backup and update it using AFTER images from COMMITTED transactions to this copy.

BACKWARD recovery:

- if DB is not actually destroyed but trans. was not completed, then we need to bring DB back to consistent state. Start with current DB and UNDO changes using BEFORE values of uncommitted trans.

Checkpoint

which is most recent correct state?

with large users/transaction it is not clear what is the **correct** database state.

Periodically system will refuse any new requests and will complete transactions in progress.

Usually done every 15 min. or so to synchronize log and DB

Recovery THRU OFFsetting trans.

- GIGO but a commit is done
- Rollback is not effective
- Create a dummy offsetting transaction

EX:

- IF WE WANT TO REDUCE BALACE BY $30 BUT BY MISTAKE WE PUT $50, an offsetting transaction would require a “dummy” transaction of ??? to ADD to the account

DIFFERENTIAL files:

- DB is not updated directly, but a DIFFERENTIAL file containing the changes is created
- DB is periodically updated in batch mode. Similar to accounting systems

CONCURRENCY CONTROL:

management of *concurrent* transaction execution. (Therefore, a single-user database does not require concurrency control!)

Lost updates

- Updates are lost in concurrency updates

Ex: ex: TOM needs to increase product A by $35 and MARY needs to

decrease it by 35

Uncommitted data (page 405)

When one transaction is rolled back but the other transaction gets uncommitted data

Inconsistent data (see 405)

When a calculation is done over a set of data while other transactions are still updating data

Scheduler

multi-user DBMSs use a process known as a *scheduler* to enforce concurrency control

It serializes the transactions to ensure "isolation".

Lock

Provides “isolation” when needed

Exclusive

Shared

TWO-PHASE locking (see fig 10.7)

TOM has finished five steps of a six step transaction, but data isneeded for sixth step is locked...

Growing Phase:

- a trans. acquires all the locks needed for that transaction

Shrinking Phase:

- transaction releases ALL locks and can not get any new locks

Deadlock (page 414)

- When two transactions wait for each other to unlock data

Control:

DBMS detect it and abort transaction (rolled back) and the other transaction continues

Get “all” the locks needed before starting a transaction

Time Stamp

Each transaction is assigned a unique time stamp

Provides order in which transactiosn are submitted to DBMS for processing

All operations within the same transaction must have the same time stamp