Transactions

What is Recovery

- Serializability helps to ensure ***isolation*** and ***consistency*** of a schedule
- Yet the ***atomicity*** and ***consistency*** may be compromised in the face of system failures
- Consider a schedule comprising a single transaction (obviously serial):
  - 1. **read**(*A*)
  - 2. *A* := *A* – 50
  - 3. **write**(*A*)
  - 4. **read**(*B*)
  - 5. *B* := *B* + 50
  - 6. **write**(*B*)
  - *7.* **commit** *// make the changes permanent; show the results to the user*
- What if system fails after Step 3 and before Step 6?
  - Leads to inconsistent state
  - Need to rollback update of A

Recoverable Schedules

- Need to address the effect of transaction failures on concurrently running transactions

- **Recoverable schedule**
  - If a transaction T_j reads a data item previously written by a transaction T_i, then the commit operation of T_i appears before the commit operation of T_j

- The following schedule is not recoverable if T_9 commits immediately after the read(A) operation

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

- If T_8 should abort, T_9 would have read (and possibly shown to the user) an inconsistent database state

- Hence, database must ensure that schedules are recoverable

Cascading Rollbacks

- **Cascading rollback:** A single transaction failure leads to a series of transaction rollbacks

- Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable)

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

- If T_{10} fails, T_{11} and T_{12} must also be rolled back

- Can lead to the undoing of a significant amount of work

Cascadeless Schedules

- **Cascadeless schedules:** Cascading rollbacks cannot occur
- For each pair of transactions T_i and T_j such that T_j reads a data item previously written by T_i, the commit operation of T_i appears before the read operation of T_j
- Every Cascadeless schedule is also recoverable
- It is desirable to restrict the schedules to those that are cascadeless
- Example of a schedule that is NOT cascadeless

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

Recoverable Schedules: Example

- **Irrecoverable Schedule:**

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
		Commit;		
Failure Point				
Commit;				

Recoverable Schedules: Example

- **Recoverable with Cascading Rollback:**

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
Failure Point				
Commit;				
		Commit;		

Recoverable Schedules: Example

- **Recoverable without Cascading Rollback:**

T1	T1's buffer space	T2	T2's Buffer Space	Database
				A=5000
R(A);	A=5000			A=5000
A=A-100;	A=4000			A=5000
W(A);	A=4000			A=4000
Commit;				
		R(A);	A=4000	A=4000
		A=A+500;	A=4500	A=4000
		W(A);	A=4500	A=4500
		Commit;		

Transaction Definition in SQL

- Data manipulation language must include a construct for specifying the set of actions that comprise a transaction
- In SQL, a transaction begins implicitly
- A transaction in SQL ends by:
  - **Commit work** commits current transaction and begins a new one
  - **Rollback work** causes current transaction to abort
- In almost all database systems, by default, every SQL statement also commits implicitly if it executes successfully
- Implicit commit can be turned off by a database directive
  - E.g., in JDBC -- connection.setAutoCommit(false);
- Isolation level can be set at database level
- Isolation level can be changed at start of transaction
  - E.g. In SQL **set transaction isolation level serializable**
  - E.g. in JDBC - connection.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE)

Transaction Control Language (TCL)

The following commands are used to control transactions

- **COMMIT:** To save the changes
- **ROLLBACK:** To roll back the changes
- **SAVEPOINT:** Creates points within the groups of transactions in which to ROLLBACK
- **SET TRANSACTION:** Places a name on a transaction

Transactional Control Commands

- Transactional control commands are only used with the **DML Commands** such as
  – INSERT, UPDATE and DELETE only
- They cannot be used while creating tables or dropping them because these operations are automatically committed in the database

TCL: COMMIT Command

- The COMMIT command is the transactional command used to save changes invoked by a transaction to the database
- The COMMIT command saves all the transactions to the database since the last COMMIT or ROLLBACK command
- The syntax for the COMMIT command is as follows:
  - `SQL> DELETE FROM CUSTOMERS WHERE AGE = 25;`
  - `SQL> COMMIT;`

Before DELETE

```
SQL> SELECT * FROM CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

After DELETE

```
SQL> SELECT * FROM CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
3	kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

TCL: ROLLBACK Command

- The ROLLBACK command is the transactional command used to undo transactions that have not already been saved to the database
- This command can only be used to undo transactions since the last COMMIT or ROLLBACK command was issued
- The syntax for a ROLLBACK command is as follows:
  - `SQL> DELETE FROM CUSTOMERS WHERE AGE = 25;`
  - `SQL> ROLLBACK;`

Before DELETE

```
SQL> SELECT * FROM CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

After DELETE

```
SQL> SELECT * FROM CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

TCL: SAVEPOINT / ROLLBACK Command

- A SAVEPOINT is a point in a transaction when you can roll the transaction back to a certain point without rolling back the entire transaction

- The syntax for a SAVEPOINT command is:

  `SAVEPOINT SAVEPOINT_NAME;`

- This command serves only in the creation of a SAVEPOINT among all the transactional statements

- The ROLLBACK command is used to undo a group of transactions
- The syntax for rolling back to a SAVEPOINT is:

  `ROLLBACK TO SAVEPOINT_NAME;`

- `SQL> SAVEPOINT SP1;`
  - Savepoint created
- `SQL> DELETE FROM CUSTOMERS WHERE ID=1;`
  - 1 row deleted
- `SQL> SAVEPOINT SP2;`
  - Savepoint created
- `SQL> DELETE FROM CUSTOMERS WHERE ID=2;`
  - 1 row deleted
- `SQL> SAVEPOINT SP3;`
  - Savepoint created
- `SQL> DELETE FROM CUSTOMERS WHERE ID=3;`
  - 1 row deleted

TCL: SAVEPOINT / ROLLBACK Command

- Three records deleted
- Undo the deletion of first two
- `SQL> ROLLBACK TO SP2;`
  - Rollback complete

- `SQL> SAVEPOINT SP1;`
- `SQL> DELETE FROM CUSTOMERS WHERE ID=1;`
- `SQL> SAVEPOINT SP2;`
- `SQL> DELETE FROM CUSTOMERS WHERE ID=2;`
- `SQL> SAVEPOINT SP3;`
- `SQL> DELETE FROM CUSTOMERS WHERE ID=3;`

At the begining

```
SQL> SELECT * FROM CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

After ROLLBACK

```
SQL> SELECT * FROM CUSTOMERS;
```

ID	NAME	AGE	ADDRESS	SALARY
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

TCL: RELEASE SAVEPOINT Command

- The RELEASE SAVEPOINT command is used to remove a SAVEPOINT that you have created

- The syntax for a RELEASE SAVEPOINT command is as follows:

```
RELEASE SAVEPOINT SAVEPOINT_NAME;
```

- Once a SAVEPOINT has been released, you can no longer use the ROLLBACK command to undo transactions performed since the last SAVEPOINT

TCL: SET Transaction Command

- The SET TRANSACTION command can be used to initiate a database transaction

- This command is used to specify characteristics for the transaction that follows:
  - For example, you can specify a transaction to be read only or read write.

- The syntax for a SET TRANSACTION command is as follows:

```
SET TRANSACTION [ READ WRITE | READ ONLY ];
```

View Serializability

- Let *S* and *S'* be two schedules with the same set of transactions
- *S* and *S'* are **view equivalent** if the following three conditions are met, for each data item *Q*,
  - 1. If in schedule S, transaction T_i reads the initial value of *Q*, then in schedule *S'* also transaction T_i must read the initial value of *Q*
  - 2. If in schedule S transaction T_i executes **read**(*Q*), and that value was produced by transaction T_j (if any), then in schedule *S'* also transaction T_i must read the value of *Q* that was produced by the same **write**(Q) operation of transaction T_j
  - 3. The transaction (if any) that performs the final **write**(*Q*) operation in schedule *S* must also perform the final **write**(*Q*) operation in schedule *S'*
- As can be seen, view equivalence is also based purely on **reads** and **writes** alone

View Serializability

- A schedule *S* is **view serializable** if it is view equivalent to a serial schedule
- Every conflict serializable schedule is also view serializable
- Below is a schedule which is view-serializable but *not* conflict serializable

T_{27}	T_{28}	T_{29}
read (Q)		
	write (Q)	
write (Q)		
		write (Q)

- What serial schedule is above equivalent to?
  - $T_{27} – T_{28} – T_{29}$
  - The one read(Q) instruction reads the initial value of Q in both schedules and
  - T_{29} performs the final write of Q in both schedules

- T_{28} and T_{29} perform write(Q) operations called **blind writes**, without having performed a read(Q) operation

Test for View Serializability

- The precedence graph test for conflict serializability cannot be used directly to test for view serializability
  - Extension to test for view serializability has cost exponential in the size of the precedence graph
- The problem of checking if a schedule is view serializable falls in the class of *NP*-complete problems
  - Thus, existence of an efficient algorithm is *extremely* unlikely
- However practical algorithms that just check some **sufficient conditions** for view serializability can still be used

View Serializability: Example 1

- Check whether the schedule is *view serializable* or not?
  - S: $r_2(B)$, $r_2(A)$, $r_1(A)$, $r_3(A)$, $w_1(B)$, $w_2(B)$, $w_3(B)$

- Solution:
  - With 3 transactions, total number of schedules possible = 3! = 6
    - $< T_1 \quad T_2 \quad T_3 >$
    - $< T_1 \quad T_3 \quad T_2 >$
    - $< T_2 \quad T_1 \quad T_3 >$
    - $< T_2 \quad T_3 \quad T_1 >$
    - $< T_3 \quad T_1 \quad T_2 >$
    - $< T_3 \quad T_2 \quad T_1 >$
  -

- Final update on data items:
  - A:
  - B: $T_1 \quad T_2 \quad T_3$
  - Since the final update on B is made by T_3, so the transaction T_3 must execute after transactions T_1 and T_2
  - Therefore, $(T_1, T_2) \rightarrow T_3$, now, removing those schedules in which T_3 is not executing at last:
    - $< T_1 \quad T_2 \quad T_3 >$
    - $< T_2 \quad T_1 \quad T_3 >$

View Serializability: Example 1

- Check whether the schedule is *view serializable* or not?
  - S: $r_2(B)$, $r_2(A)$, $r_1(A)$, $r_3(A)$, $w_1(B)$, $w_2(B)$, $w_3(B)$

- Solution:
  - Initial read + which transaction updates after read?
    - A: $T_2 - T_1 - T_3$ (initial read)
    - B: T_2 (initial read), T_1 (update after read)
    - The transaction T_2 reads B initially which is updated by T_1; so T_2 must execute before T_1
    - Hence, $T_2 \rightarrow T_1$, removing those schedules in which T_2 is executing before T_1
    - $< T_2 \quad T_1 \quad T_3 >$

- Write Read sequence (WR)
  - No need to check here

- Hence, view equivalent serial schedule is:
  - $\mathbf{T_2 \rightarrow T_1 \rightarrow T_3}$

View Serializability: Example 2

- Check whether the schedule is *conflict serializable* and *view serializable* or not?
  - S: $r_1(A)$, $r_2(A)$, $r_3(A)$, $r_4(A)$, $w_1(B)$, $w_2(B)$, $w_3(B)$, $w_4(B)$

More Complex Notions of Serializability

- The schedule below produces the same outcome as the serial schedule < T_1, T_5 >, yet is not conflict equivalent or view equivalent to it

T_1	T_5
read (A) $A := A - 50$ write (A)	
	read (B) $B := B - 10$ write (B)
read (B) $B := B + 50$ write (B)	
	read (A) $A := A + 10$ write (A)

- If we start with A = 1000 and B = 2000, the final result is with A = 960 and B = 2040
- Determining such equivalence requires analysis of operations other than read and write

Next Lecture

Concurrency Control

Thank you for your attention...

Any question?

Contact:
Department of Information Technology, NITK Surathkal, India
6th Floor, Room: 13
Phone: +91-9477678768
E-mail: shrutilipi@nitk.edu.in