Lecture Notes for Lecture 9 of CS 5200 (Database Management System) for the Summer 1, 2019 session at the Northeastern University Silicon Valley Campus.

Transactions

Philip Gust,
Clinical Instructor
Department of Computer Science

Lecture 8 Review

- Stored procedures and functions allow encapsulating a set of commonly performed actions into procedures stored by the database that can be called with parameters as part of a query or in a trigger.
- Stored procedures are statements that can only return values through reference (OUT) parameters. They can only be called using a special CALL statement.
- Stored functions are values that can only return values through the function value. They can be called anywhere a value can be used, including *select* and *value* expressions and *check*.
- The language for stored procedures is non-standard and varies by database product. ApacheDB/JavaDB/Derby uses Java.

Today's Topics

- The final topic in our study of SQL databases is a mechanism called *transactions*.
- A transaction groups database operations together so that they their results are not visible outside the transaction until they either all succeed, or all of their effects are rolled back.
- We will learn about the underlying principals of transaction management, and then look at the transactional controls available through SQL and how those are performed with JDBC.

Transactions

- A *transaction* is a group of operations or tasks. A single task is the minimum processing unit which cannot be divided further.
- For example, a bank transfers $500 from A's account to B's account. This very simple transaction involves several small tasks that must be performed as unit:

A's Account

- Open account (A)
- Old_Balance = A.balance
- New_Balance = Old_Balance-500
- A.balance = New_Balance
- Close_Account (A)

B's Account

- Open account (B)
- Old_Balance = B.balance
- New_Balance = Old_Balance+500
- B.balance = New_Balance
- Close_Account (B)

Transactions

- A transaction is a set of operations that contain a group of tasks that must be performed together.
- A transaction must maintain a set of properties in order to ensure accuracy, completeness, and data integrity.
- These properties are referred to by the acronym ACID:
  - **A**tomicity
  - **C**onsistency
  - **I**solation
  - **D**urability
- We will briefly consider each of these in turn.

Transactions

Atomicity

- This property states that a transaction must be treated as an atomic unit.
- That is, either all of its operations are executed or none.
- There must be no state in a database where a transaction is left partially completed.
- At the end of the transaction, states should be defined as the sum of the operations in the transaction if it succeeds, or be restored to what they were before the transaction if it aborts/fails.

Transactions

Consistency

- The database must remain in a consistent state after any transaction.
- No transaction should have any adverse effect on the data residing in the database.
- If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Transactions

Isolation

- In a database system where more than one transaction is being executed concurrently, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.

Transactions

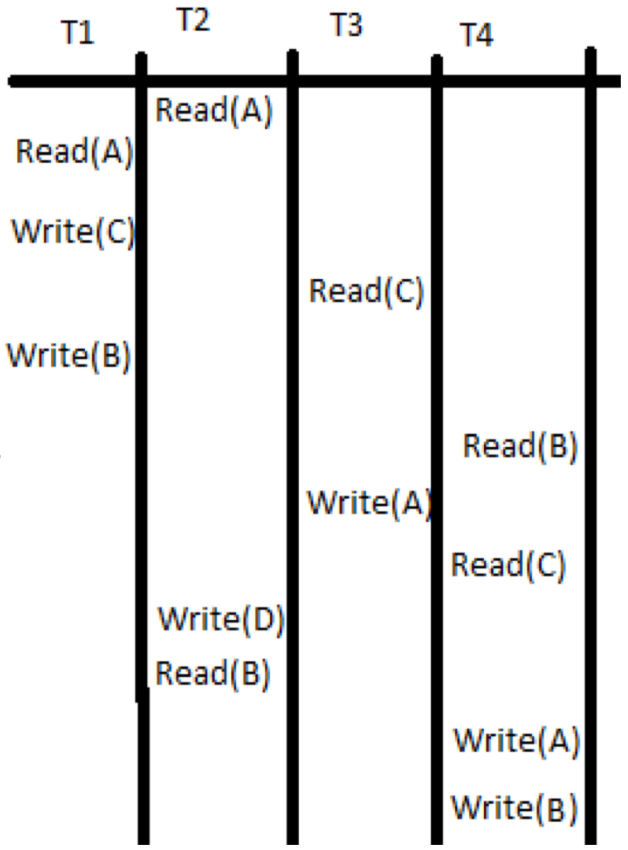
Durability

- The database should be durable enough to hold all its latest updates even if the system fails or restarts.
- If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
- If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system comes back up.

Transactions

Serializing Transaction Operations

- Although we think about transactions executing serially, one transaction finishing before the next one begins, that is not really very practical in a multiprogramming environment.
- Instead, operations in one transaction are interleaved with operations in other transactions.
- The sequence of operations in a transaction cannot be changed, but the operations across transactions can appear to occur randomly.



Transactions

Serializing Transaction Operations

- This execution does no harm if transactions are mutually independent and working on different segments of data.
- However, if two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.
- To avoid this, the DBMS must carefully schedule operations across transactions so that they have the same effect as though their transactions ran serially.

Transactions

Serializing Transaction Operations

- A *schedule* is a chronological sequence of operations that are being executed concurrently. A schedule can have many transactions in it, each with a number of instructions/tasks.
- A *serial schedule* is one in which transactions are executed serially, where the first one completes before the second one is executed.
- A *serializable schedule* is any schedule of operations from multiple transactions whose outcome is equivalent to the outcome of a serial schedule.

Transactions

Equivalence Serializable Schedules

- To avoid possible interference, a schedule must either be serial, or equivalent to a serial schedule.
- An equivalence schedule can be of the following types:
  - Result Equivalence
  - View Equivalence
  - Conflict Equivalence

Transactions

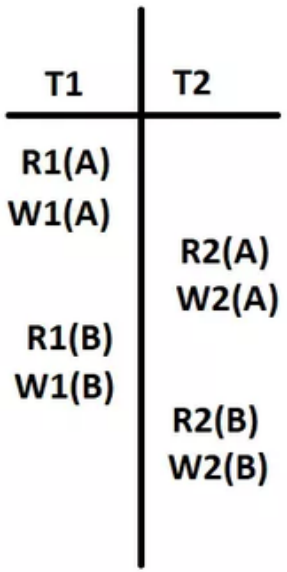
Result Equivalence

- Two schedules are result equivalent if they produce the same result after execution for a given value.
- However, the two schedules may yield different results for another set of values. That's why result equivalence is not generally considered significant.

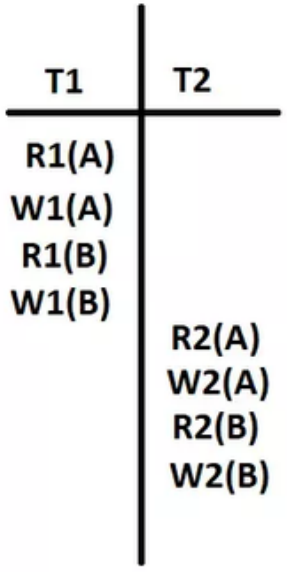
Transactions

View Equivalence

- Two schedules are view equivalent if the transactions in both the schedules perform similar actions in a similar manner for every value. Consider the following schedules:



S1 (Schedule 1)



S2 (Schedule 2)

Transactions

View Equivalence

- A schedule S is called view serializable if it is view equal to a serial schedule (no overlapping transactions).
- Two schedules S1 and S2 are said to be view equal if and only if the following conditions are satisfied :
  - *Initial Read*: If a transaction T1 reading data item A from initial database in S1 then in S2 also T1 should read A from the initial database.
  - *Updated Read*: If T1 is reading A which is updated by T2 in S1 then in S2 also T1 should read A which is updated by T2.
  - *Final Write*: If a transaction T1 updated A at last in S1, then in S2 also T1 should perform final write operations.

Transactions

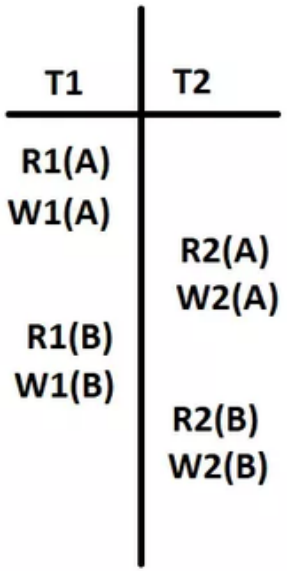
Conflict Equivalence

- Two schedules are conflicting if they have the following properties for every value.
  - Both belong to separate transactions.
  - Both accesses the same data item.
  - At least one of them is "write" operation.
- Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if
  - Both the schedules contain the same set of transactions.
  - The order of conflicting pairs of operation is maintained in both the schedules.

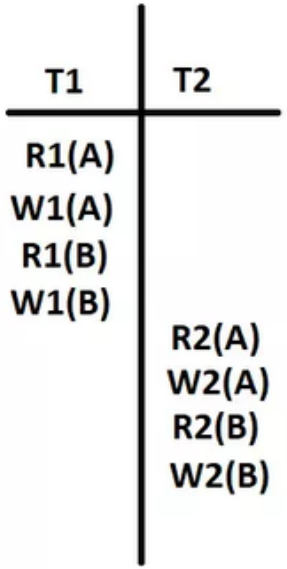
Transactions

Conflict Equivalence

- Consider again the following schedules:



S1 (Schedule 1)



S2 (Schedule 2)

Transactions

Conflict Equivalence

- The pair of operations (R1(A), W2(A)) are conflicting because they belong to two different transactions on same data item A and one of them is write operation.
- Similarly, (W1(A), W2(A)) and (W1(A), R2(A)) pairs are also conflicting.
- On the other hand, (R1(A), W2(B)) pair is non-conflicting because they operate on different data item. Similarly, (W1(A), W2(B)) pair is non-conflicting.
- Two schedules are said to be conflict equivalent when one can be transformed to another by swapping non-conflicting operations.

Transactions

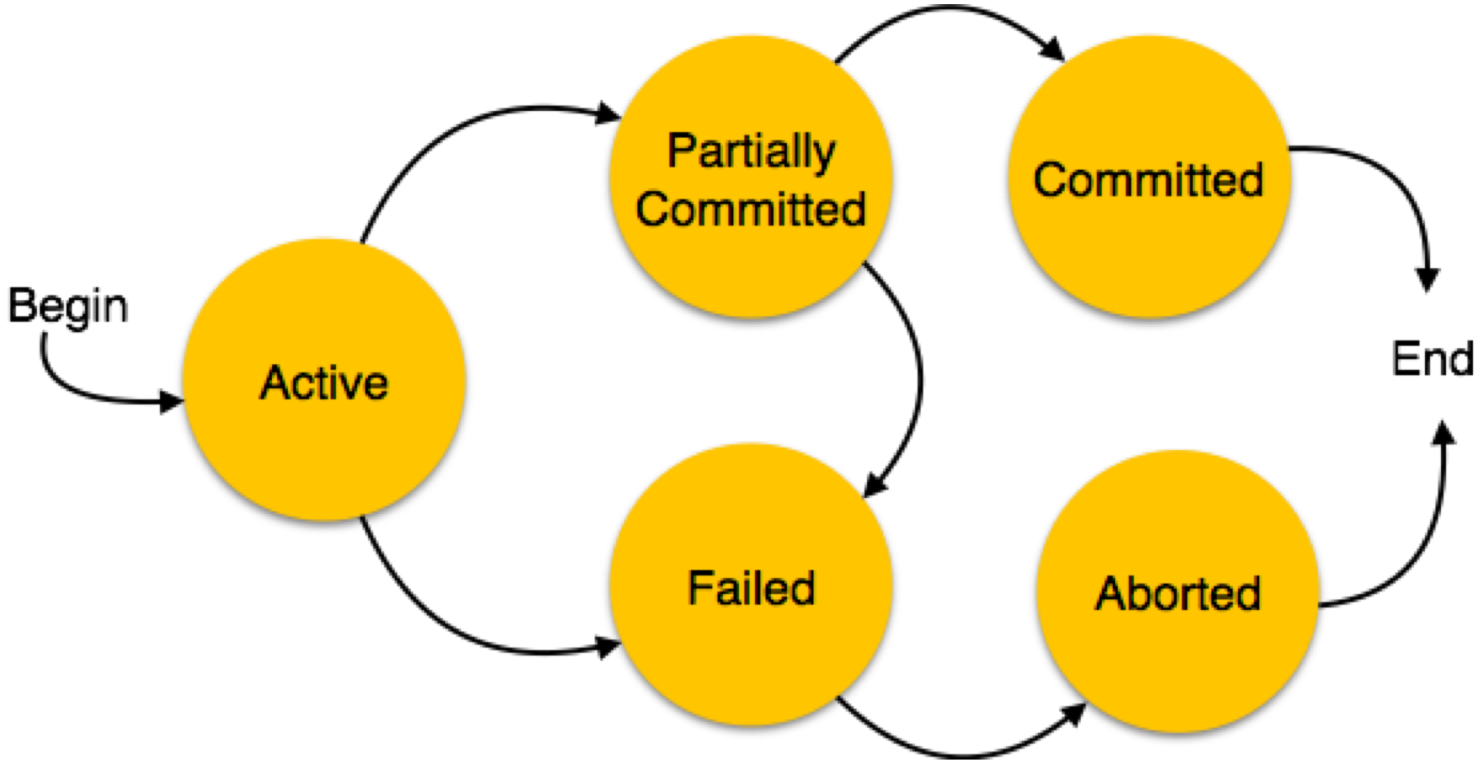
View vs Conflict Equivalence

- View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable.
- All conflict serializable schedules are also view serializable. However not all view serializable schedules are conflict serializable.

Transactions

States of Transactions

- A database transaction can be in one of the following states.



Transactions

States of Transactions

- **Active** – In this state, the transaction is being executed. This is the initial state of every transaction.
- **Partially Committed** – When a transaction executes its final operation, it is said to be in a partially committed state.
- **Committed** – If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.
- **Failed** – A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

Transaction Control

- Here are the set of commands provided by SQL 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 are only used with the DML commands such as INSERT, UPDATE and DELETE only.
- They cannot be used while creating or dropping tables because these operations are automatically committed in the database.

Transactions and JDBC

- Transactional control in JDBC is done through the Connection object.
- By default the results of each command is automatically committed. To disable this, use the statement

  conn.setAutoCommit(false)

- The following JDBC functions perform transaction control.

  conn.commit()
  conn.rollback()
  conn.setSavepoint(“SP3”)
  conn.releaseSavepoint(“SP3”)
  conn.setReadOnly(true)

Transaction Control

COMMIT

- Example: Consider the CUSTOMERS table with the following records.

Id	FamilyName	GivenName	Email
100	Smith	Mary	mary.smith@yahoo.com
110	Smith	David	david.smith@comcast.net
120	Jones	Tom	tjones@att.net
130	Rogers	Richard	rogersr@verizon.com
140	Martin	Mary	mary_r_martin@google.com
150	Peters	Elizabeth	epeters@yahoo.com
160	Elkins	Aaron	aaron_elkins@comcast.net

Transaction Control

COMMIT

- This example deletes those records from the table that have age = 25 and then COMMIT the changes in the database.

```
DELETE FROM CUSTOMERS WHERE FAMILYNAME = 'Smith';
COMMIT;
```

- Two rows from the table would be deleted produce the following result.

Id	FamilyName	GivenName	Email
120	Jones	Tom	tjones@att.net
130	Rogers	Richard	rogersr@verizon.com
140	Martin	Mary	mary_r_martin@google.com
150	Peters	Elizabeth	epeters@yahoo.com
160	Elkins	Aaron	aaron_elkins@comcast.net

Transaction Control

ROLLBACK

- 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

```
ROLLBACK;
```

Transaction Control

ROLLBACK

- Example: Consider the CUSTOMERS table with the following records.

Id	FamilyName	GivenName	Email
100	Smith	Mary	mary.smith@yahoo.com
110	Smith	David	david.smith@comcast.net
120	Jones	Tom	tjones@att.net
130	Rogers	Richard	rogersr@verizon.com
140	Martin	Mary	mary_r_martin@google.com
150	Peters	Elizabeth	epeters@yahoo.com
160	Elkins	Aaron	aaron_elkins@comcast.net

Transaction Control

ROLLBACK

- This example delete those records from the table which have the age = 25 and then ROLLBACK the changes in the database.

```
DELETE FROM CUSTOMERS WHERE FAMILYNAME = 'Smith';
ROLLBACK;
```

Id	FamilyName	GivenName	Email
100	Smith	Mary	mary.smith@yahoo.com
110	Smith	David	david.smith@comcast.net
120	Jones	Tom	tjones@att.net
130	Rogers	Richard	rogersr@verizon.com
140	Martin	Mary	mary_r_martin@google.com
150	Peters	Elizabeth	epeters@yahoo.com
160	Elkins	Aaron	aaron_elkins@comcast.net

Transaction Control

SAVEPOINT

- 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.

Transaction Control

SAVEPOINT

- The syntax for rolling back to a SAVEPOINT is as shown below.

  ROLLBACK TO SAVEPOINT_NAME;

- In the following example you plan to delete the three different records from the CUSTOMERS table.
- You want to create a SAVEPOINT before each delete, so that you can ROLLBACK to any SAVEPOINT at any time to return the appropriate data to its original state.

Transaction Control

SAVEPOINT

- Example: Consider the CUSTOMERS table with the following records.

Id	FamilyName	GivenName	Email
100	Smith	Mary	mary.smith@yahoo.com
110	Smith	David	david.smith@comcast.net
120	Jones	Tom	tjones@att.net
130	Rogers	Richard	rogersr@verizon.com
140	Martin	Mary	mary_r_martin@google.com
150	Peters	Elizabeth	epeters@yahoo.com
160	Elkins	Aaron	aaron_elkins@comcast.net

Transaction Control

SAVEPOINT

- The following code block contains the series of operations.

```
SAVEPOINT SP1;
DELETE FROM CUSTOMERS WHERE ID=100;
SAVEPOINT SP2;
DELETE FROM CUSTOMERS WHERE ID=110;
SAVEPOINT SP3;
DELETE FROM CUSTOMERS WHERE ID=120;
```

- After the three deletions, you ROLLBACK to the SAVEPOINT identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone

```
ROLLBACK TO SP2;
```

Transaction Control

SAVEPOINT

- Only the first deletion took place since rolling back to SP2.

Id	FamilyName	GivenName	Email
110	Smith	David	david.smith@comcast.net
120	Jones	Tom	tjones@att.net
130	Rogers	Richard	rogersr@verizon.com
140	Martin	Mary	mary_r_martin@google.com
150	Peters	Elizabeth	epeters@yahoo.com
160	Elkins	Aaron	aaron_elkins@comcast.net

Transaction Control

RELEASE SAVEPOINT

- 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.

Transaction Control

SET TRANSACTION

- 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 ];
```

Transaction Control

Example

- Consider again the bank transaction of transferring money from one account to another. To ensure consistency, the operations for the transfer must be done in one transaction.

Set Transaction Read Write

A's Account

- Open account (A)
- Old_Balance = A.balance
- New_Balance = Old_Balance-500
- A.balance = New_Balance
- Close_Account (A)

B's Account

- Open account (B)
- Old_Balance = B.balance
- New_Balance = Old_Balance+500
- B.balance = New_Balance
- Close_Account (B)

Commit

Transactions and JDBC

- **Statement Versus Transaction Runtime Rollback**
- When an SQL statement generates an exception, it results in a *runtime rollback*.
- A runtime rollback is a system-generated rollback of a statement or transaction by the Derby, as opposed to an explicit *rollback* call from your application.
- Extremely severe exceptions, such as disk-full errors, shut down the system, and the transaction is rolled back when the database is next booted.
- Severe exceptions, such as deadlock, cause transaction rollback; Derby rolls back all changes since the beginning of the transaction and implicitly begins a new transaction.
- Less severe exceptions, such as syntax errors, result in statement rollback; Derby rolls back only changes made by the statement that caused the error. The application developer can insert code to explicitly roll back the entire transaction if desired.