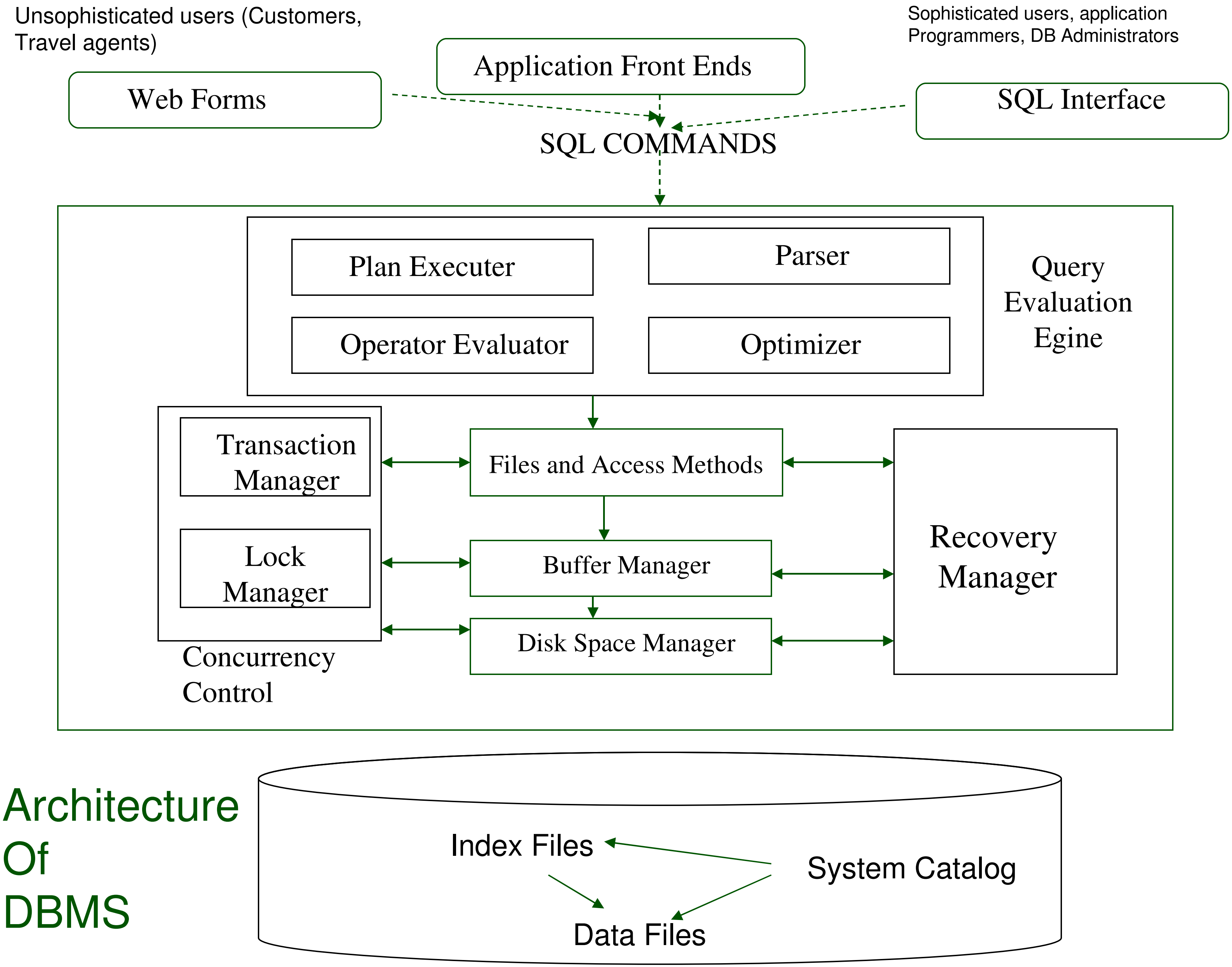
Transaction Management Overview

Chapter 16



Architecture Of DBMS

Outline

- The ACID Properties
- Transactions and Schedules
- Concurrent Execution of Transactions
- Lock-based Concurrency control
- Performance of Locking
- Transaction Support in SQL
- Introduction to Crash Recovery

Concurrency

- Concurrent execution of user programs is essential for good DBMS performance.

- Because disk accesses are frequent, and relatively slow, it is important to keep the CPU humming by working on multiple user programs concurrently.

Transactions

- A user's program may carry out many operations on the data from the database,

- but the DBMS is only concerned about what data is read/written from/to the database.

- A transaction is the DBMS's abstract view of a user program: a sequence of reads and writes

Example of User Program and Transaction

- Assume A is a tuple in relation Account
- User program

```
Calculate_Interests (Account A, Rate r) {
   float x,y;
    x := A;   // read from database
    if (x <= 1000) y := 1.05*x
    else if (x <= 10000) y := 1.06*x
    else y := 1.065*x;
    A := y;   // write to database
}
```

- Corresponding Transaction T

  T: x := A, A := y   or   T: R(A), W(A)

Transactions

- A transaction is a sequence of operations.
- The operations can be reading R(A), writing W(A), commit or abort

- Required Properties of transactions:
  - Atomicity
  - Consistency
  - Isolation
  - Durability

- Called as “ACID properties”

Atomicity of Transactions

- A transaction might commit after all its actions, or it could abort (or be aborted) after executing some actions.

- when it commits, all its writes are written
  when it aborts, none of its writes is written

This is called the **atomicity** property

Atomicity

- Why atomicity?
- Partial effects leave wrong data in DB

Transfer (Acount A, Acount B, Money r)

{ A := A – r; B := B + r } //Transfer $r from A to B

- What if **Transfer** is aborted after A := A - r and before B := B + r?

- DBMS logs all actions so that it can undo the actions of aborted transactions.
- Transaction Manager and RECOVERY

Consistency

- Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins (called **consistency** property).

- DBMS assumes that the consistency holds for each transaction

- Ensuring consistency property is the responsibility of the user.

Isolation

- Users submit transactions, and can think of each transaction as executing by itself (called **isolation** property).
  - concurrently executing transactions will have the same effect as “one at a time”
  - Transactions are isolated or protected.
  - Isolation is achieved by the DBMS

Isolation

- Consider two transactions :

```
T1:     BEGIN   A=A+100,   B=B-100   END
T2:     BEGIN   A=1.06*A,   B=1.06*B   END
```

- T1 transfers $100 from B’s account to A’s account.
- T2 credits both accounts with a 6% interest payment.

- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together.
- However, the net effect *must* be equivalent to these two transactions running serially in some order.

Serial schedule: Schedule that does not interleave the actions of different transactions.

Either serial schedule is ok

A=1000
B=1000

Time line

1100*1.06
=1166

900*1.06
=954

T1: A=A+100, B=B-100
T2: A=1.06*A, B=1.06*B

T1: A=A+100, B=B-100
T2: A=1.06* 1.06*B

1160

960

1060

A non-serial execution (interleaving schedule):

T1: A=A+100, B=B-100
T2: A=1.06*A, B=1.06*B

Durability

- Once a transaction commits, its effect persists (durability property.)

- ACID: Atomicity, Consistency,
  Isolation, Durability

Outline

- The ACID Properties
- **Transactions and Schedules**
- Concurrent Execution of Transactions
- Lock-based Concurrency control
- Performance of Locking
- Transaction Support in SQL
- Introduction to Crash Recovery

A Schedule

- DBMS sees transaction as a list of actions: reads, writes
- $R_T(O)$: Reading object O.
- $W_T(O)$: writing object O.
- In addition, commit and abort actions are included.
  - $Commit_T$ denotes committing action
  - $Abort_T$ denotes aborting action.

- T = {T1, ... , Tn} is a set of transactions.
- A schedule (history) is a sequence of operations in T1, ...,Tn such that for each

  1. Each operation in Ti appears exactly once, and

Transaction Interleaving

- Consider a possible interleaving schedule:

A=1000
B=1000

Time line

900*1.06
=954

T1:	A=A+100,		B=B-100	
T2:		A=1.06*A,		B=1.06*B

- This is OK. It is equivalent to the following serial execution

T1:	A=A+100,	B=B-100		
T2:			A=1.06*A,	B=1.06*B

1100*1.06
=1166

900*1.06
=954

- But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

1100 1166 960

1060

- But what about:

T1: A=A+100, B=B-100

T2: A=1.06*A, B=1.06*B

- It is not equivalent to any serial execution

900*1.06 =954

1166

T1: A=A+100, B=B-100

T2: A=1.06*A, B=1.06*B

1060 1060

T1: A=A+100, B=B-100

T2: A=1.06*A, B=1.06*B

1160 960

- The DBMS’s view of the above schedule:

T1: R(A), W(A), R(B), W(B)

T2: R(A), W(A), R(B), W(B)

Scheduling Transactions

A:
T1: A=A+100, B=B-100
T2: A=1.06*A, B=1.06*B

- Equivalent schedules: the effects (on the database) of two schedules are identical

B:
T1: A=A+100, B=B-100
T2: A=1.06*A, B=1.06*B

C:
T1: A=A+100, B=B-100
T2: A=1.06*A, B=1.06*B

- A and B are equivalent, but A and C are not.

- Translate this example to DBMS view, we have:

T1: A=A+100, B=B-100
T2: A=1.06*A, B=1.06*B

A:
T1: R(A), W(A), R(B), W(B)
T2: R(A), W(A), R(B), W(B)

B:
T1: R(A), W(A), R(B), W(B)
T2: R(A), W(A), R(B), W(B)

C:
T1: R(A), W(A), R(B), W(B)
T2: R(A), W(A), R(B), W(B)

- A and B are equivalent, but A and C are not.
- Problem: WR conflict

Serializable Schedule

- A schedule that is equivalent to some serial execution of the transactions is called a serializable schedule.

- every serializable schedule preserves consistency

  - In the serial schedule, transactions are executed one after another
  - Every transaction itself preserves consistency (by consistency property)
  - So the serial schedule preserves consistency
  - So the serializable schedule preserves consistency

Anomalies with Interleaved Execution

Should not have allowed T2 to withdraw $500 !!!

- Reading Uncommitted Data
- (WR Conflicts, “dirty reads”):

Commit T2 !

T1:	R(A), W(A),		Abort
T2:		R(A), W(A), C	

Initially A=0

T1 : Deposit $1000

T2 : Withdraw $500

T1 decides not to deposit the $1000 after all

Anomalies with Interleaved Execution

- Unrepeatable Reads
- (RW Conflicts):

If T2 reads A again here, it will find that A=$200

T1:	R(A),		W(A)
T2:		R(A), W(A), C	

Initially A=$1000

T2: withdraw $800

Commit T2 !

T1 withdraws $500, thinking that there is still $1000

- Imagine T1 verifies first that fund is sufficient, then withdraw.

Anomalies

- Overwriting Uncommitted Data (WW Conflicts):

A=1 B=1

```
T1:    W(A),                          W(B), C
T2:              W(A), W(B), C
```

A=0 B=0

  - Both A and B must be initialized to 1 or 0.
  - T1 writes 1 to both, and T2 writes 0 to both.
  - In the end, A = 0, B = 1

    (T1 writes 1 to A, but overwritten by T2)

    Now A and B have got different values

‘

None of these anomalies is a serializable schedule!!

Recoverable Schedules

- In a recoverable schedule, transactions commit only after all transactions whose changes they read commit.
  - Cascading abort problem
- The schedule becomes recoverable, if a transactions read the effect of committed transactions.

- The following schedule is not recoverable

```
T1:   R(A) W(A),                              A
T2:             R(A) W(A) R(B) W(B)  C
```

Concurrency Control

- DBMS must
  - Ensure serializable schedules
  - Recoverable schedules
  - Improve parallelism for high throughput and low response time.

- A DBMS uses a locking-based protocol.

- Strict Two-phase Locking (Strict 2PL) Protocol:
  - transaction must obtain a S (shared) lock on object before reading, and an X (exclusive) lock on object before writing.
  - If a transaction holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
  - All locks held by a transaction are released when the transaction completes

Strict 2PL allows only serializable schedules.

S-X lock conflict and X-X lock conflict

A is X-locked by T1 !

- T1: R(A), W(A), Abort
- T2: R(A), W(A), C

A is S-locked by T1 !

T1: R(A), R(A), W(A), Abort
T2: R(A), W(A), C

A is X-locked by T1 !

T1: W(A), W(B), C
T2: W(A), W(B), C

Aborting a Transaction

- If a transaction Ti is aborted, all its actions have to be undone.

  if Tj reads an object last written by Ti, Tj must be aborted as well!

- Most systems avoid such cascading aborts by releasing a transaction's locks only at commit time.
  - If Ti writes an object, Tj can read this only after Ti commits.

Aborting a Transaction

- In order to undo the actions of an aborted transaction, the DBMS maintains a log in which every write is recorded.

- This mechanism is also used to recover from system crashes:

  all active transactions at the time of the crash are aborted when the system comes back up.

Deadlocks

- A cycle of transactions waiting for a lock
- Timeout is a simple mechanism
- Details will be discussed later.

Performance of Locking

- Lock-based schemes resolve conflicts using two basic mechanisms: blocking and aborting.
  - Both involve performance penalty
- Blocking leads to cascading waiting
  - CPU is free but can not be used!
  - Deadlocks- extreme blocking!
    - About 1 % of transactions are involved in deadlock.
  - Transaction blocking is significant overhead
- Thrashing occurs as MPL increases
  - Occurs if about 30% of transactions are blocked.
- Throughput can be increased
  - By locking the small sized objects possible.
  - By reducing the time that transaction holds
  - By reducing the **hotspots**
    - A hotspot is a data object that is frequently accessed and modified.

Transaction Support in SQL

- So far we have studied abstract model
- A transaction automatically starts when user executes a statement that accesses the database or catelogs: SELECT, UPDATE, CREATE TABLE
- The statements are executed until COMMIT and ROLLBACK (for abort) commands is encountered.
- In SQL:1999 the following features are provided
  - Savepoint: allows us to identify a point and selectively rollback operations carried out after this point.
    - SAVEPOINT (Name)
    - ROLLBACK TO SAVEPOINT (Savepoint name)

What should we lock ?

- Transaction locks objects
- Consider the following query
  - SELECT S.rating, MIN (S.age)
  - FROM Sailors S
  - WHERE S.rating=8
- Option 1:
  - DBMS should put a shared lock on entire Table. Another Transaction which modifies Sailor table should have exclusive lock on the table. LEADS to low concurrency.
  - We can lock the smaller objects
    - Set an exclusive lock on single row.
    - Reading can be carried out in parallel
- So, DBMS can lock the objects at different granularities!
  - Tables-level and row-level

Phantom Problem

- Suppose DBMS sets row-level locks with rating =8 for T1.
- T3 can add another row and lock it.
- Then T1 gets two results based on execution of T1 relative to T2.
- Phantom problem: A transaction retrieves collection of objects twice and sees different results, even though it does not modify any results.
- Solution: lock entire table.

Transaction characteristics in SQL

- SQL allows the programmers to specify three characteristics of transaction.
  - Access mode
  - Diagnostics size
  - Isolation level
- Diagnostics size:
  - determines the number of error conditions that can be recorded.
- Access mode:
  - if access mode is READ ONLY, the transaction is not allowed to modify the database.
  - If the mode is READ-WRITE
    - INSERT, DELETE, UPDATE and CREATE commands can be executed.
- Isolation Level:
  - controls the extent to which a given transaction is exposed to the actions of other transactions executing concurrently.
  - By choosing one of four possible isolation level settings, the user can obtain greater concurrency.
- Isolation levels: READ UNCOMMITTED, READ COMITTED, REPEATABLE READ, SERIALIZABLE

Transaction characteristics in SQL

- SERIALIZABLE: Strict 2PL
  - Ensures that T reads only changes made by committed transactions
  - No value **read or written** by T is changed by other transactions
  - Avoids a phantom phenomenon.
- REPEATABLE READ: Does not do index locking.
  - First and second are the same
  - But, T could experience a phantom phenomenon.
- READ COMMITTED: Read locks are released immediately, exclusive locks are held till the end
  - First is same
  - No value **written** by T is modified by other transactions
  - However, the value read by T may be modified by other transactions while T is in progress.
  - T could experience a phantom phenomenon.
- READ UNCOMMITTED: does not obtain shared locks and does not request exclusive locks.

Level	Dirty Read	Unrepeatable read	Phantom
READ UNCOMMITTED	Maybe	Maybe	Maybe
READ COMMITTED	No	Maybe	Maybe
REPEATABLE READ	No	No	Maybe
SERIALIZABLE	No	No	No

Transaction Characteristics in SQL

- SERIALIZABLE ISOLATION level is the safest and recommended for most transactions.
- For some transactions we can run at lower levels to improve the performance.
- Example: A statistical query which finds an average sailors age can be run at READ COMMITTED level or even READ UNCOMMITTED level
- The ISOLATION LEVEL and ACCESS MODE can be specified using the SET TRANSACTION command.
- Example: The following command declares the current transaction to be SERIALIZABLE and READ ONLY

  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ ONLY
- When transaction is started the default is SERIALIZABLE and READ WRITE

Introduction to Crash Recovery

- Recovery Manager is responsible for atomicity and durability.
- It ensures atomicity by undoing the actions of transactions
- It ensure durability by making sure that all actions of committed transactions survive system crashes and media failures.
- After the restart, the recovery manager is given the control to bring the database to a consistent state.
- Write ahead logging protocol is employed

Summary

- Concurrency control and recovery are among the most important functions provided by a DBMS.
- Users need not worry about concurrency.
  - System automatically inserts lock/unlock requests and schedules actions of different transactions to ensure that the resulting execution is serializable
  - logging is used to undo the actions of aborted transactions and to restore the system to a consistent state after a crash.