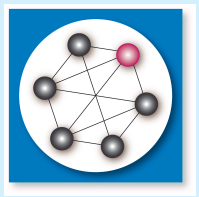


www.dbtechnet.org

Transactions in Java programming - an JDBC overview

DBTech VET Tutorial/Lecture8

by Martti Laiho, 2013-06-17

(originally scheduled to 2013-06-13)

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Objectives

- To demonstrate transaction API differencies between application programming and interactive SQL
- To explain the BankTransfer example of Appendix 2 in the ”SQL Transactions Student Guide”, focusing only on basic JDBC
  - JDBC is the first and still a basic data access API in Java programming
  - We are not trying to provide an example of good Java programming style
  - We are not covering the more advanced JDBC programming topics, but just ”Hello World” level introduction to JDBC and DebianDB Lab
- To explain the ”universal data access” by ODBC and JDBC which tries to hide the SQL differencies between the DBMS products, which however cannot hide the behaviors of different DBMS products

Agenda

1. *"Universal Data Access" (UDA) and Drivers*
   + MetaData demo
2. *An overview of JDBC API*
   + the minimalistic JdbcDemo program
3. *Explaining the BankTransfer program*
   (see Appendix 2 in the Student Guide)
4. *Listing some more Advanced Topics ..*

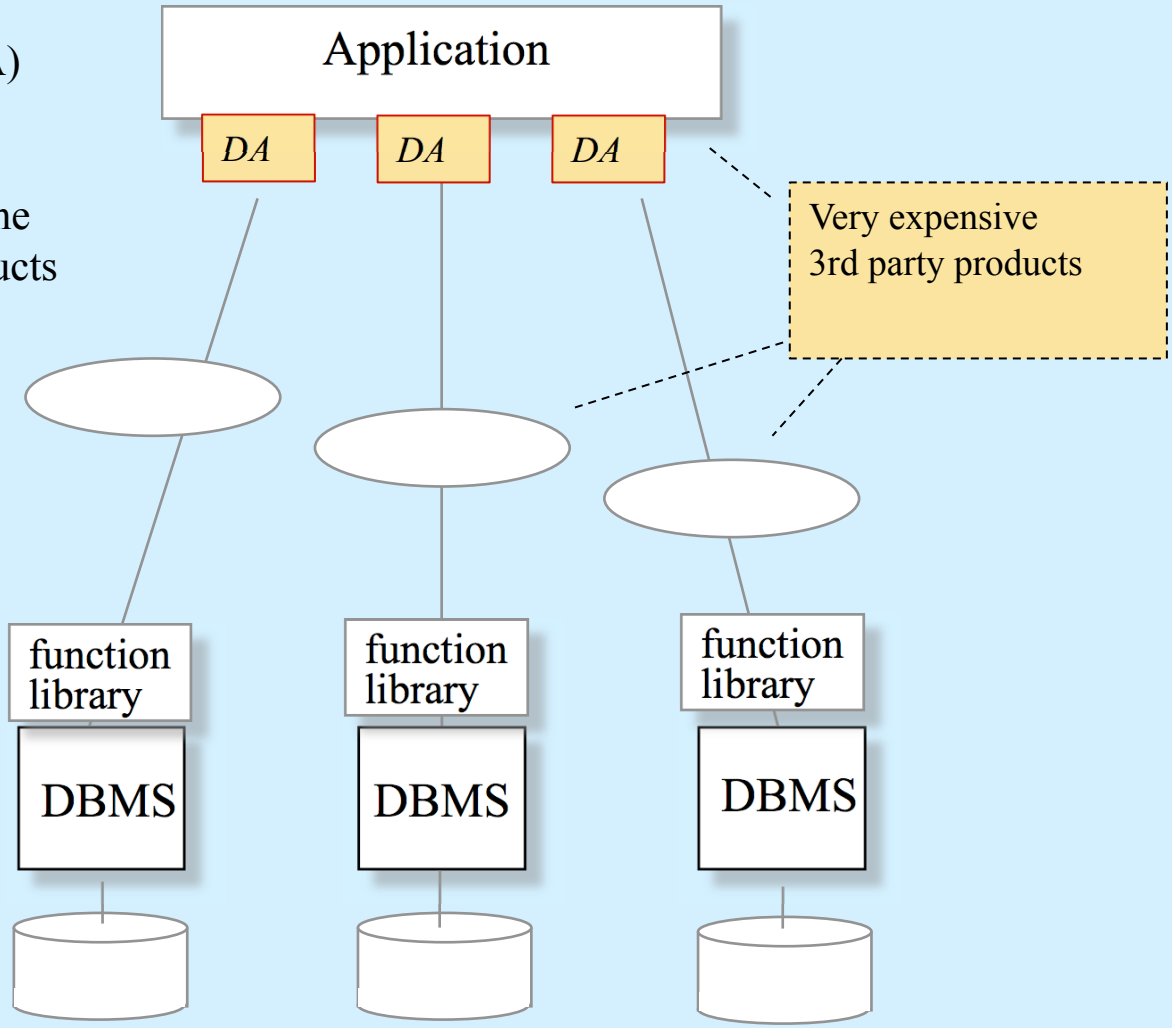
1. "Universal Data Access" (UDA) and Drivers

- Proprieatary solutions before UDA
- UDA architectures started by ODBC
- Types of JDBC implementations
- JDBC solutions of Oracle (example)
- JDBC drivers in the DebianDB virtual laboratory
- An overview of JDBC API
- JDBC Meta Data + demo

At Time of Proprietary Data Access

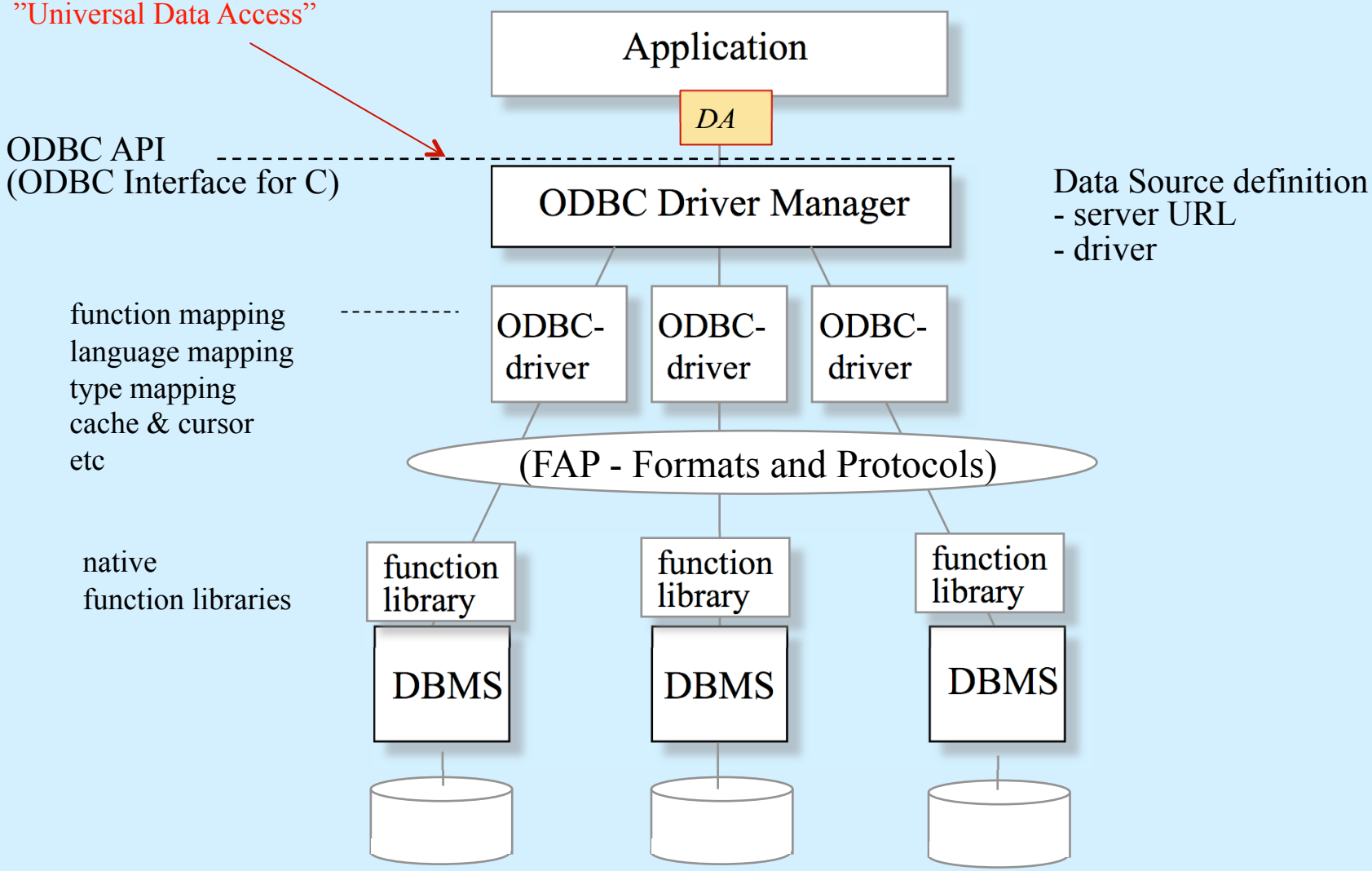
”On stone age”

Data access code (DA) was tailored to access the proprietary function libraries of the possible DBMS products



The ODBC Architecture

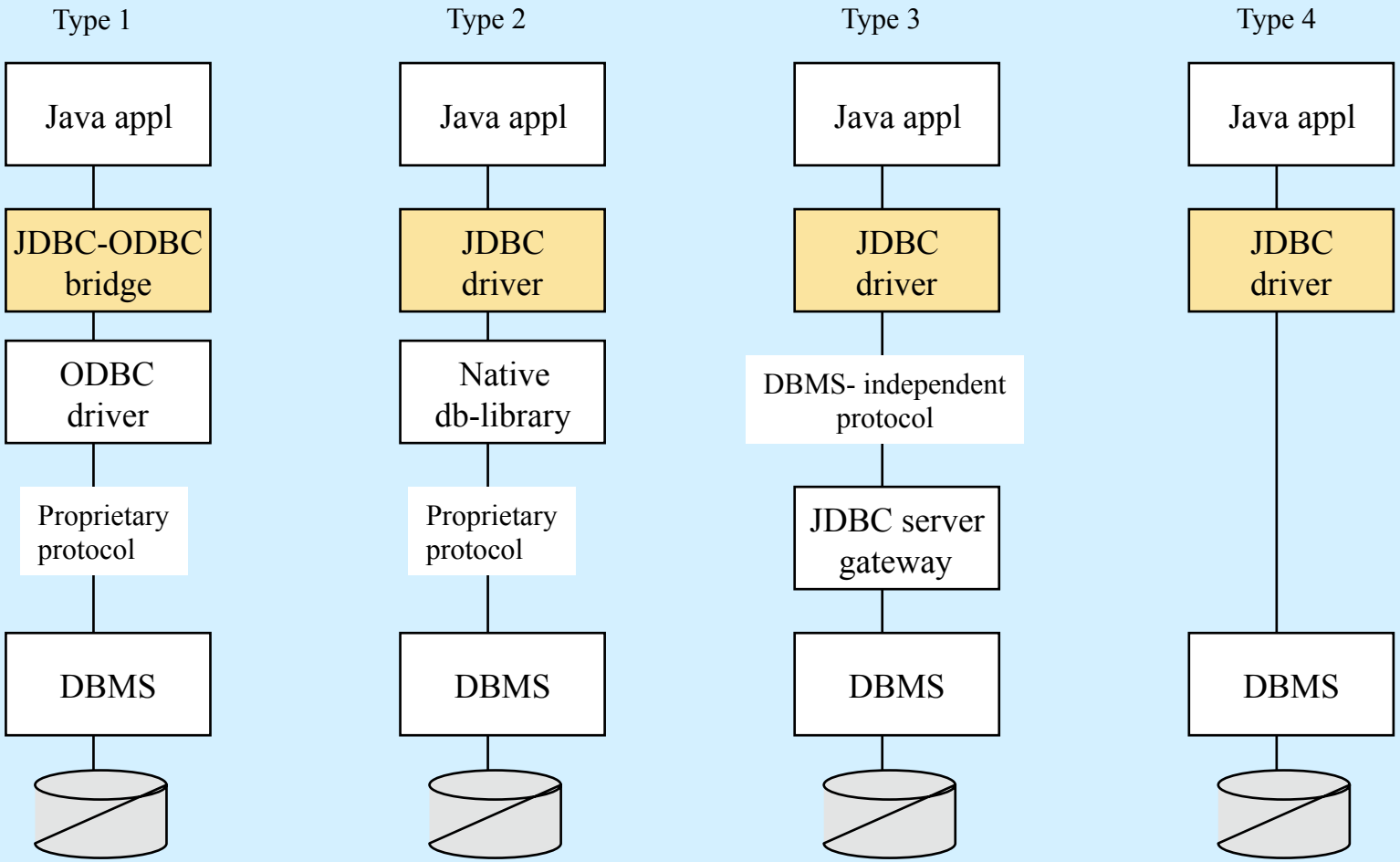
Then came ODBC providing
"Universal Data Access"



"Universal Data Access" model of ODBC
was adapted for Java by JDBC

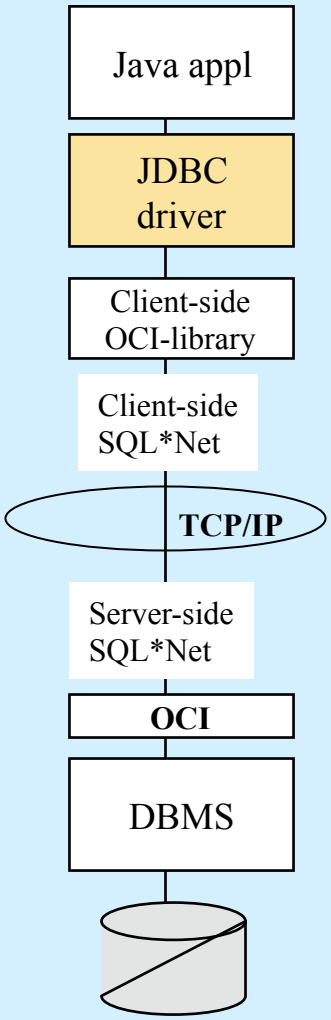
Types of JDBC Implementations

- Melton & Eisenberg



- Oracle JDBC/OCI

JDBC connections to Oracle

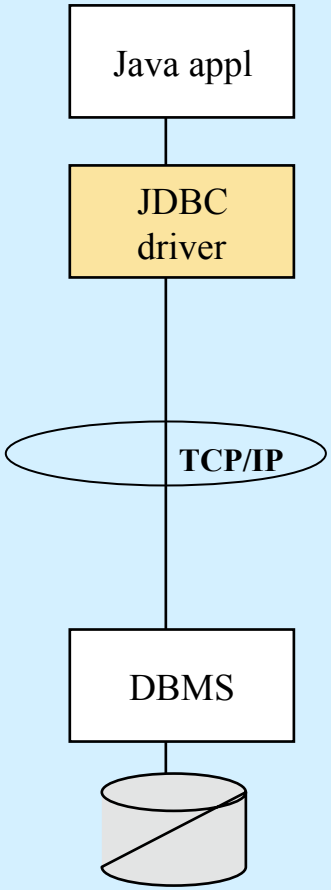


Type 2

- Oracle JDBC/OCI

Type 4

- Oracle Thin JDBC



JDBC Drivers in the DebianDB Lab

Available at www.dbtechnet.org/VET/EN/jdbc-drivers.zip

DB2:

MySQL:

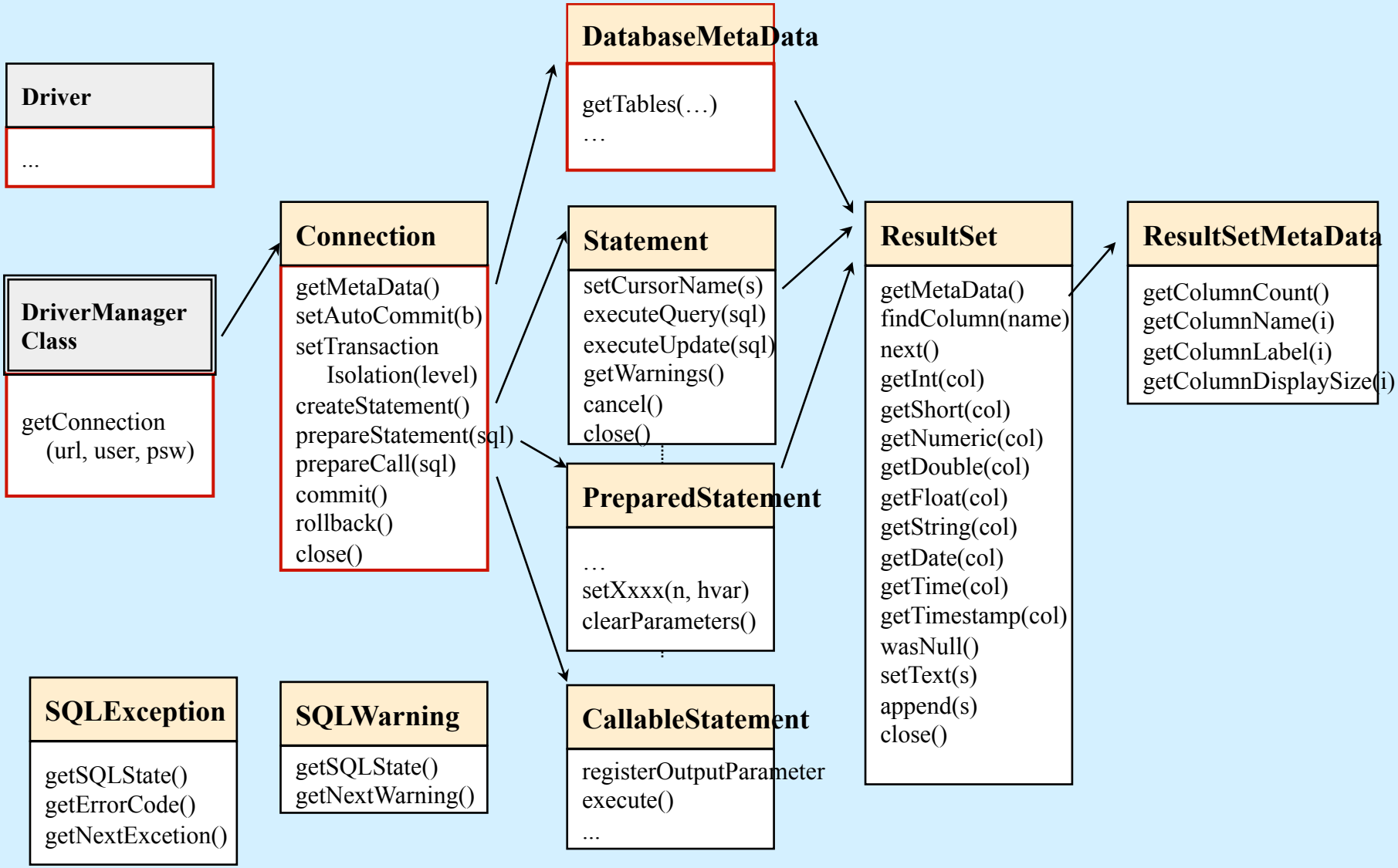
Oracle:

PostgreSQL:

Name	Type	Size	Compress...	Password ...
db2jcc.jar	jarfile	3 073 KB	2 921 KB	No
db2jcc_license_cu.jar	jarfile	1 KB	1 KB	No
db2jcc4.jar	jarfile	3 236 KB	3 079 KB	No
mysql-connector-java-5.1.23-bin.jar	jarfile	824 KB	788 KB	No
mysql-connector-java-5.1.25.zip	zip Archive	3 854 KB	3 854 KB	No
ojdbc6.jar	jarfile	2 651 KB	2 537 KB	No
postgresql-9.2-1002.jdbc4.jar	jarfile	567 KB	540 KB	No

Note: There is no standard directory in Linux for JDBC drivers.
In DebianDB we recommend JDBC drivers to be implemented in /opt/jdbc-drivers
For the details, see Listing 2-3 in the Student Guide Appendix 2

Objects and Methods in JDBC API



JDBC Metadata

Metadata available about

- the driver
- the DBMS product
  - but not about concurrency control mechanism, etc
  - as the driver reports, not necessary verified !
- the database objects (table structures, etc)

A complete example code in DBMetaData.java

DBMetaData about Oracle XE

```
# Oracle XE
export CLASSPATH=.:/opt/jdbc-drivers/ojdbc6.jar
export DRIVER=oracle.jdbc.driver.OracleDriver
export URL=jdbc:oracle:thin:@localhost:1521:xe
export USER=student
export PASSWORD=password
java DBMetaData $DRIVER $URL $USER $PASSWORD

DBMetaData version 0.2
Database product name: Oracle
Database product version: Oracle Database 10g Express Edition Release 10.2.0.1.0 - Production
Driver name: Oracle JDBC driver
Driver version: 11.2.0.2.0
Driver major version: 11
Driver minor version: 2
JDBC major version for the driver: 11
JDBC minor version for the driver: 2
Maximum number of concurrent connections to this database: 0
Maximum number of active statements to this database: 0
Supports ISO SQL concatenation of NULL values: true
NULL values are sorted at the start regardless of sort order: false
NULL values are sorted at the end regardless of sort order: false
NULL values are sorted low regardless of sort order: true
NULL values are sorted high regardless of sort order: false
Supports retrieving of generated keys: true
DDL autocommitted: true
Supports SELECT .. FOR UPDATE: true
Supports open cursors across commit: false
Supports open cursors across rollback: false
Supports Isolation Level None [0]: false
Supports Isolation Level Read UnCommitted [1]: false
Supports Isolation Level Read Committed [2]: true
Supports Isolation Level Repeatable Read [3]: false
Supports Isolation Level Serializable [4]: true
Supports savepoints: true
Default Isolation Level: 2
Supports multiple transactions open at once (on different connections): true
Supports statement pooling: true
```

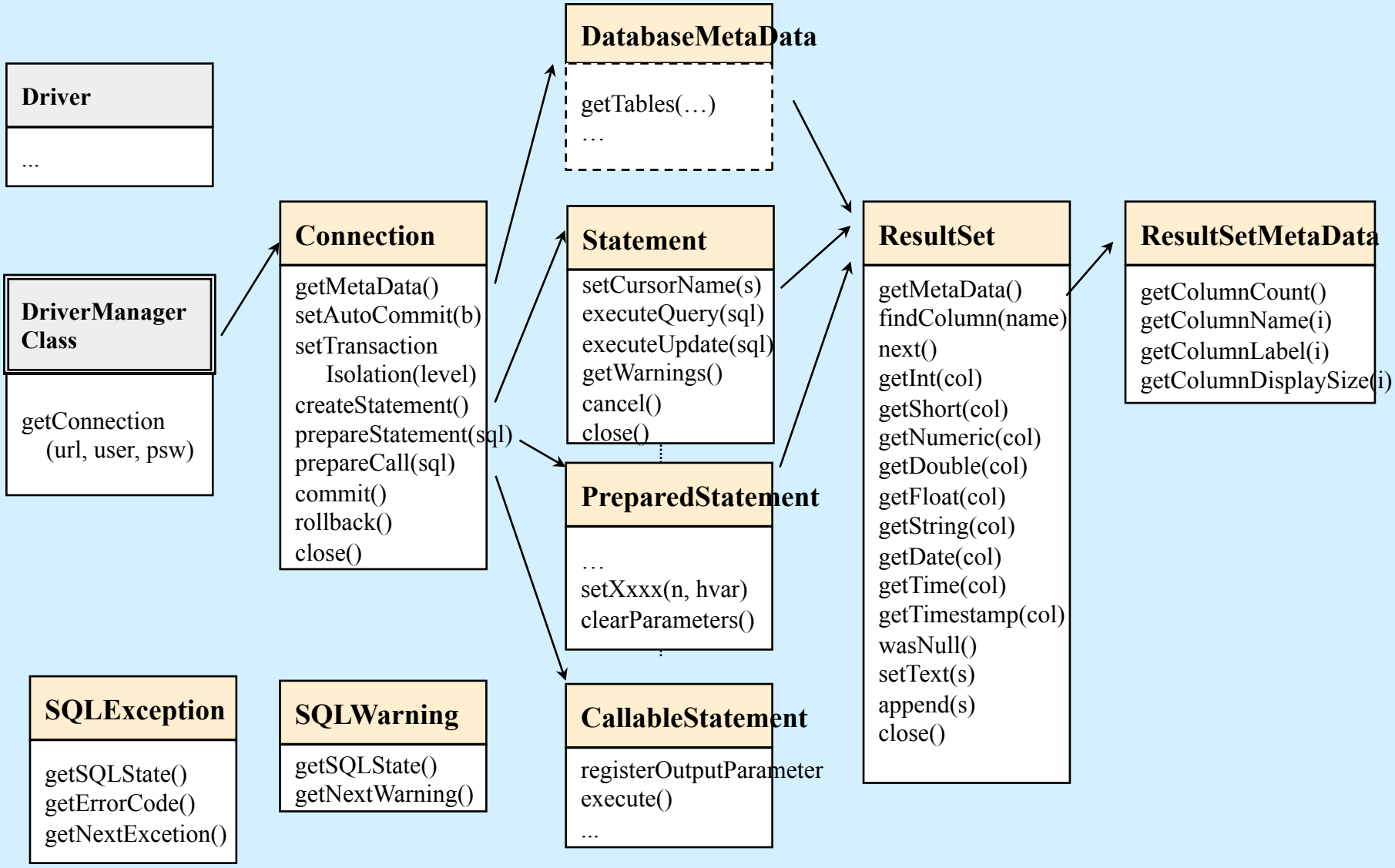
DBMetaData Demo?

Questions ?

2. Overview of JDBC API

- Objects and methods in JDBC API
- Opening a database connection (SQL-session)
- Client/Server dialogues
- Exception handling
- JDBC transactions
- Isolation levels
- A transaction template
- On means of unifying SQL by JDBC escape macros
- Sequence diagram of SQL Query
- A minimalistic JDBC demo

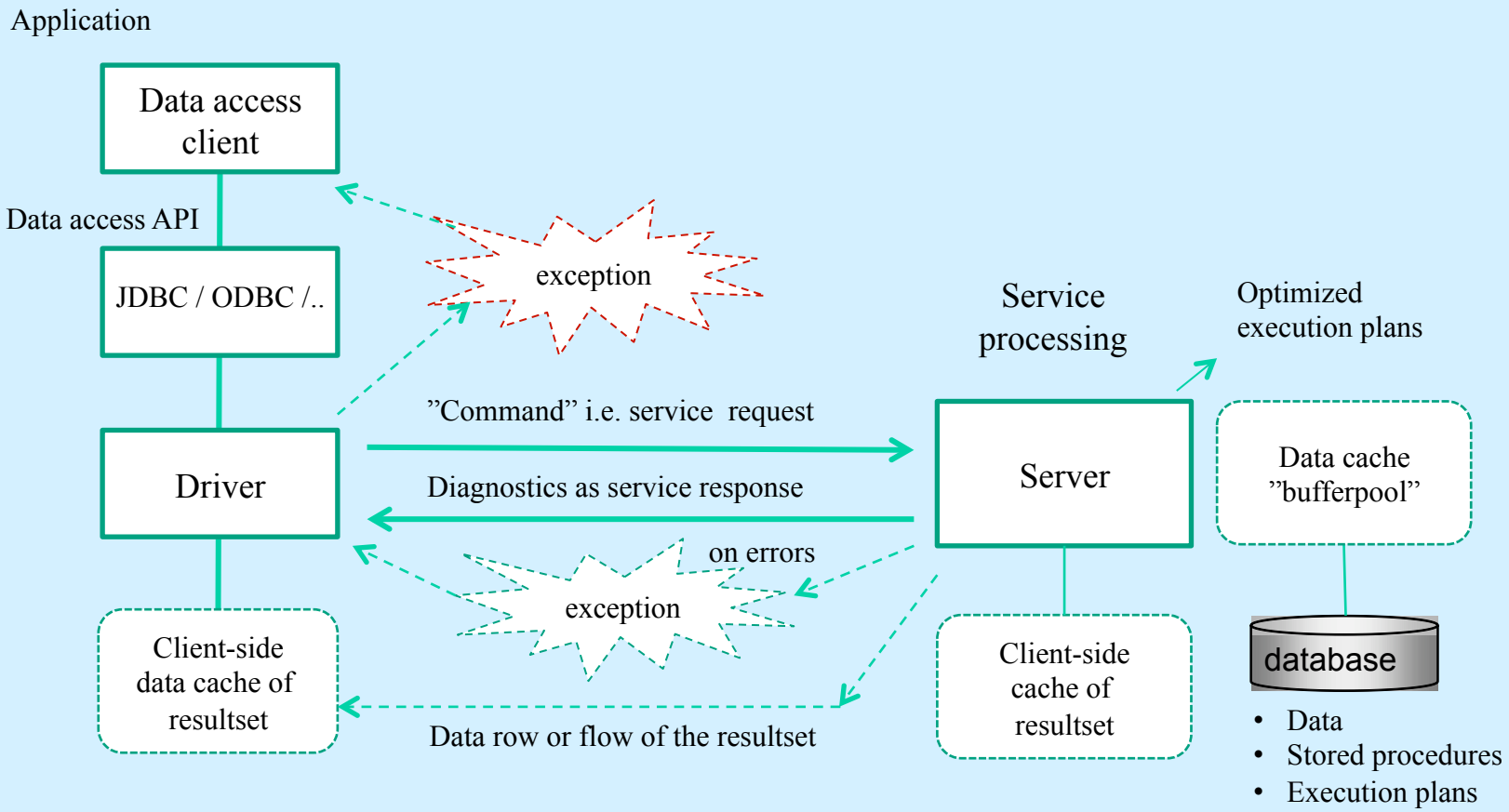
Objects and Methods in JDBC API



Opening Database Connection

```
// registering the JDBC driver and opening the database connection
try {
    Class.forName(driver);
    conn = java.sql.DriverManager.getConnection(URL,user,password);
}
catch (SQLException ex) {
      System.out.println("URL: " + URL);
      System.out.println("** Connection failure: " + ex.getMessage() +
                         "\n SQLSTATE: " + ex.getSQLState() +
                         " SQLcode: " + ex.getErrorCode());
      System.exit(-1);
}
```

Client/Server Dialogue



Exception handling

Example adapted from Core JAVA Vol II ch 4 p 206

```
try {
    jdbc method calls ...
}
catch (SQLException ex) {
   System.out.println (”\nSQLException:");
   while (ex != null) {
      System.out.println (”SQLState: "+ex.getSQLState());
      System.out.println (”Message:  "+ ex.getMessage());
      System.out.println (”Sqlcode:   "+ ex.getErrorCode());
      ex = ex.getNextException();
   }
}
catch (java.lang.Exception ex) {
    System.out.println("Exception: " + ex);
    ex.printStackTrace ();
}
```

Note: SQLWarnings do not raise exceptions, but need to be tested by the application code!

JDBC Transactions

Default: AutoCommit mode (independent of the DBMS product)

Methods:

// conn is the object managing the database connection

```
conn.setAutoCommit(false);
int level = conn.getTransactionIsolation();     // get
conn.setTransactionIsolation(level);            // set
conn.commit();
conn.rollback();
```

Isolation Levels:

0	TRANSACTION_NONE (read-only)
1	TRANSACTION_READ_UNCOMMITTED
2	TRANSACTION_READ_COMMITTED
3	TRANSACTION_REAPEATABLE_READ
4	TRANSACTION_SERIALIZABLE

JDBC and Isolation Levels

- The JDBC isolation levels are based on the ISO SQL standard
- The JDBC driver maps these to appropriate isolation levels of the DBMS product
  - For example mapping isolation levels of DB2 as follows: UR => read uncommitted, CS => currently committed (new semantics), RS => repeatable read, RR => serializable
  - Mapping to unsupported isolation levels depends on the driver raising SQLException or to more restricting isolation level
  - In transactional mode the default isolation level depends on the DBMS product

An JDBC Transaction Template

```
try {
     conn.setAutoCommit(false);   // transaction begins
     conn.setTransactionIsolation(
             Connection.TRANSACTION_SERIALIZABLE);
     // transaction logic using Java and
     // JDBC with SQL statements as parameters
     // preferrably using prepared JDBC statements
     // ...
     conn.commit();
}
catch (SQLException ex) {
     // handling of JDBC exceptions
     // ...
     conn.rollback();
}
catch (Exception ex) {
     // handling of other exceptions
     // ...
     conn.rollback();
}
finally {
     // completing the processing
     // ...
}
```

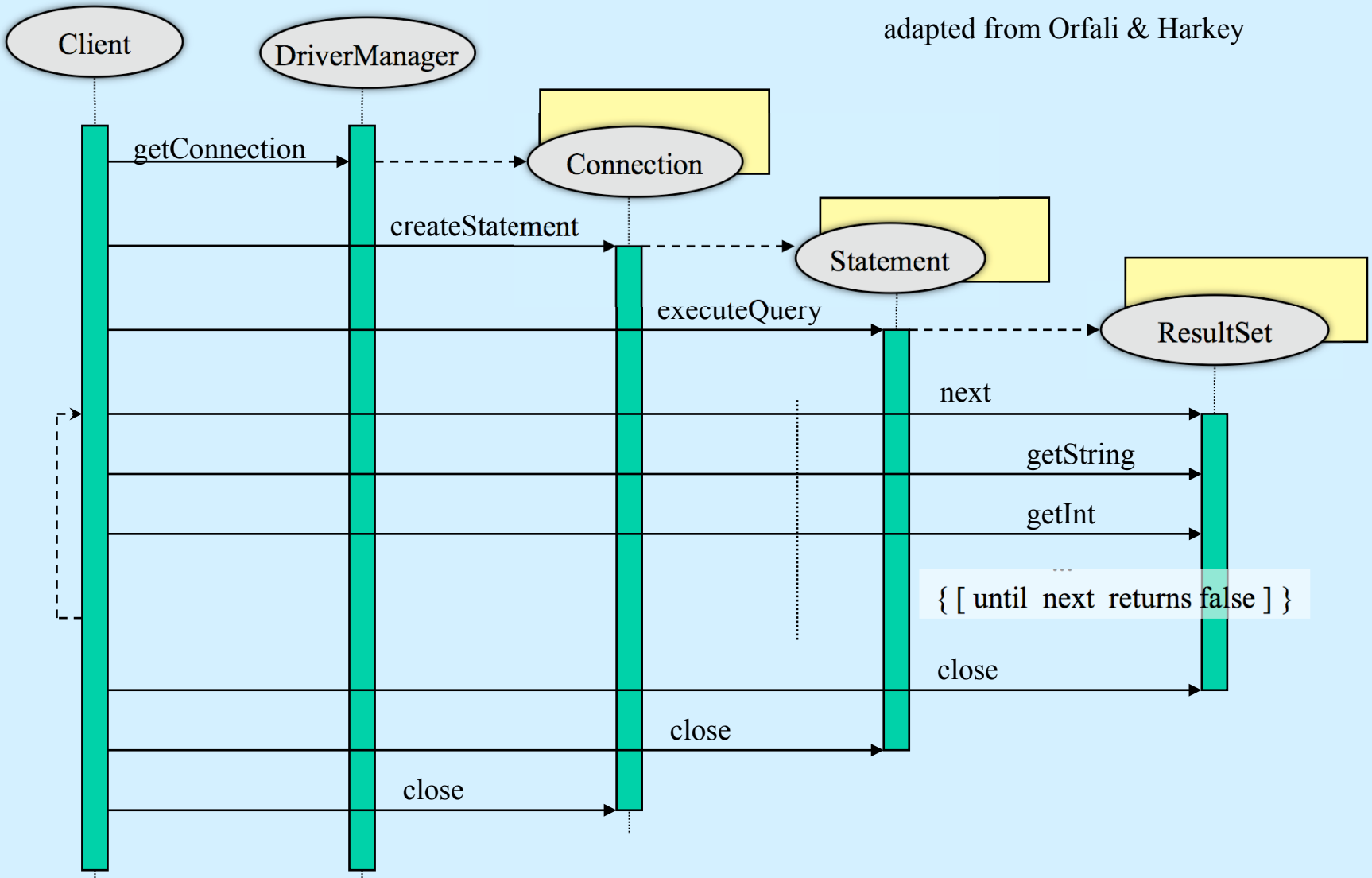
JDBC Escape Syntax

- SQL Transformations by the JDBC driver (like ODBC)
- Unifying the used SQL language

procedure calls	{call proc (arg1, …) }
?=call	{?= call proc (arg1, …) }
date	{d 'yyyy-mm-dd'}
escape char	{escape '%'}
functions	{fn function (arg1, …) }
outer joins	{oj outer-join }
time	{t 'hh:mm:ss'}
timestamps	{ts 'yyyy-mm-dd hh:mm:ss.fffff '}

JDBC: SQLQuery Sequence Diagram

adapted from Orfali & Harkey



A Minimalistic JDBC Demo

A small experiment in DebianDB using JdbcDemo.java demonstrating

- try-catch exception handling
- opening database connection
- transaction and rollback
- Statement object and a DDL command
- PreparedStatement and binding of parameter values
- SELECT and [cursor] processing of the Resultset

Afterwards you may experiment with it using other DBMS products in DebianDB virtual laboratory

JdbcDemo.java

Questions ?

3. Explaining the BankTransfer program

- Explaining details in the BankTransfer program (Appendix 2 in the Student Guide)
  - ReTry wrapper of a transaction
  - Starting a JDBC transaction
  - Synchronizing for concurrency conflict cases
  - Catching Concurrency Conflicts
  - Demo of a concurrency test

ReTry Wrapper for a Transaction

```
// Retry wrapper block of TransaferTransaction
    if (counter++ > 0) {                                    Avoiding "Livelocks"
      System.out.println("retry #" + counter);
      if (sqlServer) {
        conn.close();
        System.out.println("Connection closed");
        conn = java.sql.DriverManager.getConnection(URL,user,password);
        conn.setAutoCommit(true);
      }
    }
    TransferTransaction (conn,
                fromAcct, toAcct, amount,
                sqlServer, errMsg //,moreRetries
                );
    System.out.println("moreRetries="+moreRetries);     Just for testing
    if (moreRetries.equals("Y")) {
      long pause = (long) (Math.random () * 1000); // max 1 sec.
      System.out.println("Waiting for "+pause+ " mseconds"); // just for testing
      Thread.sleep(pause);
    }                                                       Avoiding "Livelocks"
  } while (moreRetries.equals("Y") && counter < 10);  // max 10 retries
  // end of the Retry wrapper block
```

Giving time for the other transaction to commit before the next retry

An Example JDBC Transaction

```
static void TransferTransaction (Connection conn,
        int fromAcct, int toAcct, int amount,
        boolean sqlServer,
        String errMsg //, String moreRetries
        )
        throws Exception {
        String SQLState = "******";
        try {
             conn.setAutoCommit(false);   // transaction begins
             conn.setTransactionIsolation(
                  Connection.TRANSACTION_SERIALIZABLE);
             errMsg = "";
             moreRetries = "N";

             // a parameterized SQL command
             PreparedStatement pstmt1 = conn.prepareStatement(
                "UPDATE Accounts SET balance = balance + ? WHERE acctID = ?");
             // setting the parameter values
             pstmt1.setInt(1, -amount);  // how much money to withdraw
             pstmt1.setInt(2, fromAcct); // from which account
             int count1 = pstmt1.executeUpdate();
             if (count1 != 1) throw new Exception ("Account "+fromAcct + " is missing!");
```

Explicit start of transaction

Declaring transaction isolation level
(even if it only affects on read operations!)

Parameterized SQL statement

Synchronizing for Concurrency Conflict cases

Programmed ”breakpoint” in the application code just for synchronizing the concurrency conflict cases

```
// --- interactive pause for concurrency testing -----
// In the following we arrange the transaction to wait
// until the user presses ENTER key so that another client
// can proceed with a conflicting transaction.
// This is just for concurrency testing, so don't apply this
// user interaction in real applications!!!
   System.out.print("\nPress ENTER to continue ...");
   BufferedReader reader = new BufferedReader(
            new InputStreamReader(System.in));
   String s = reader.readLine();
// --- end of waiting ----------------------------------
```

Catching Concurrency Conflicts

```
catch (SQLException ex) {
   try {
      errMsg = "\nSQLException:";
      while (ex != null) {
           SQLState = ex.getSQLState();
           // is it a concurrency conflict?
           if ((SQLState.equals("40001")   // Solid, DB2, SQL Server,...
             || SQLState.equals("61000")   // Oracle ORA-00060: deadlock detected
             || SQLState.equals("72000"))) // Oracle ORA-08177: can't serialize access
                moreRetries = "Y";
           errMsg = errMsg + "SQLState: "+ SQLState;
           errMsg = errMsg + ", Message:  "+ ex.getMessage();
           errMsg = errMsg + ", Vendor:   "+ ex.getErrorCode() + "\n";
           ex = ex.getNextException();
      }
      // SQL Server does not allow rollback after deadlock !
      if (sqlServer == false) {
         conn.rollback();  // explicit rollback needed for Oracle
                           // and the extra rollback does not harm DB2
      }
      // println for testing purposes
      System.out.println("** Database error: " + errMsg);
   }
   catch (Exception e) { // In case of possible problems in SQLException handling
      System.out.println("SQLException handling error: " + e);
      conn.rollback(); // Current transaction is rolled back
      ;  // This is reserved for potential exception handling
   }
} // SQLException
```

Adapting to different behaviors of DBMS products

For re-trying the transaction

Adapting to different behaviors of DBMS products

BankTransfer Example

Listing 2-3 Scripts for experimenting with BankTransfer using MySQL on Linux platform

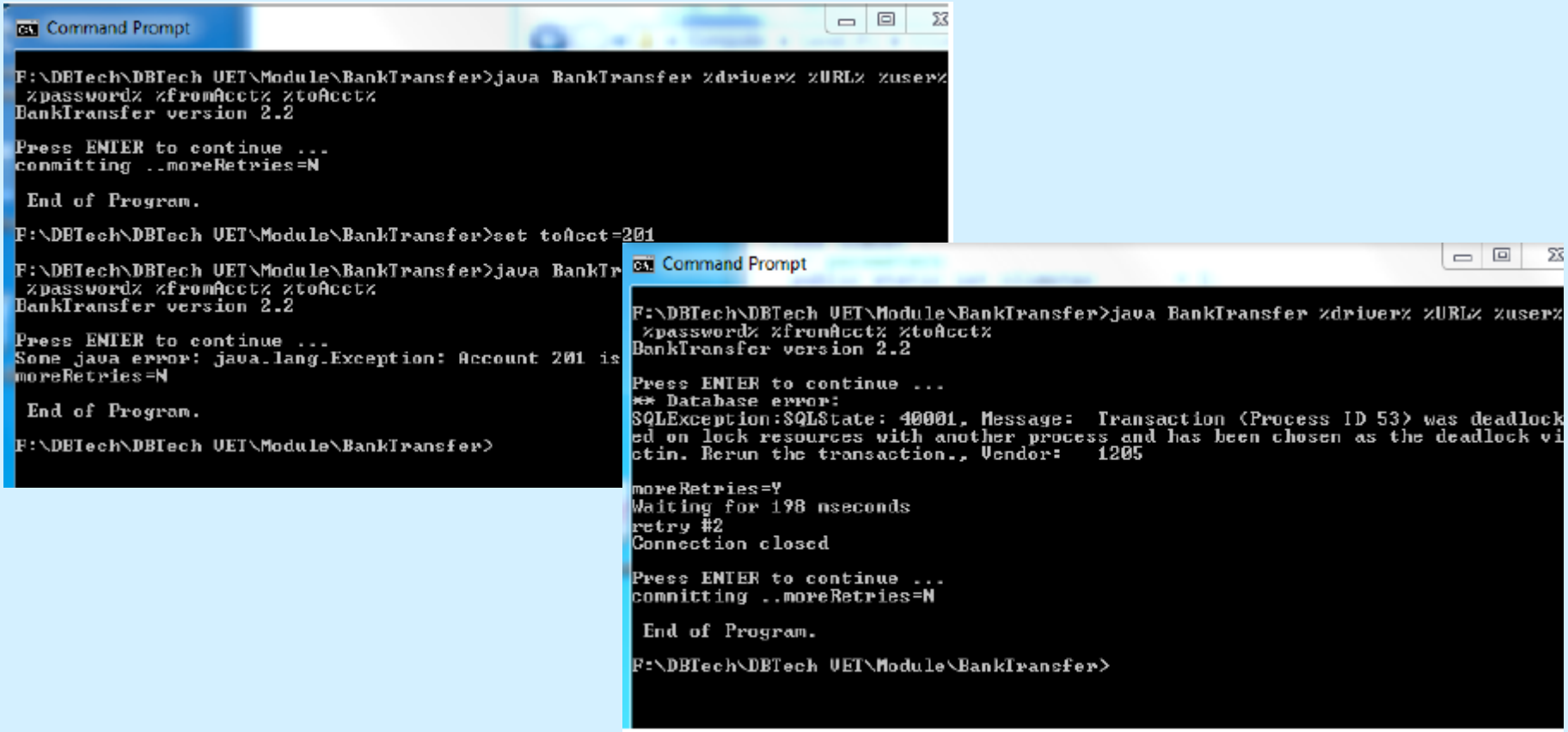
```
Scripts for MySQL on Linux
# creating directory /opt/jdbc-drivers for JDBC drivers
cd /opt
mkdir jdbc-drivers
chmod +r+r+r jdbc-drivers
# copying the MySQL jdbc driver to /opt/jdb-drivers
cd /opt/jdbc-drivers
cp $HOME/mysql-connector-java-5.1.23-bin.jar
# allow read access to the driver to everyone
chmod +r+r+r mysql-connector-java-5.1.23-bin.jar

#******* MySQL/InnoDB ***************************************************
# First window:
export CLASSPATH=/opt/jdbc-drivers/mysql-connector-java-5.1.23-bin.jar
export driver=com.mysql.jdbc.Driver
export URL=jdbc:mysql://localhost/testdb
export user=user1
export password=sql
export fromAcct=101
export toAcct=202
java -classpath .:$CLASSPATH BankTransfer $driver $URL $user $password $fromAcct $toAcct

# Second window:
export CLASSPATH=/opt/jdbc-drivers/mysql-connector-java-5.1.23-bin.jar
export driver=com.mysql.jdbc.Driver
export URL=jdbc:mysql://localhost/testdb
export user=user1
export password=sql
export fromAcct=202
export toAcct=101
java -classpath .:$CLASSPATH BankTransfer $driver $URL $user $password $fromAcct $toAcct
#*************************************************************************************
```

BankTransfer demo?

(The following experiment was run on Windows)



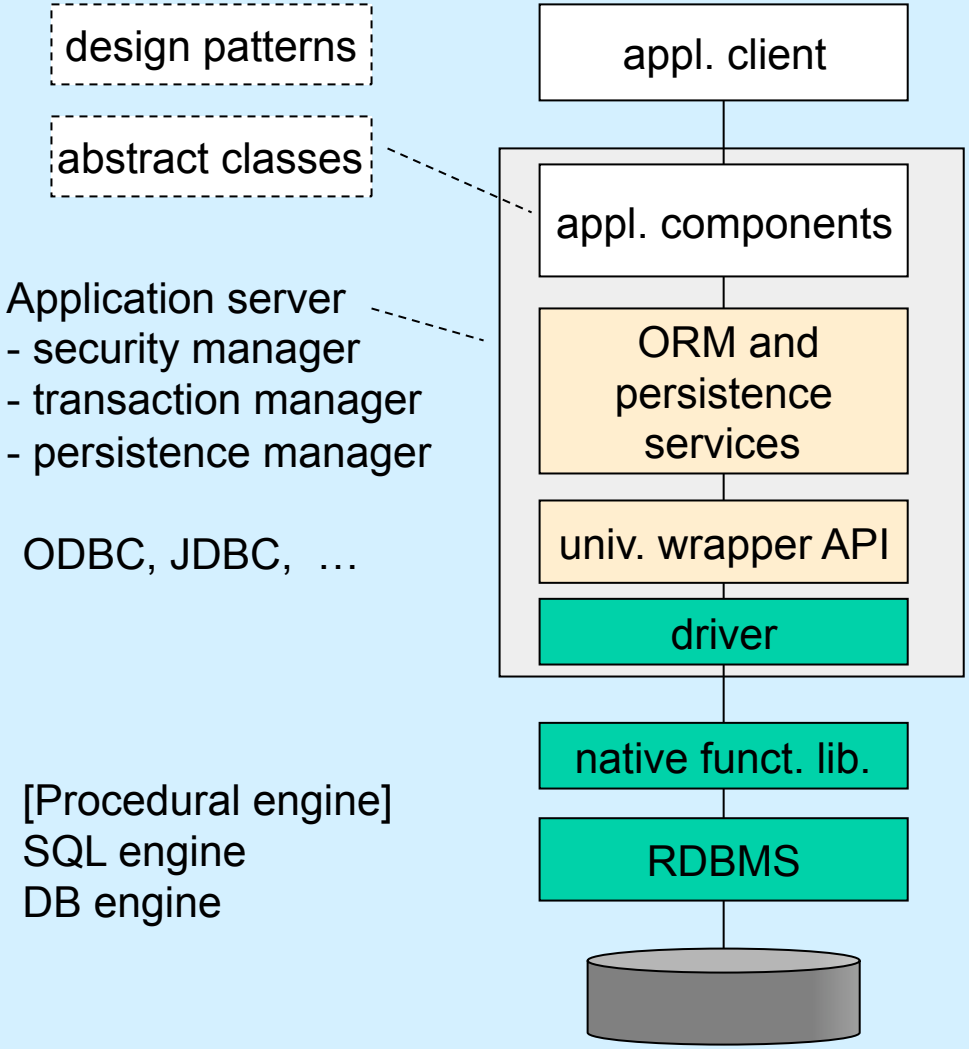
Questions ?

4. More Advanced Topics ..

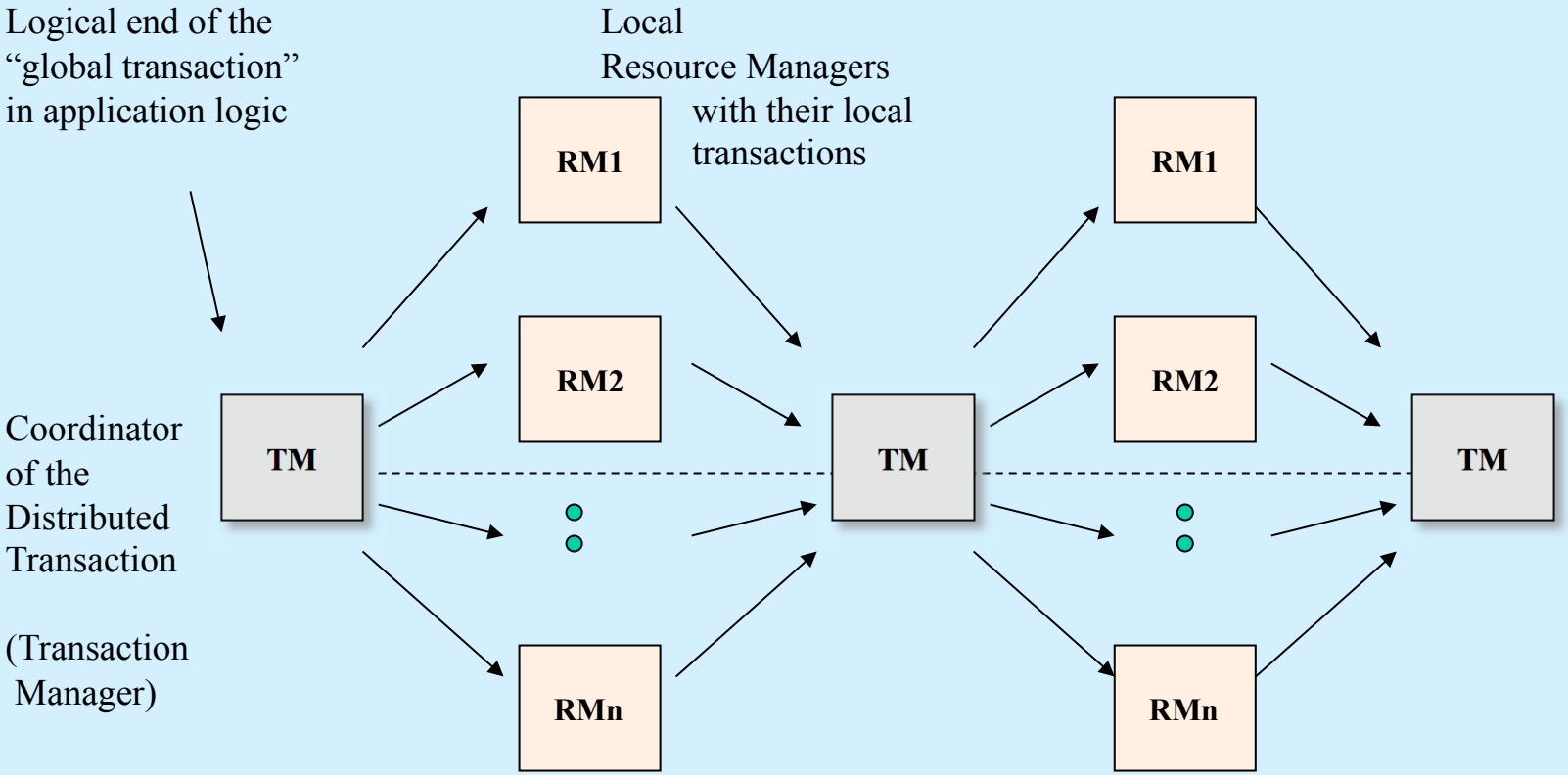
- More advanced JDBC, stored programs, ..
- SQLJ, JDO, ..
- Architectures ..
- Distributed transactions ..
- Object-relational mappings (ORM), JPA
- …

A Big Picture of Data Access Layers

in n-Tier architecture



Distributed Transaction
Two-Phase Commit (2PC) Protocol

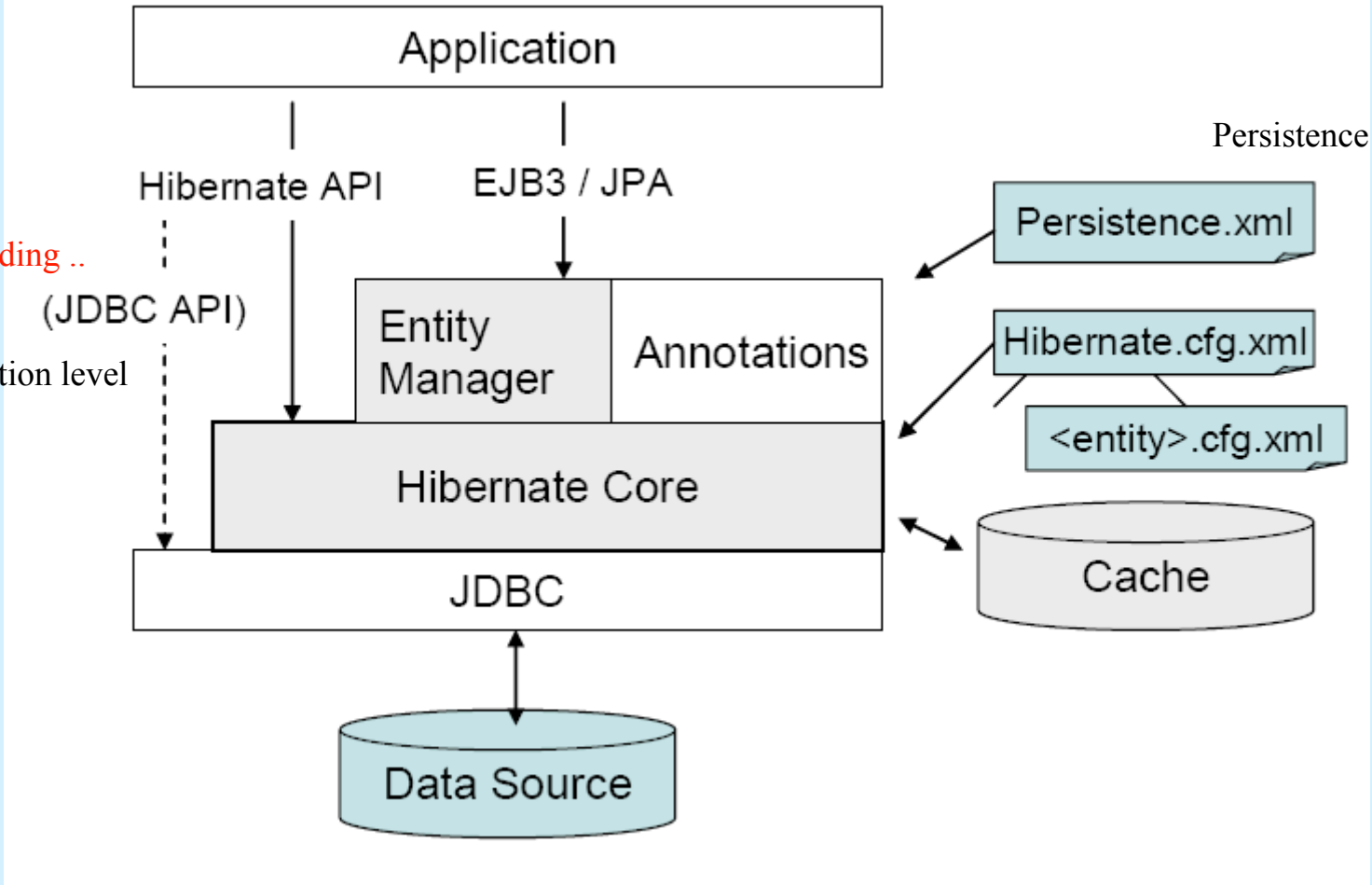


Prepared to Commit ?

All Yes → Commit !

One No → Abort !

Even the Hibernate Stack needs JDBC



Questions ..

For more information on JDBC see

•Melton J and Eisenberg A, 2000, ”Understanding SQL and Java Together”, Morgan Kaufmann Publishers
•Oracle, 2011, Java Developer’s Guide, 11*g* Release 2
•IBM, 2012, Developing Java Applications
•MySQL, connector-j.pdf in mysql-connector-java-5.1.23.jar/docs
•PostgreSQL, 2011, PostgreSQL JDBC Driver