# Logging and Recovery

444 Section, April 23, 2009

## Reminders

- Project 2 out: Due Wednesday, Nov. 4, 2009
- Homework 1: Due Wednesday, Oct. 28, 2009

# Outline

- Project 2: JDBC
- ACID: Recovery
  - Undo, Redo logging

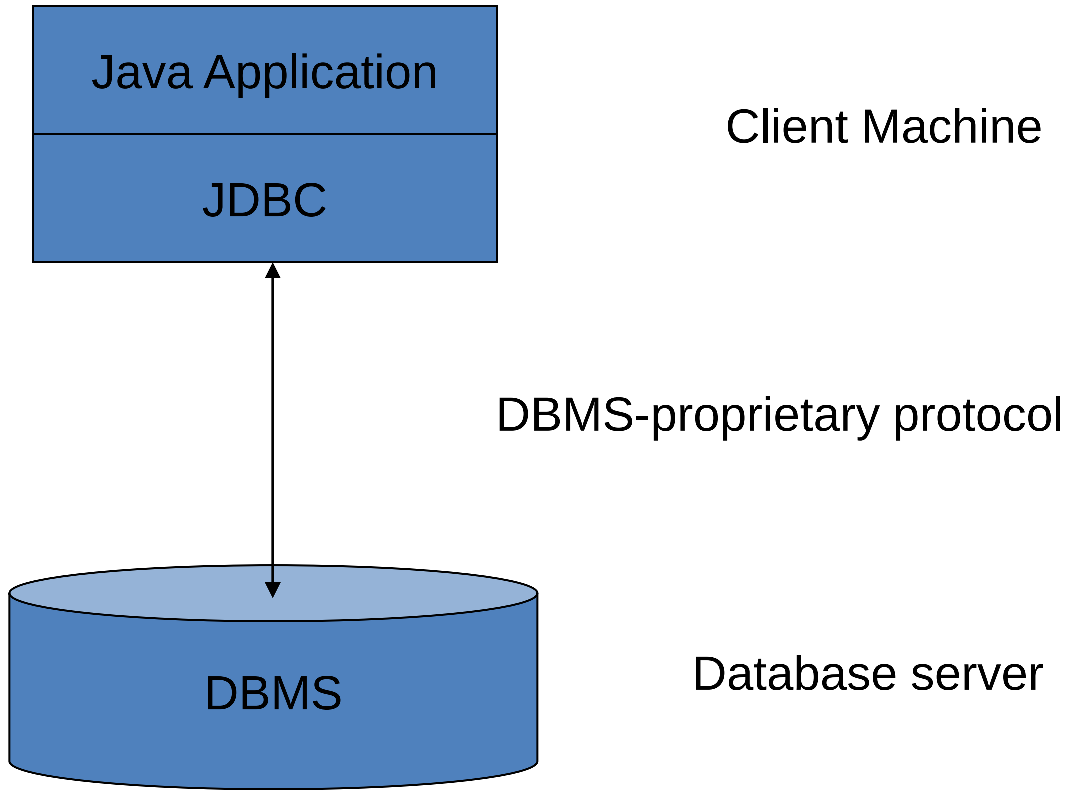
# JDBC

- Java API to access database
  1. Connect to a data source
  2. Send queries and update statements
  3. Retrieve and process results

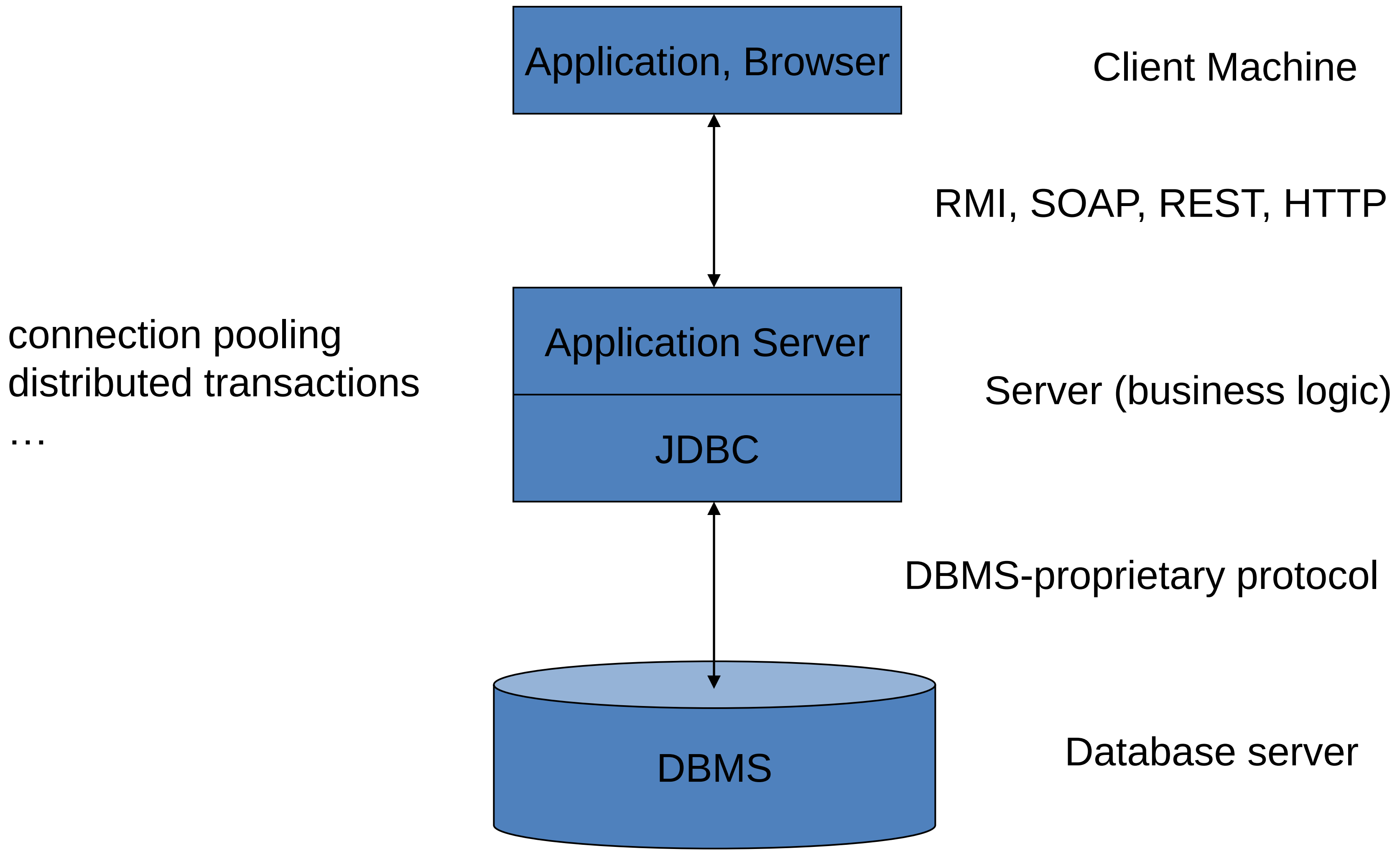
## JDBC Example

```
Connection con = DriverManager.getConnection
      ("jdbc:sqlserver://iisqlsrv:database=imdb",
       "myLogin", "myPassword");

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery
      ("SELECT a, b, c FROM Table1");

while (rs.next()) {
      int x = rs.getInt("a");
      String s = rs.getString("b");
      float f = rs.getFloat("c");
}
```

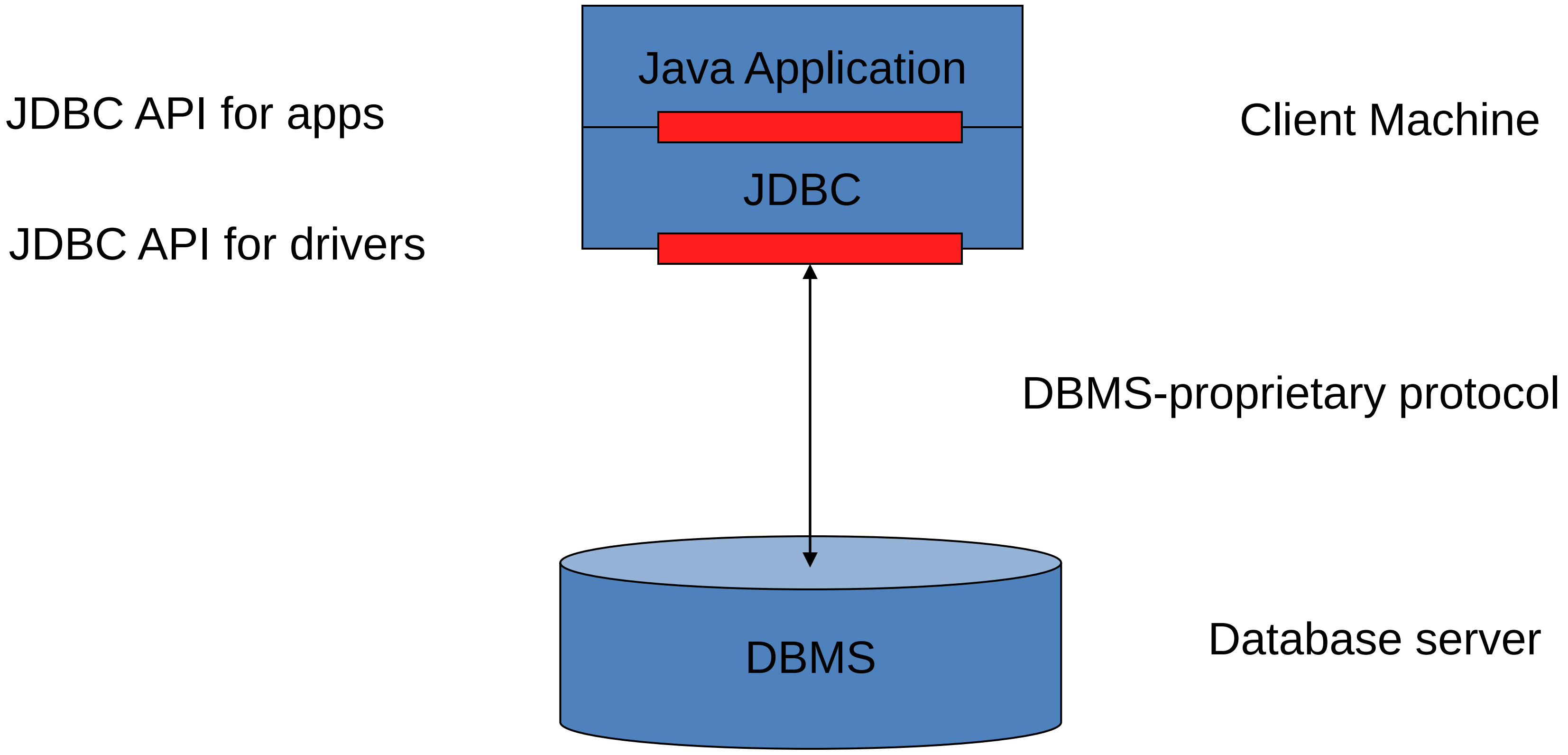
# JDBC Architecture
# Two-tier model



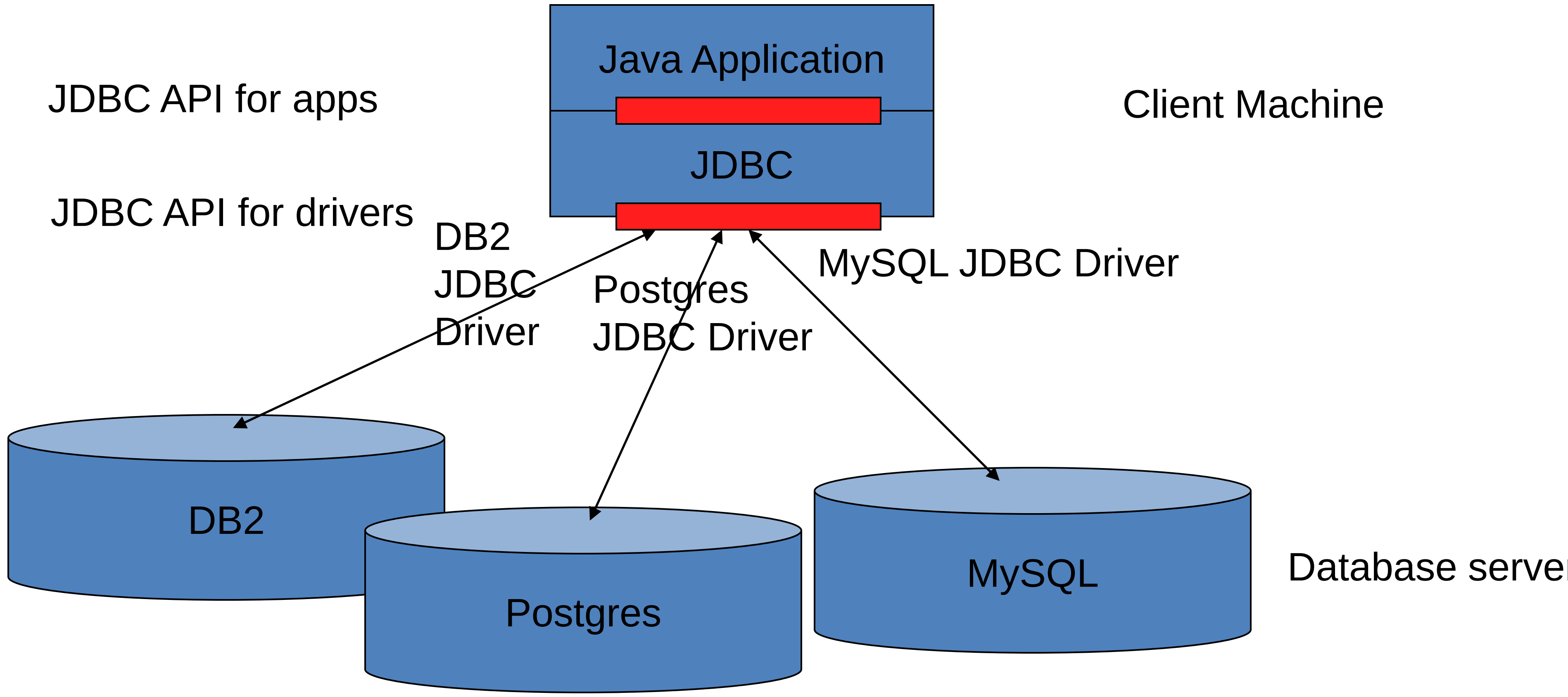
# JDBC Architecture
# Three-tier model



# JDBC API


JDBC API for apps
JDBC API for drivers
Java Application
JDBC
Client Machine
DBMS-proprietary protocol
DBMS
Database server

# JDBC API


JDBC API for apps
JDBC API for drivers
Java Application
JDBC
Client Machine
DB2 JDBC Driver
Postgres JDBC Driver
MySQL JDBC Driver
DB2
Postgres
MySQL
Database serve

# First need to load driver

- SQL Server Driver
  sqljdbc4.jar
- Postgres Driver
  postgresql-8.3-603.jdbc4.jar

- Put on class path, then tell Java to load it

```
Class.forName(“com.microsoft.sqlserver.jdbc.SQLServerDriver “);
Class.forName(“org.postgresql.Driver”);
Class.forName
   (“com.microsoft.sqlserver.jdbc.SQLServerDriver”);

Connection con = DriverManager.getConnection
      ("jdbc:sqlserver://iisqlsrv:database=imdb",
       "myLogin", "myPassword");

PreparedStatement pstmt = con.prepareStatement
      (“SELECT lname FROM persons WHERE id = ?”);

pstmt.setInt(1, 34);
ResultSet rs = pstmt.executeQuery();

while (rs.next()) {
      String s = rs.getString(“lname");
}

rs.close();

pstmt.close();

con.close();
```

```
Class.forName
   ("com.microsoft.sqlserver.jdbc.SQLServerDriver");

Connection con = null;

try {
        con = DriverManager.getConnection( … );


        …


} catch (Exception e) {
        e.printStackTrace();
} finally {
        con.close();
}
```

## PreparedStatement

```
PreparedStatement pstmt = con.prepareStatement
    (“SELECT lname FROM persons WHERE id = ?”);
…
pstmt.setInt(1, 34);
ResultSet = pstmt.executeQuery();
…

pstmt.setInt(1, 63);
ResultSet = pstmt.executeQuery();

…
```

---

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery
      ("SELECT a, b, c FROM Table1");
```

# PreparedStatement

- No need to worry about quotes ‘, “

```
PreparedStatement pstmt = con.prepareStatement
    (“SELECT website FROM shops
       WHERE name = ? OR owner = ?”);
…
pstmt.setString(1, “George’s”);
pstmt.setString(2, “Oh \”wow\”!”);
…
```

---

```
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery
      ("SELECT website FROM shops
        WHERE name = \”George’s\” OR …");
```

# Transactions – Option 1

```
String sBeginTx    = “BEGIN TRANSACTION”;
String sCommitTx   = “COMMIT TRANSACTION”;
String sRollbackTx = “ROLLBACK TRANSACTION”;

PreparedStatement pBeginTx   =
      con.prepareStatement(sBeginTx);
PreparedStatement pCommitTx   =
      con.prepareStatement(sCommitTx);

PreparedStatement pRollbackTx =
      con.prepareStatement(sRollbackTx);
…
pBeginTx.executeUpdate();
…
if (ok) pCommitTx.executeUpdate();
else pRollbackTx.executeUpdate();
```

# Transactions – Option 2

```
…
con.setAutoCommit(false);
…
if (ok) con.commit();
else con.rollback();

con.setAutoCommit(true);
```

## Why do we need to recover a DB?

## How can logging make recovery easier/better?

# Our undo log notation

- <START T>
  - Transaction T has begun
- <COMMIT T>
  - T has committed
- <ABORT T>
  - T has aborted
- <T,X,v>   -- Update record
  - T has updated element X, and its *old* value was v

# An undo logging problem

- Given this undo log, when can each data item be written to disk?

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<T2,E,e>** |
| 9 | **<START T4>** |
| 10 | **<T4,F,f>** |
| 11 | **<T3,G,g>** |
| 12 | **<COMMIT T2>** |

# Undo logging problem, continued

- After writing these log entries, the DBMS crashes. What does it do when it restarts?

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<T2,E,e>** |
| 9 | **<START T4>** |
| 10 | **<T4,F,f>** |
| 11 | **<T3,G,g>** |
| 12 | **<COMMIT T2>** |

# Undo logging problem, part 3

- Now suppose the DBMS crashes after log entry 8.  What does it do when it restarts?

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<T2,E,e>** |
| 9 | **<START T4>** |
| 10 | **<T4,F,f>** |
| 11 | **<T3,G,g>** |
| 12 | **<COMMIT T2>** |

# What if it was a redo log?

- Now, <T,X,v> means X’s *new* value is v
- So now when can we write each datum?

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<T2,E,e>** |
| 9 | **<START T4>** |
| 10 | **<T4,F,f>** |
| 11 | **<T3,G,g>** |
| 12 | **<COMMIT T2>** |

# Redo log problem, continued

- What do we do after the DBMS crashes and restarts?

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<T2,E,e>** |
| 9 | **<START T4>** |
| 10 | **<T4,F,f>** |
| 11 | **<T3,G,g>** |
| 12 | **<COMMIT T2>** |

## Log checkpointing

- Why would we add (non-quiescent) checkpoints to a log?

# Garcia-Molina, problem 17.2.7(i)

- Given this undo log, suppose we add a START CKPT after each of:
  - 2: <S,A,60>
  - 5: <T,A,10>
  - 7: <U,B,20>
  - 10: <U,D,40>
  - 13: <T,E,50>
- When is the earliest time that the END CKPT can be placed?

| | |
|---|---|
| 1 | **<START S>** |
| 2 | **<S, A, 60>** |
| 3 | **<COMMIT S>** |
| 4 | **<START T>** |
| 5 | **<T, A, 10>** |
| 6 | **<START U>** |
| 7 | **<U, B, 20>** |
| 8 | **<T, C, 30>** |
| 9 | **<START V>** |
| 10 | **<U, D, 40>** |
| 11 | **<V, F, 70>** |
| 12 | **<COMMIT U>** |
| 13 | **<T, E, 50>** |
| 14 | **<COMMIT T>** |
| 15 | **<V, B, 80>** |
| 16 | **<COMMIT V>** |

# Checkpoints look different in undo and redo logs!

- Which is the undo log and which the redo log?

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<COMMIT T1>** |
| 9 | **<START CKPT(T2,T3)>** |
| 10 | **<T2,E,e>** |
| 11 | **<START T4>** |
| 12 | **<T4,F,f>** |
| 13 | **<T3,G,g>** |
| 14 | **<COMMIT T3>** |
| 15 | **<END CKPT>** |
| 16 | **<COMMIT T2>** |
| 17 | **<COMMIT T4>** |

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<COMMIT T1>** |
| 9 | **<START CKPT(T2,T3)>** |
| 10 | **<T2,E,e>** |
| 11 | **<START T4>** |
| 12 | **<T4,F,f>** |
| 13 | **<T3,G,g>** |
| 14 | **<COMMIT T3>** |
| 15 | **<COMMIT T2>** |
| 16 | **<END CKPT>** |
| 17 | **<COMMIT T4>** |

## Undo-log recovery with checkpoints

- The DBMS crashed with the previous undo log; what do we do to recover?
  - Which log entries are read?
  - Which data do we change?

| | |
|---|---|
| 1 | <START T1> |
| 2 | <T1, A, a> |
| 3 | <T1, B, b> |
| 4 | <START T2> |
| 5 | <T2 C, c> |
| 6 | <START T3> |
| 7 | <T3 D, d> |
| 8 | <COMMIT T1> |
| 9 | <START CKPT(T2,T3)> |
| 10 | <T2,E,e> |
| 11 | <START T4> |
| 12 | <T4,F,f> |
| 13 | <T3,G,g> |
| 14 | <COMMIT T3> |
| 15 | <COMMIT T2> |
| 16 | <END CKPT> |
| 17 | <COMMIT T4> |

# Redo-log recovery with checkpoints

- Now we'd like to recover the redo log:
  - Which log entries are read?
  - Which transactions are redone?
  - Which data do we change?

| | |
|---|---|
| 1 | **<START T1>** |
| 2 | **<T1, A, a>** |
| 3 | **<T1, B, b>** |
| 4 | **<START T2>** |
| 5 | **<T2 C, c>** |
| 6 | **<START T3>** |
| 7 | **<T3 D, d>** |
| 8 | **<COMMIT T1>** |
| 9 | **<START CKPT(T2,T3)>** |
| 10 | **<T2,E,e>** |
| 11 | **<START T4>** |
| 12 | **<T4,F,f>** |
| 13 | **<T3,G,g>** |
| 14 | **<COMMIT T3>** |
| 15 | **<END CKPT>** |
| 16 | **<COMMIT T2>** |
| 17 | **<COMMIT T4>** |

# A slightly harder problem

- New notation: <OUTPUT X>
  - Write data item X to disk
- What are the legal sequences of events that consist of the entries in the following undo log, *plus* interspersed OUTPUT events?

| | |
|---|---|
| 1 | **<START T>** |
| 2 | **<T, A, 10>** |
| 3 | **<T, B, 20>** |
| 4 | **<COMMIT T>** |

# A slightly harder problem, continued

- Can you derive a formula for the number of schedules given some number *n* of actions, modifying *n* distinct data items?
  - Hint, try listing the schedules for this 3-datum example

| | |
|---|---|
| 1 | **<START T>** |
| 2 | **<T, A, 10>** |
| 3 | **<T, B, 20>** |
| 4 | **<T, C, 30>** |
| 5 | **<COMMIT T>** |