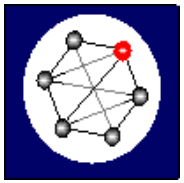


Lifelong Learning Programme

www.DBTechNet.org

Data Access Patterns

in Data Access Engineering

With focus on the mainstream DBMS systems

used by the European Industry

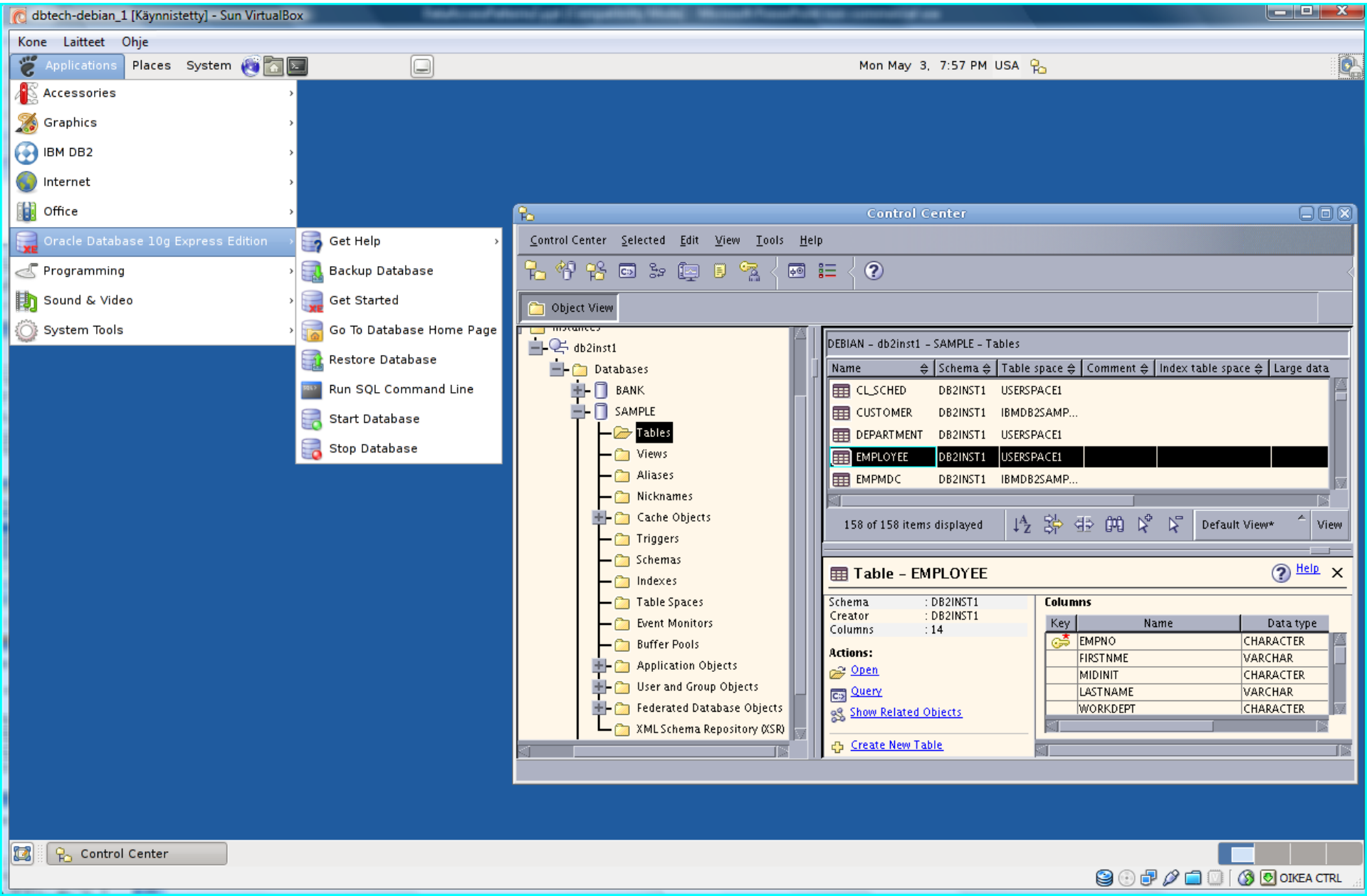
presentation on 2010-05-05 (updated 05-06)
at DBTech EXT workshop in Malaga University

Disclaimer
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.

Contents

- [Introduction of the Virtual Laboratory, DB2, Oracle, ...]
- Data Access as part of Software Engineering
- Patterns in general, focusing on Data Access Patterns
- Data Access APIs
  - SQL, ESQL, ODBC, JDBC, ADO.NET, ...
  - Cursor Processing
  - ORM
- Reliability and Integrity
  - SQL Transactions and Concurrency Control
  - Retryer Pattern
  - User Transactions and RVV
- Performance and Scalability
  - Connection Pooling and Statement Pooling
  - Execution Plan Management
- ORM
  - JDO, JPA, LINQ

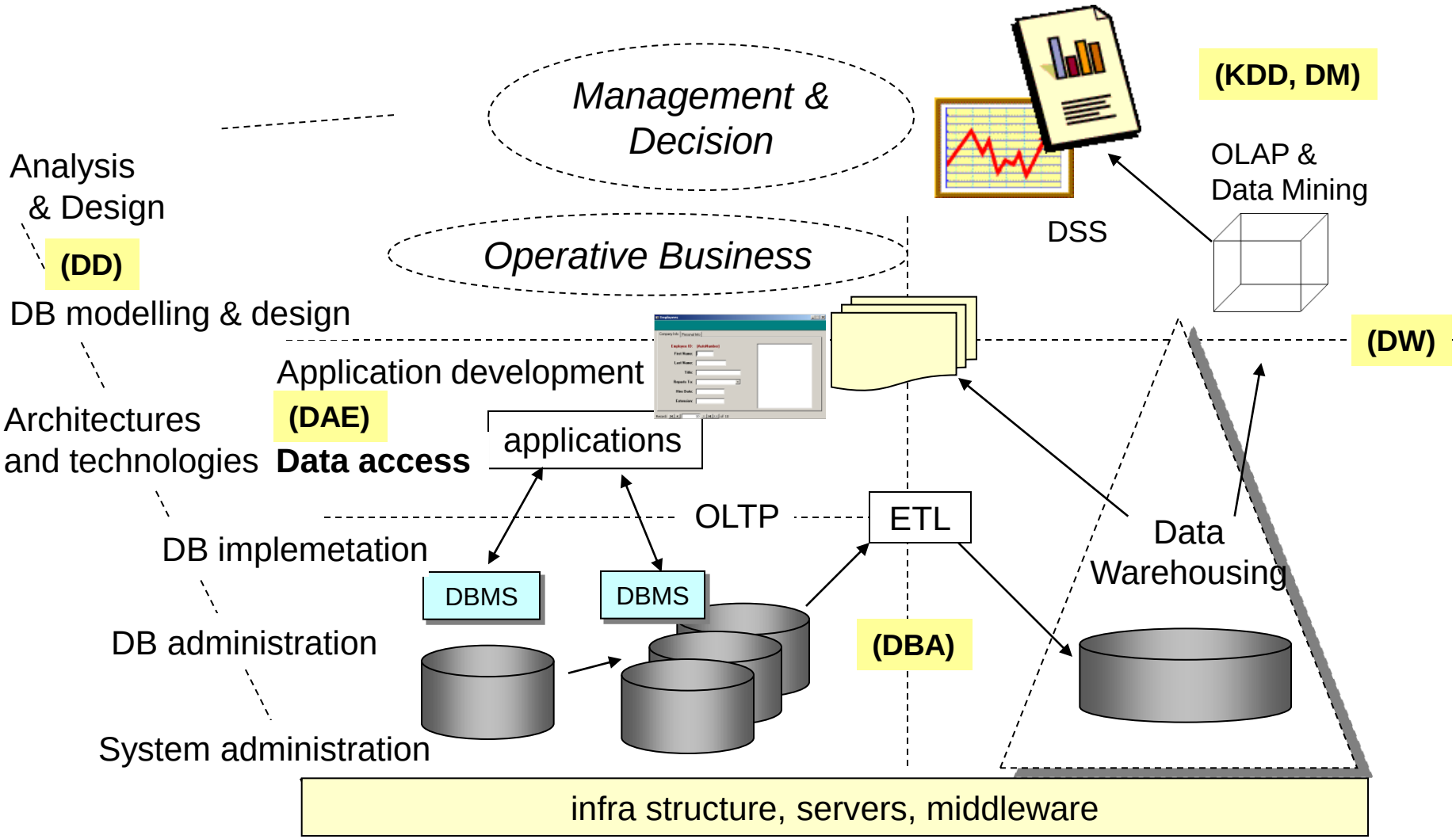
VirtualBox dbtech-debian



Data Access Technologies

- Data Access
  - Technologies, models and methods of applications accessing database services
- Challenges:
  - Continuous change, rapidly changing software architectures, models, methods, and infrastructures
  - Database technologies considered rather matured, but expanding
    - ISO SQL standard and different implementations
  - Rapid hardware development
    - Processing power, Multiprocessor systems, Capacities, Network
  - Concurrency management
  - Growing impact of Internet
  - 24x7 High Availability (HA)

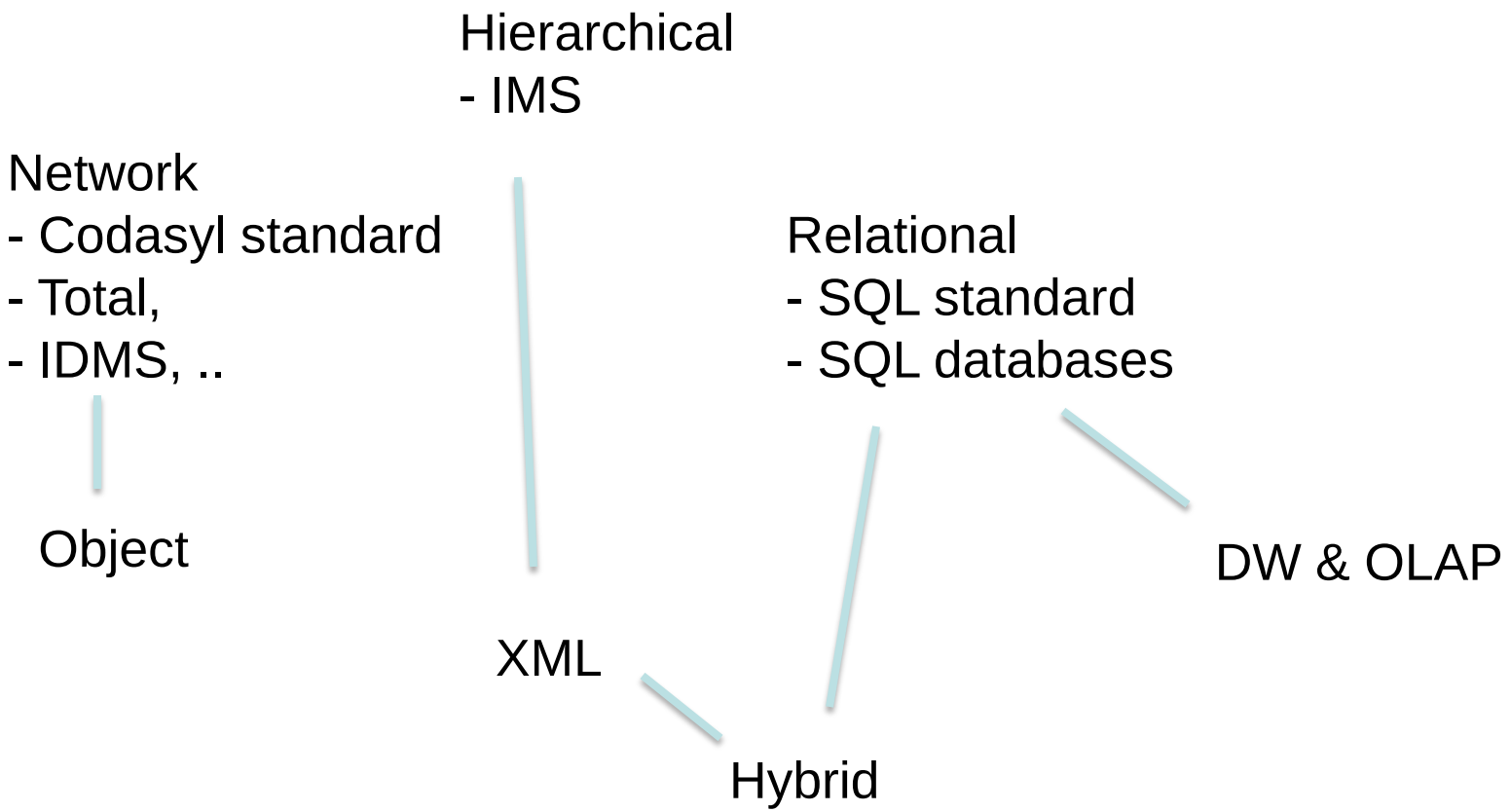
Roles and Knowledge Areas



What do we mean with Patterns?

- Application Architectures, Frameworks
  - JavaEE, .NET, SOA ?
- Methodologies, Mainstream Programming Paradigms
  - Procedural, Functional
  - Object-Oriented
- Programming languages, *models*, ...
  - Model-View-Control (MVC) model, …
- Best practices
  - Design Patterns of GoF
    - collection of generic classes found in applications
  - Data Access Objects (DAO), J2EE/JavaEE Patterns
- On generic use of Data Models and APIs
  - Hierarchical, Network, Relational, DW, Object, XML, Hybrid

Data Models


Hierarchical
- IMS
Network
- Codasyl standard
- Total,
- IDMS, ..
Relational
- SQL standard
- SQL databases
Object
DW & OLAP
XML
Hybrid

Pattern Objectives

- **Reliability** of the software
  - Nr 1 priority in OLTP, we cannot accept less than 100% reliability in a database
  - Integrity constraints, Transaction programming and services, Security
  - Exception programming
- Productivity of the work
  - For cost effetiveness
- **Performance**, **scalability**, and predictability
  - For costs and user satisfaction
- Maintainability of the software
  - For reliability and productivity
- Portability
  - For future
  - based on de facto standards

Data Access APIs

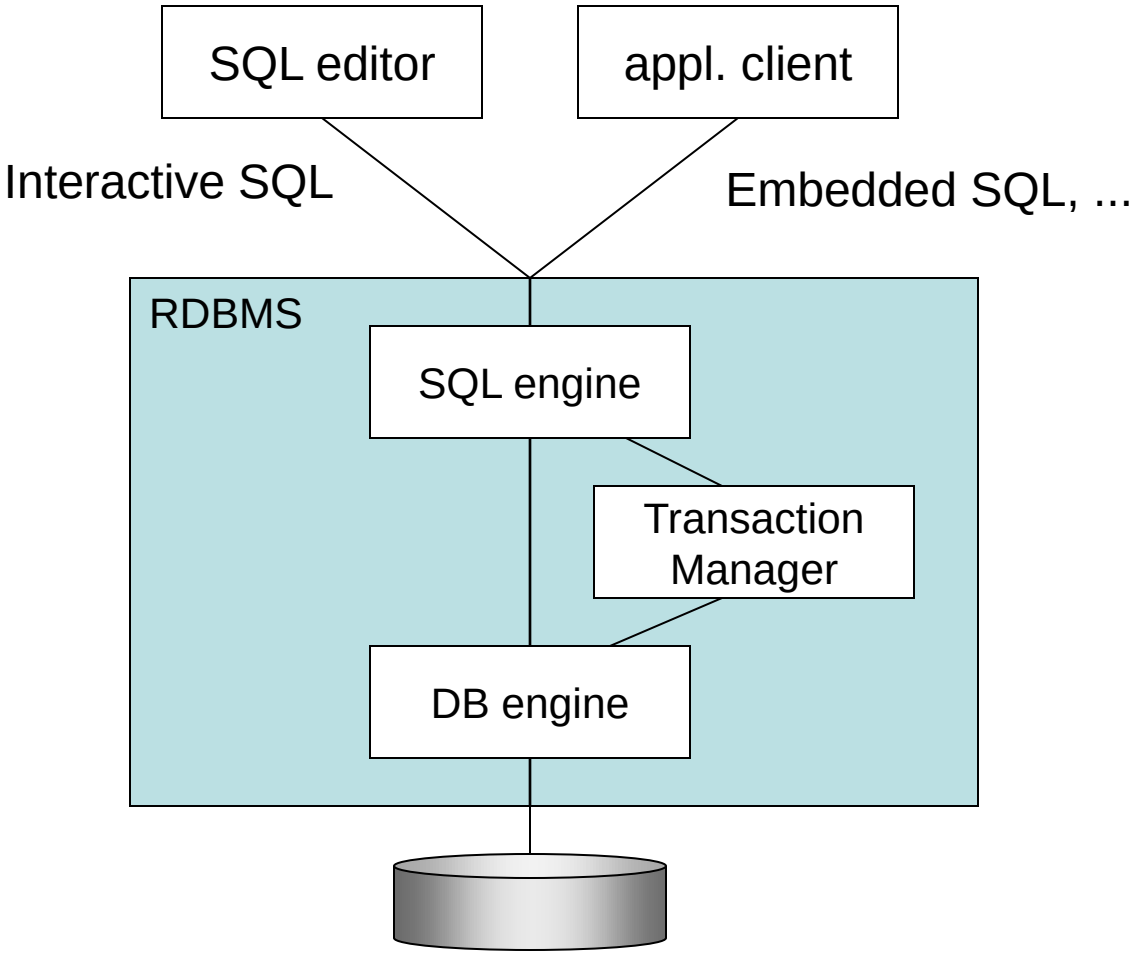
- The basic "abstract API" for RDBMS is ***SQL***

- Native function libraries of the RDBMSs
- Embedded SQL (ESQL)
  - The first generic API
- Call Level Interface (CLI)
  - Universal function libraries
  - ODBC, OLE DB, (JDBC driver type 4)
- Generic CLI wrapper libraries
  - ADO, ADO.NET, JDBC, SQLJ, JDO?
- Object/Relational Mapping (ORM) middlewares
  - JDO, EJB3, Hibernate, JPA, LINQ, ..

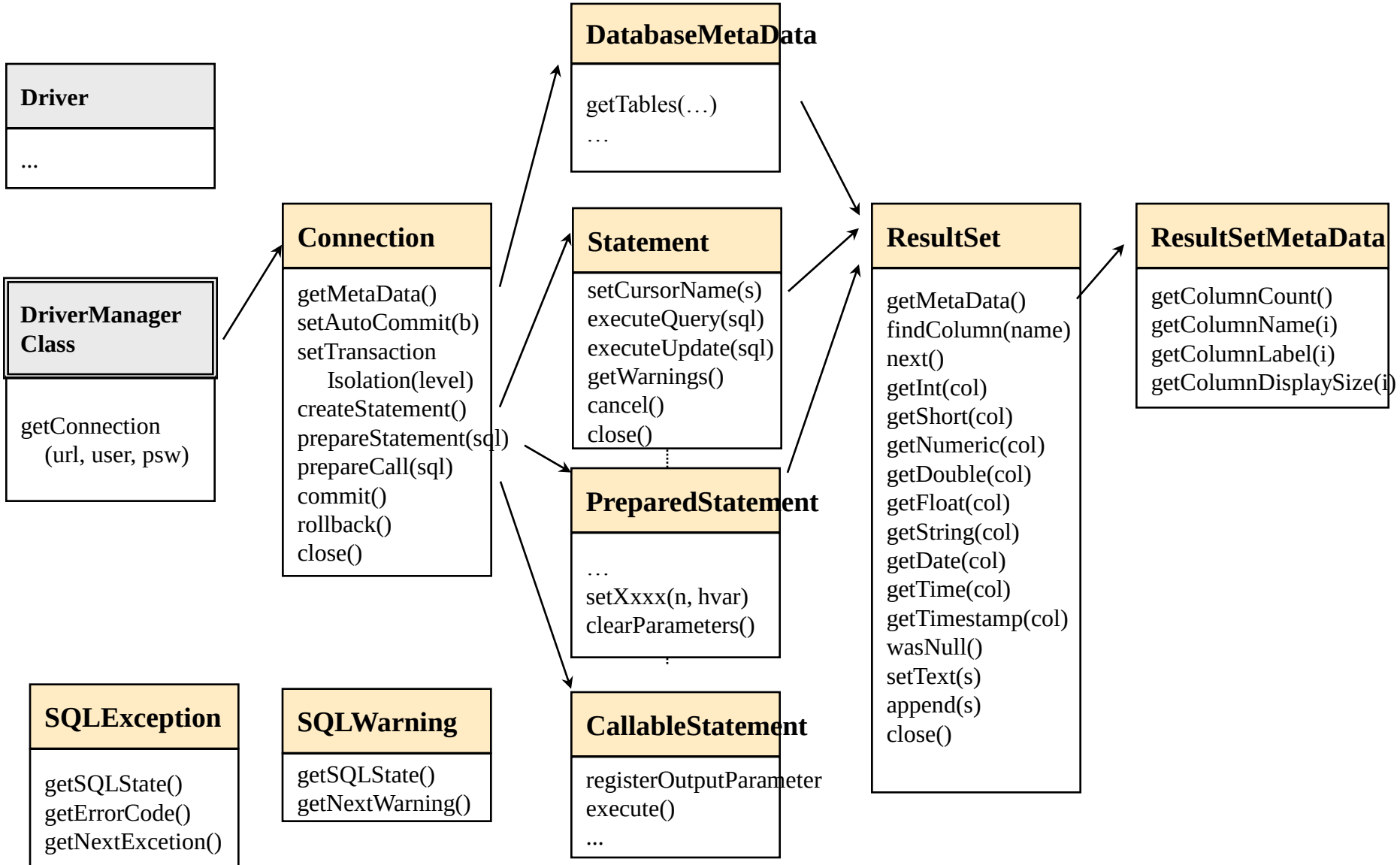
The abstract SQL API

and the basic Client/Server architecture

abstract layers of a RDBMS

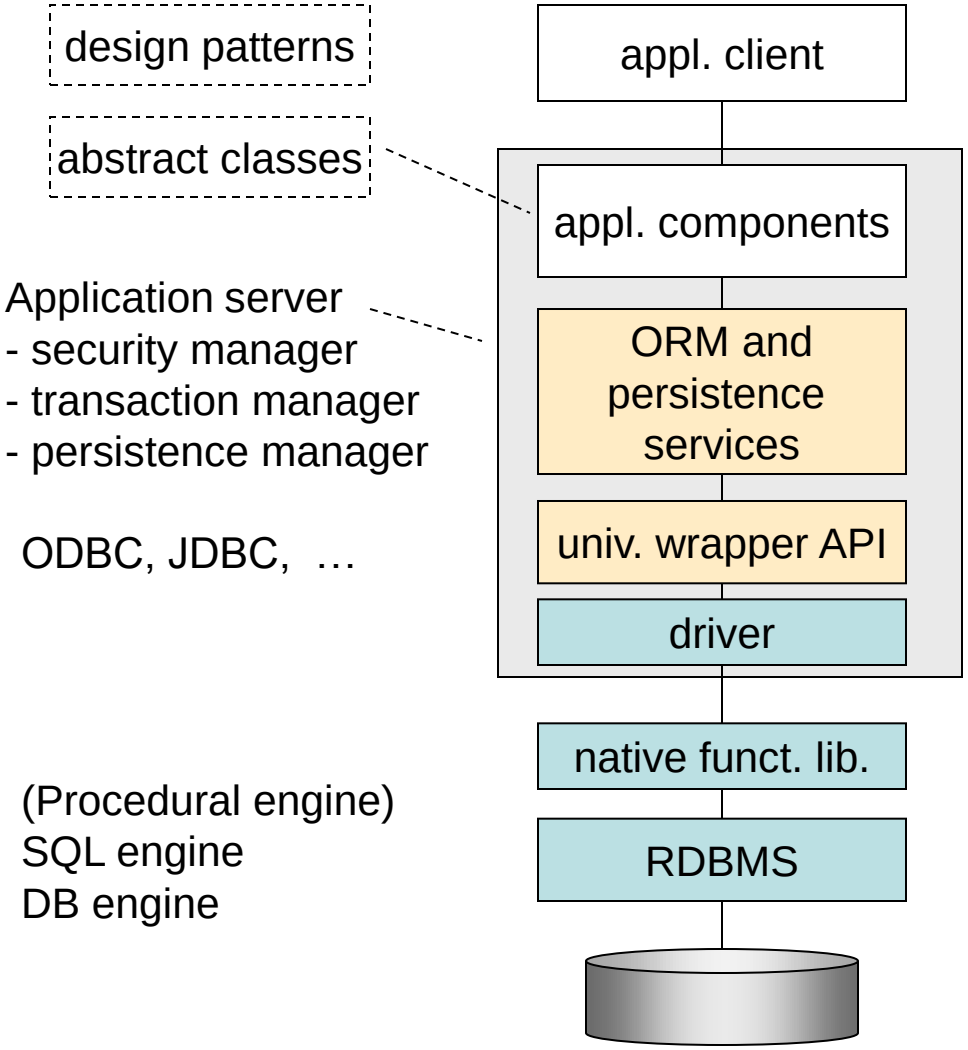


JDBC 1 - Interfaces, Methods, Objects



Data Access Layers

in n-Tier architecture



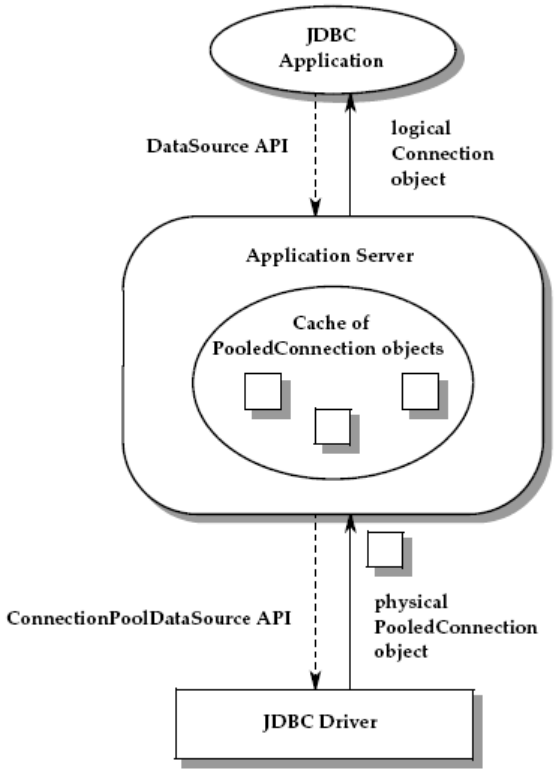
Middleware

JDBC specification:
"The driver layer may mask differences between standard SQL:2003 syntax and the native dialect supported by the data source"
- ?

Data Access Patterns

- Interfaces, class libraries (wrappers)
- Connection Management
  - Connection pooling
- Cursor models (result set processing)
- Procedural extensions
  - stored procedures
  - User Defined Functions (UDF)
- Execution Plan Management
  - Statement, PreparedStatement, Statement Pooling, Stored Procedures
- SQL transaction models
  - Local: Flat, Savepoints, Nested, ..
  - Concurrency control and Isolation levels
  - Retryer pattern
- User transaction models
  - RVV, ”long transactions”, Global/Distributed

Connection Pooling



Source: JDBC 4 Specification

A connection pool per connection string (URL, uid, psw)

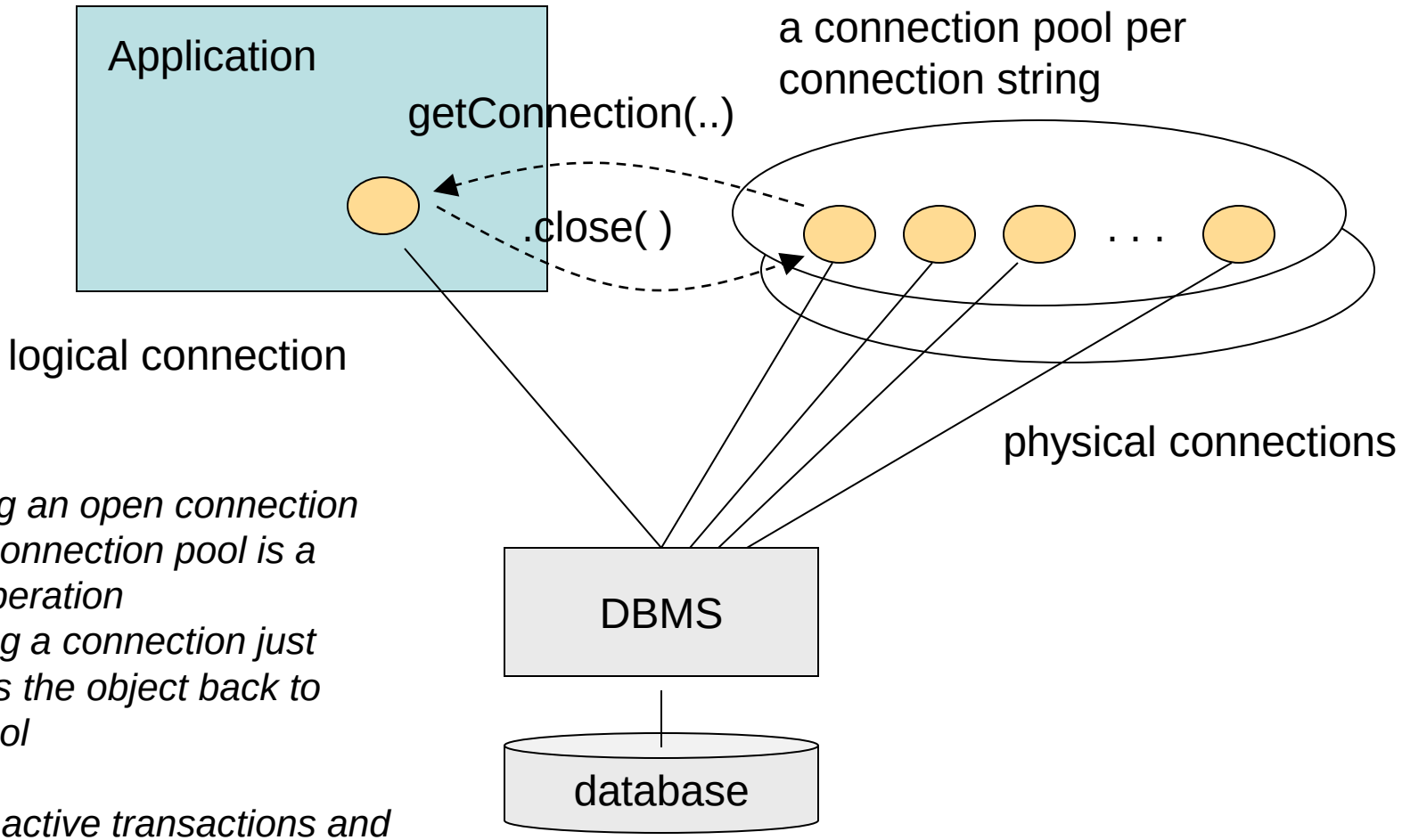
Reason:
creating a new connection is slow operation - a performance problem

Operations:
First getConnection creates <minIdle> +1 open connections in the pool and takes 1 to be used by the application.

Closing the connection returns the active connection back to the pool, unless there would be more than <maxIdle> connections in the pool, in which case the connection is closed.

If all connection objects have been idle over <maxWait> time, then all connections are closed.

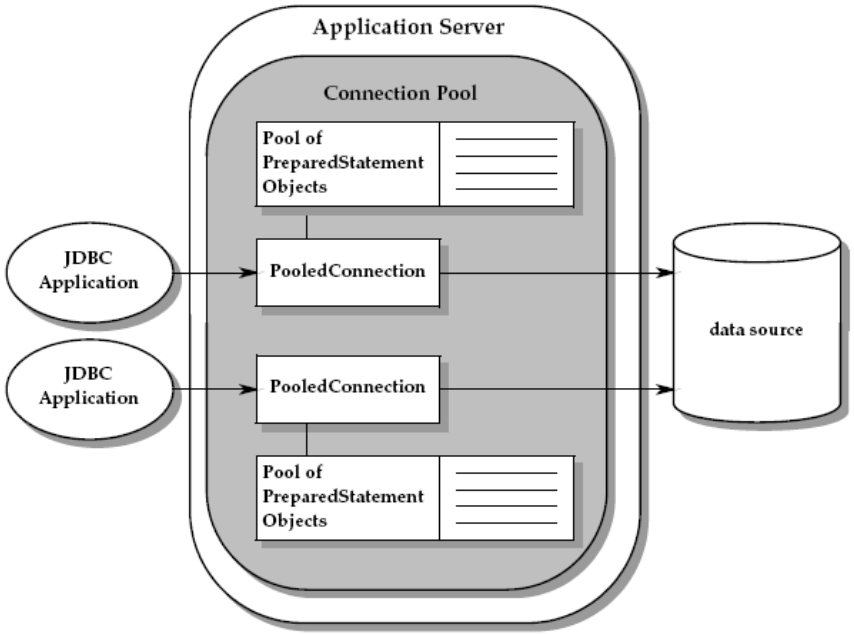
Connection Pool



Getting an open connection from connection pool is a fast operation
Closing a connection just returns the object back to the pool

Note: *active transactions and handles to exection plans belong to the single connection !*

Statement Pooling



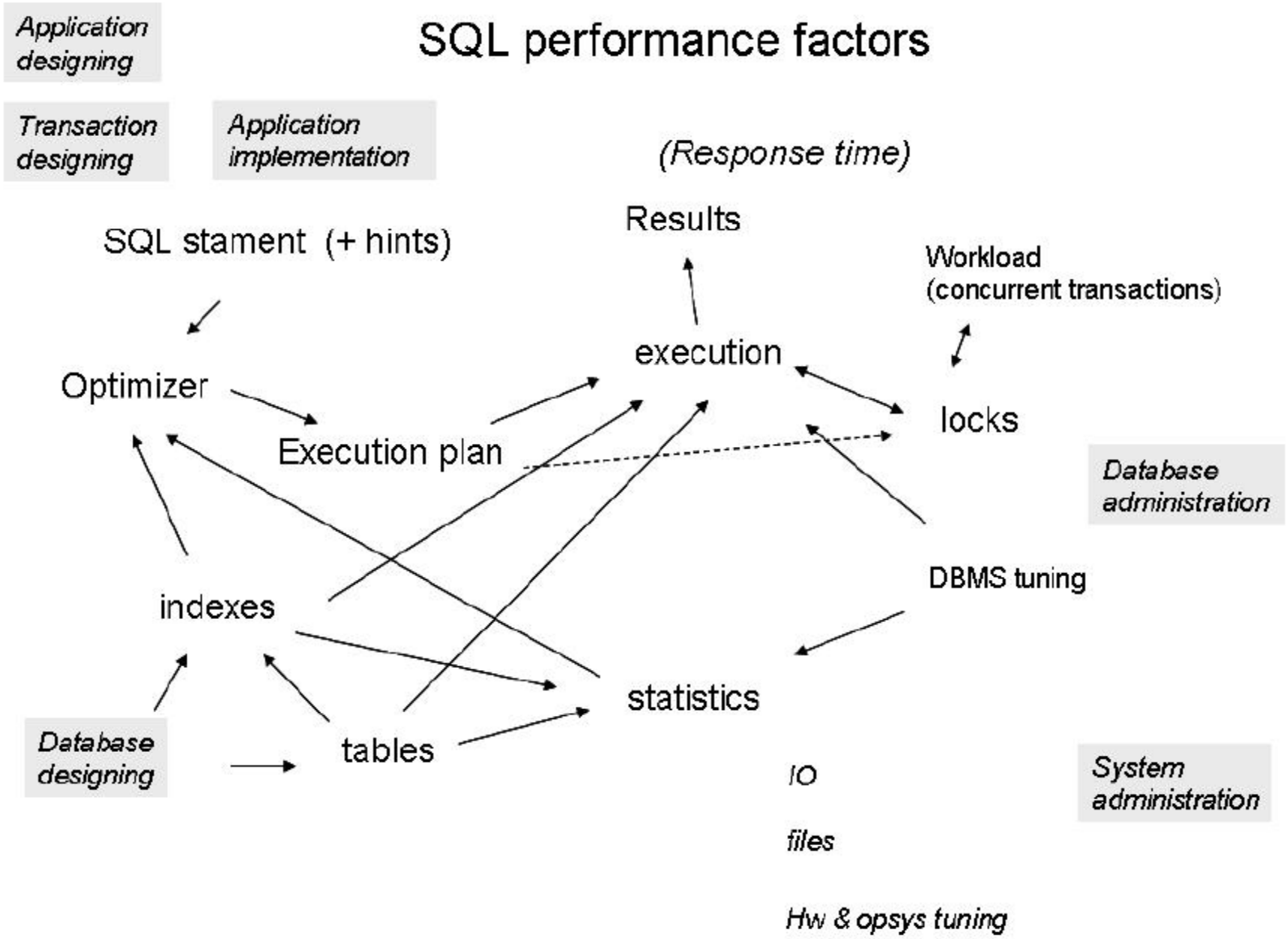
Source: JDBC 4 Specification

According to the specification, the statement pooling can be implemented in
- Application Server
- JDBC driver, or
- Data Source (database server)

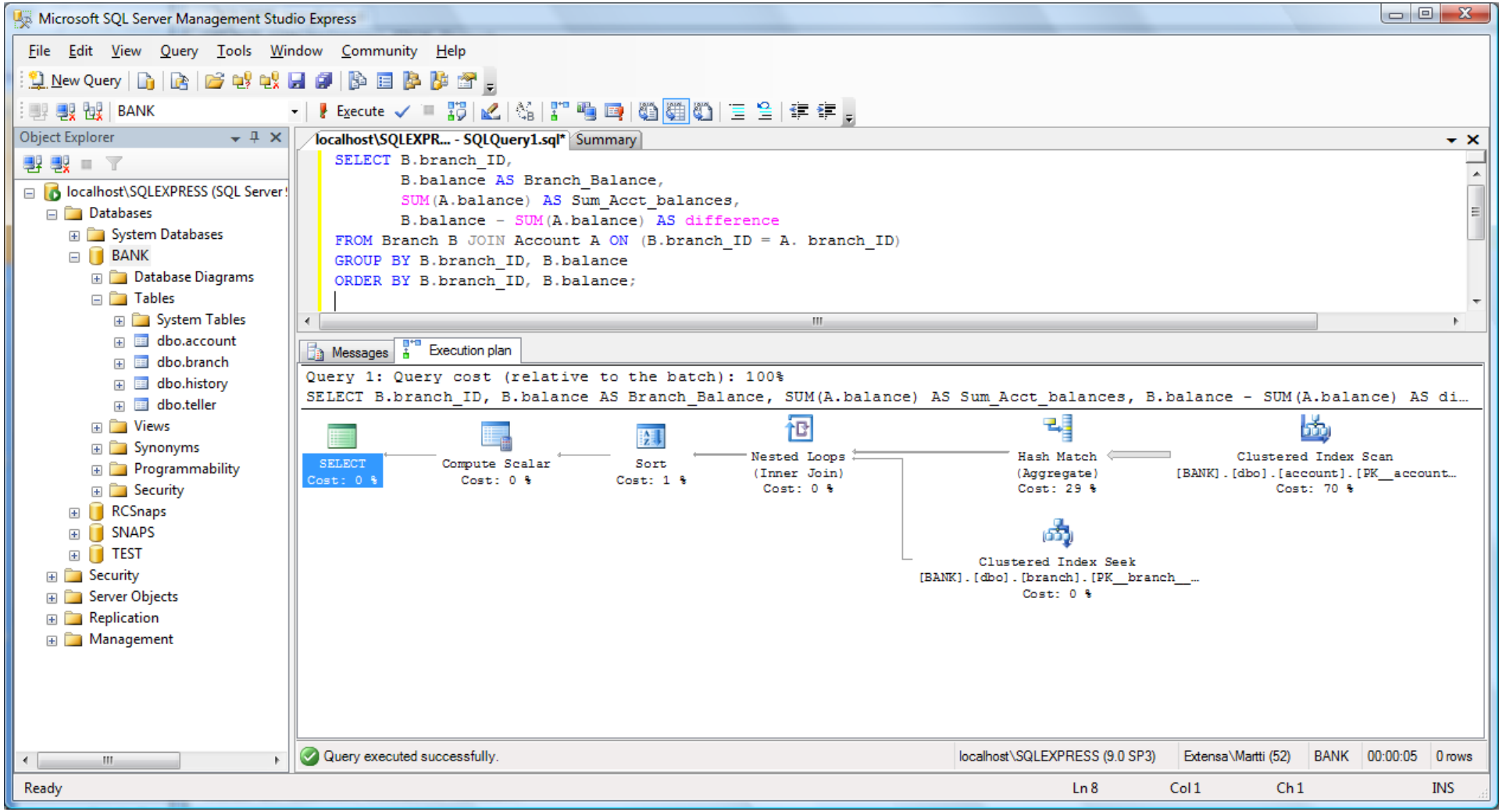
Statement pooling means that PreparedStatement objects live after closing, and a new prepareStatement just activates an idle PrepararedStement object and its execution plan in the server.

Use of statement pooling is transparent to the application. Application has no control on use of statement pooling.

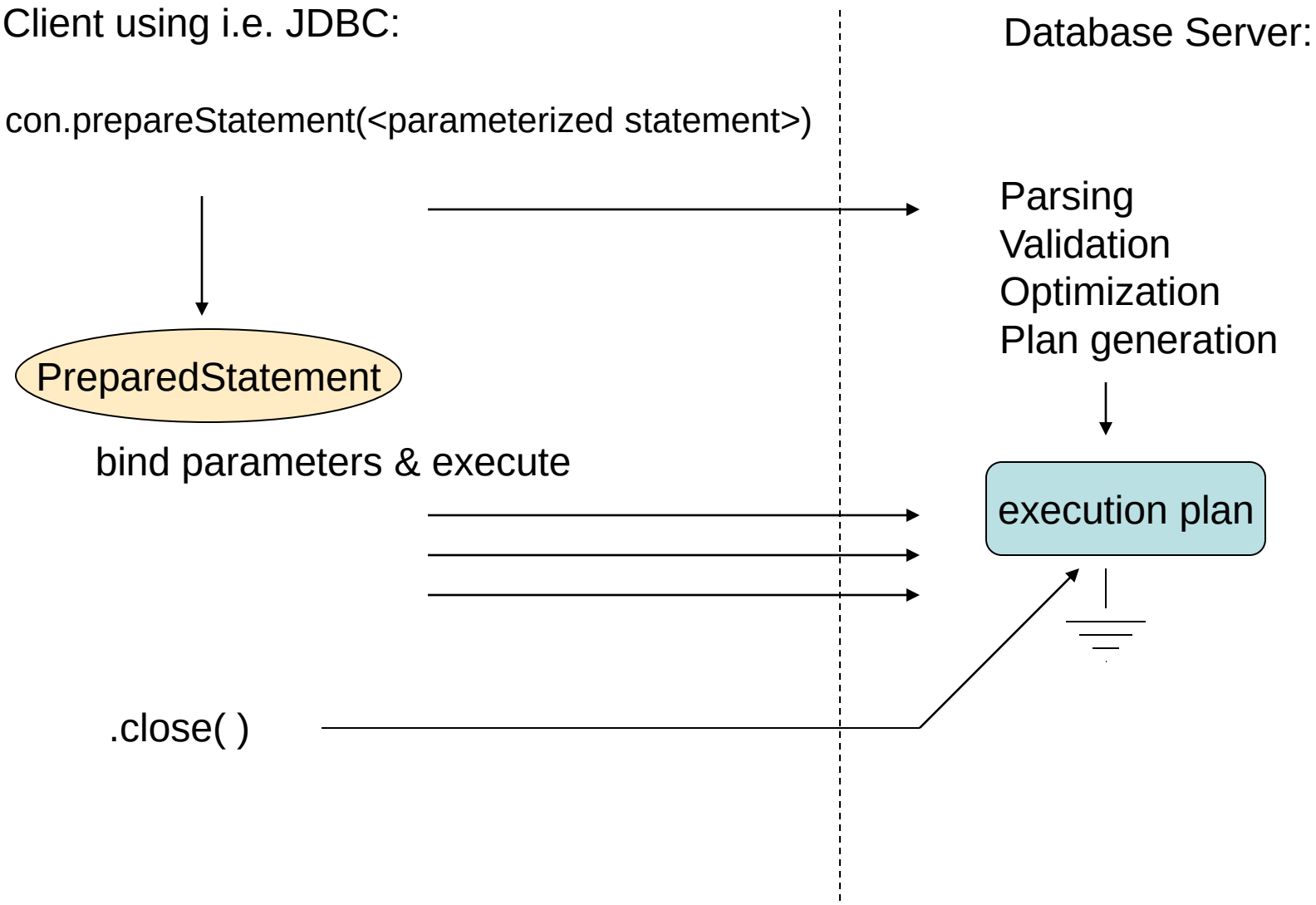
SQL performance factors



A Sample Execution Plan



Execution Plan Management

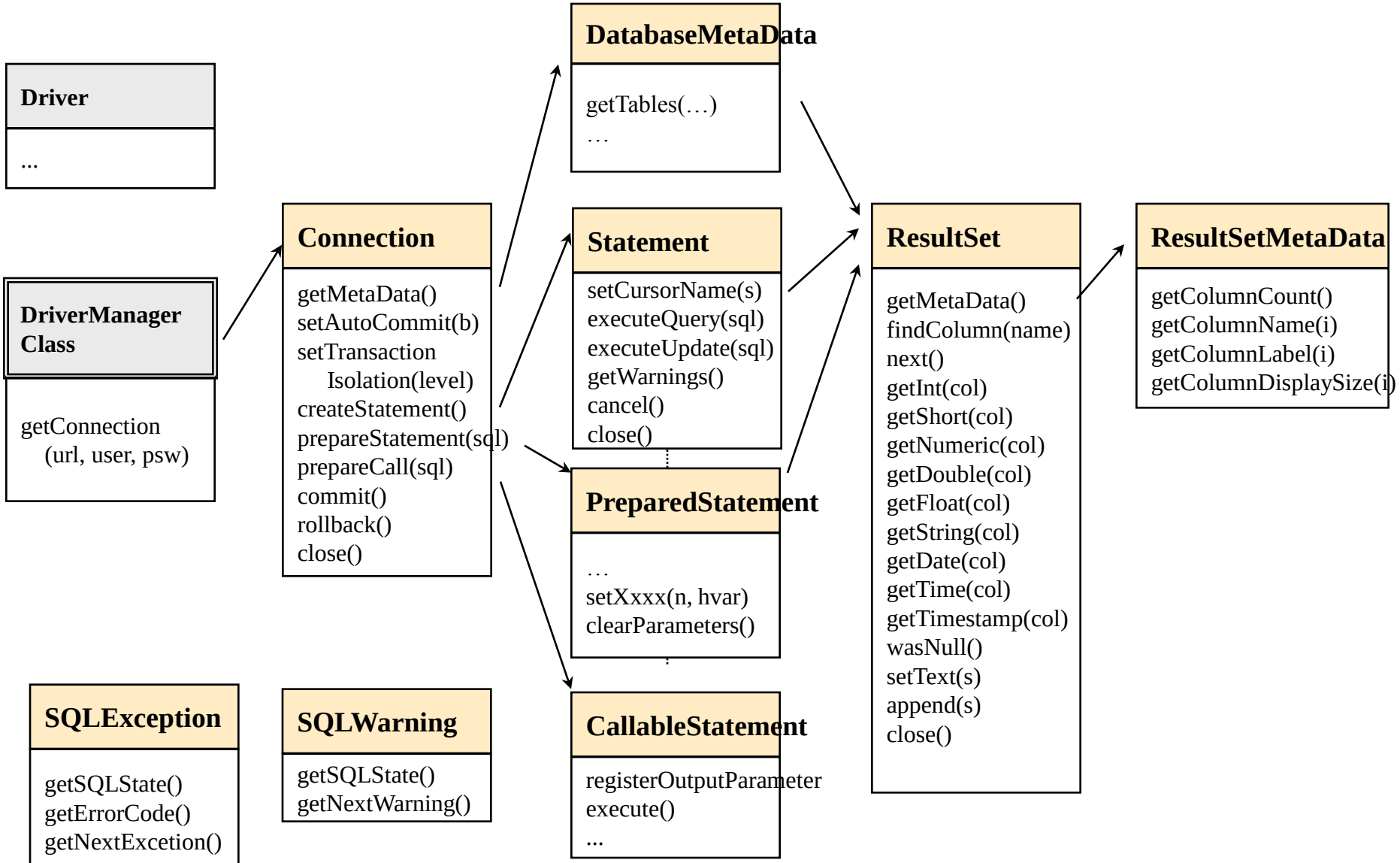


Cursor Processing

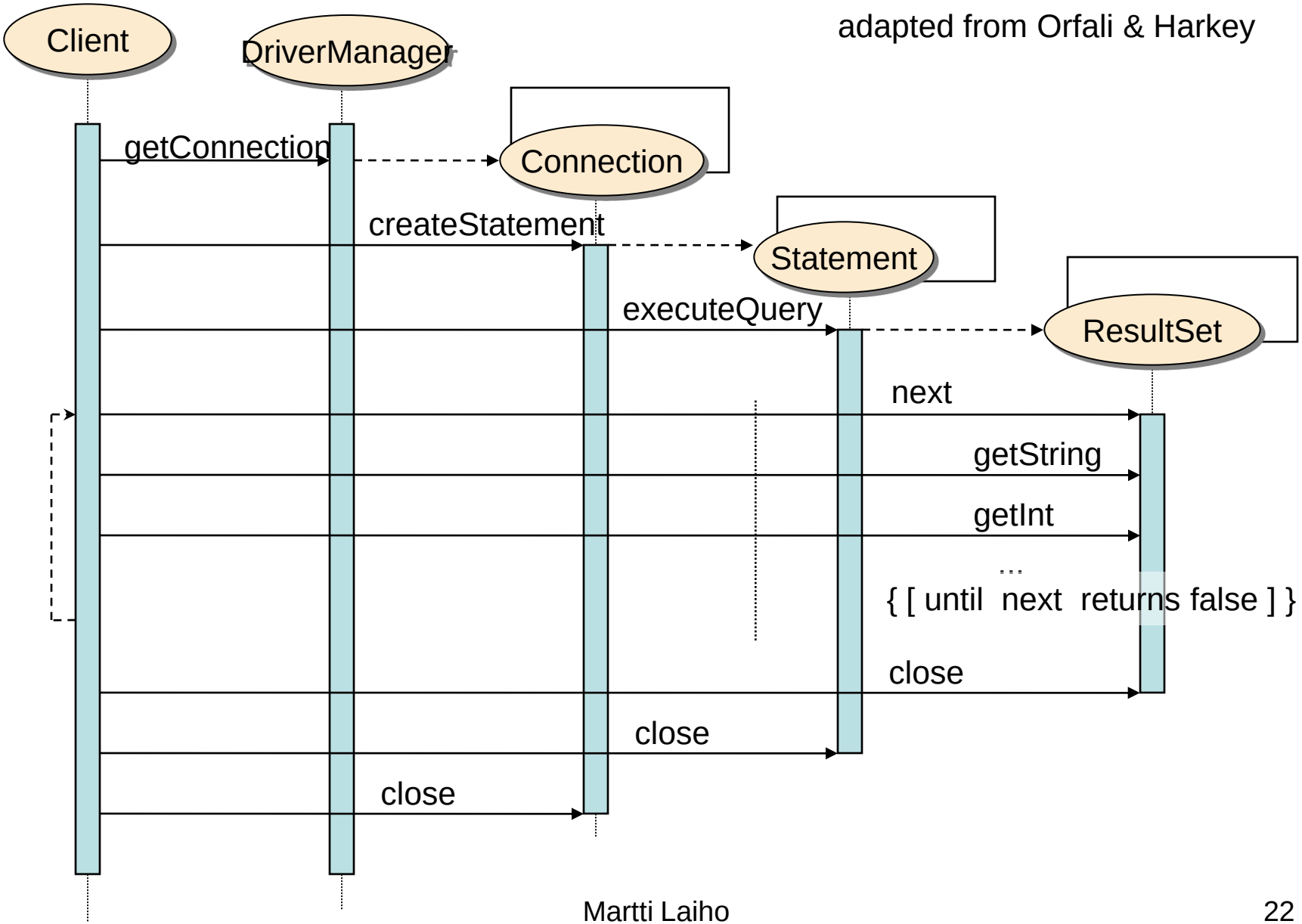
- Solves the paradigm mismatch between
  - Procedural Programming and
  - (”Relational”) SQL databases

- Scrolling / Forward only
- Sensitive / insensitive (snapshot)
- Server-side / client-side cache
- Optimistic concurrency
- Scope: transaction / (holdable) multiple transactions
- Options (hints)

JDBC 1 - Interfaces, Methods, Objects

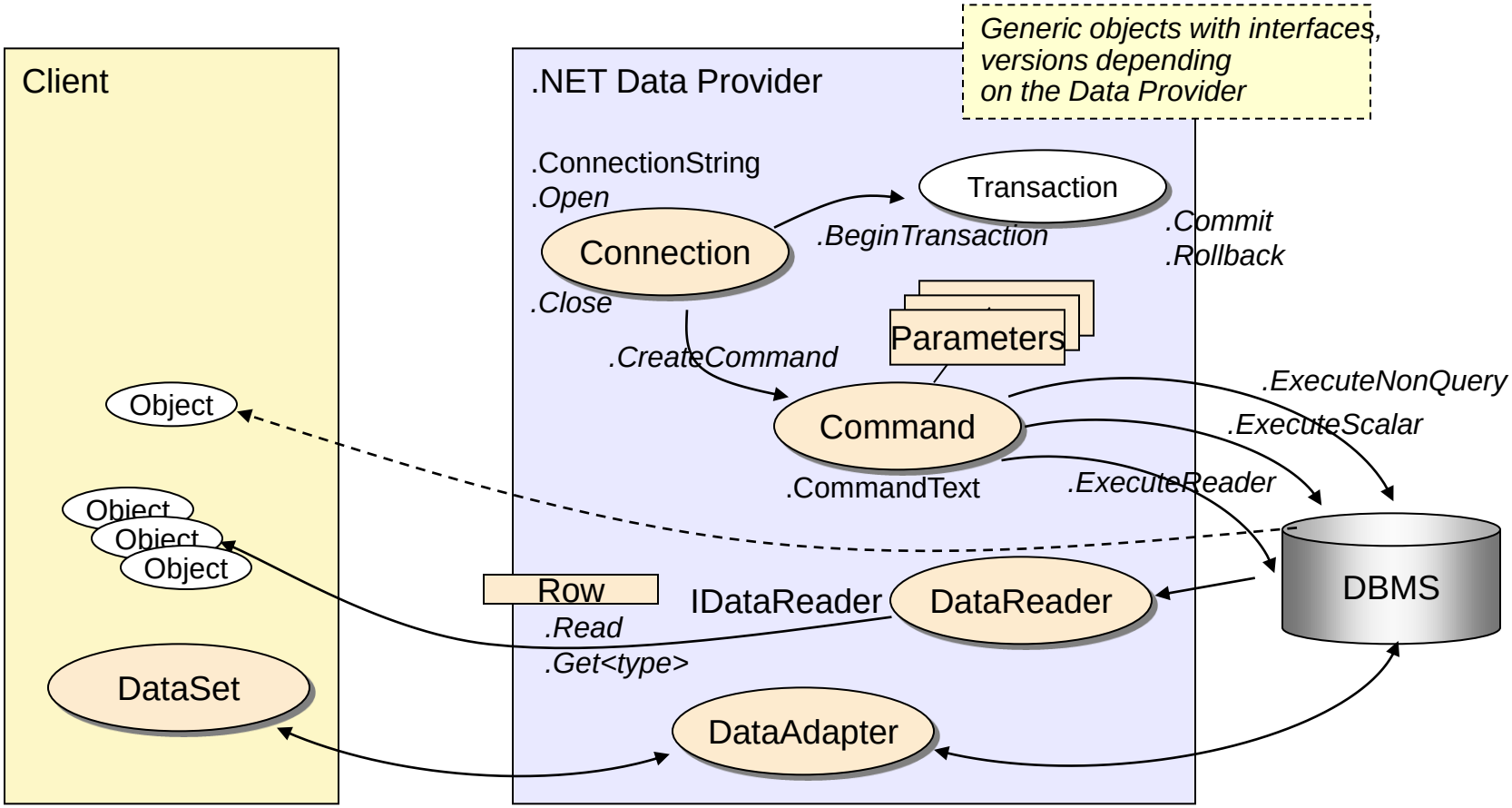


JDBC: SQLQuery Sequence Diagram



Source: Understanding .NET
David Chappell

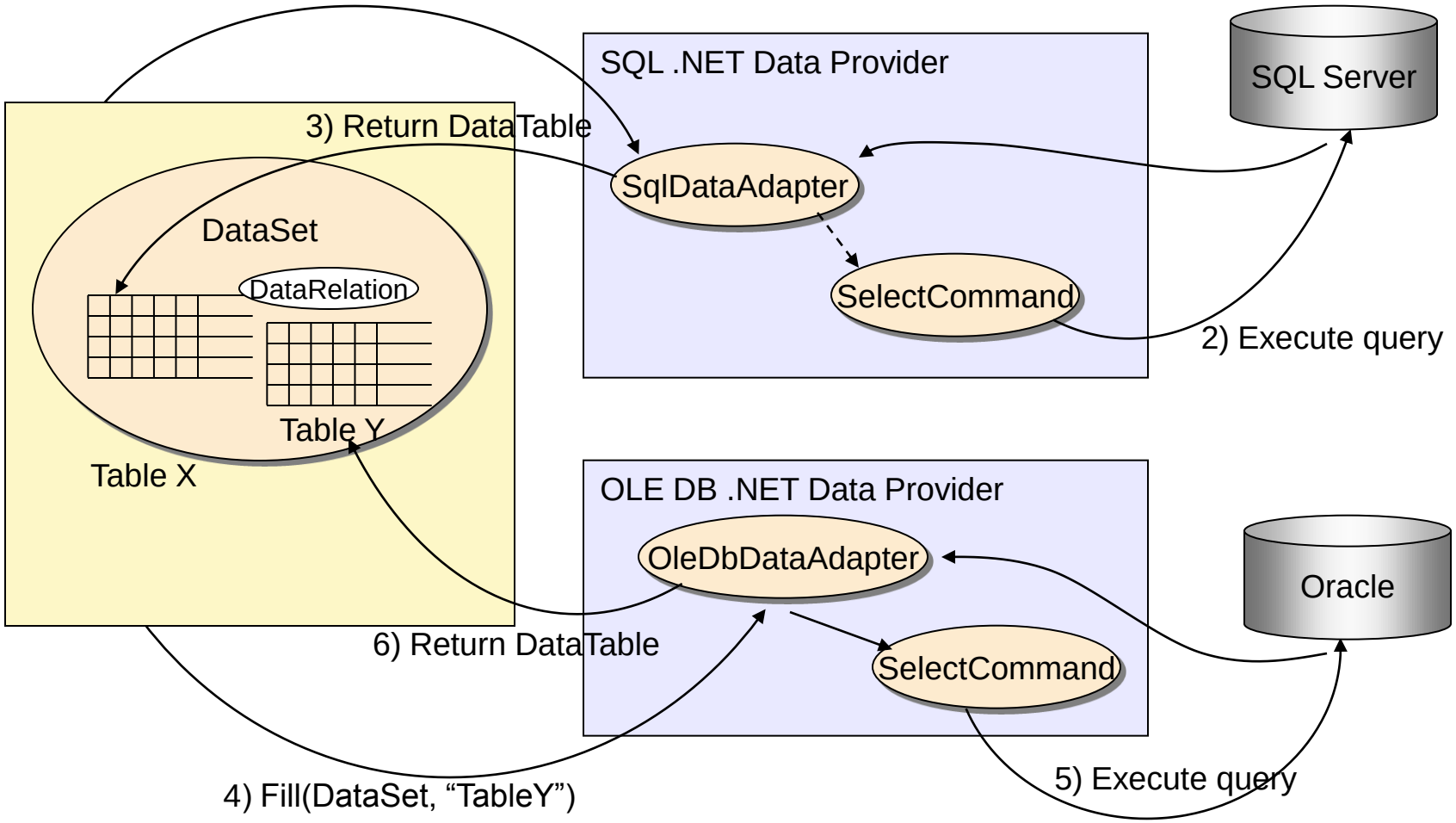
ADO.NET: Connected Mode



Data Providers implement object classes of their own:
- SqlClient: Sql<object type>
- OleDb: OleDb<object type>
- Odbc: Odbc<object type>
- OracleClient: Oracle<object type>

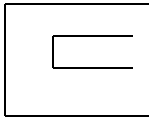
ADO.NET: Disconnected Mode

Source: Understanding .NET
David Chappell



SQL Transaction Models

- Flat transaction
  - ACID properties
    - Atomicity (all or nothing!)
    - Consistency (integrity constraints)
    - Isolation (based on LSCC, MVCC, or OCC concurrency control)
    - Durability (persistency)
  - Savepoints
    - Atomicity in parts
  - Isolation levels
    - AC(I-)D - compromizing for performance
    - Default for commands in the transaction
    - Can be defined differently for cursors and single commands
- Nested transactions



- Chained transactions

ACID SQL transaction

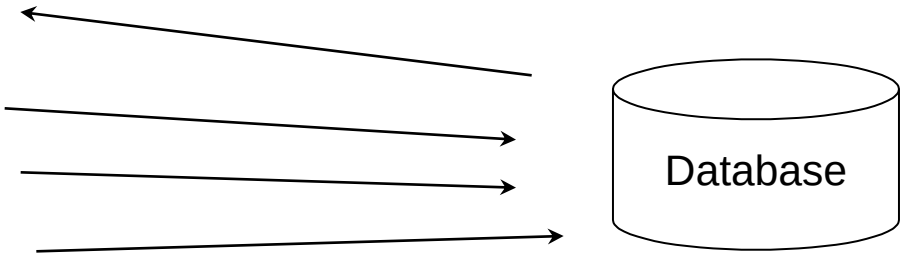
[{SET | START} TRANSACTION [READ ONLY | READ WRITE]
ISOLATION LEVEL {READ UNCOMMITTED |
READ COMMITTED |
REPEATABLE READ |
SERIALIZABLE }

Isolation

[if ..]]
SET {UNIQUE | REFERENCIAL} CONSTRAINTS
{DEFERRED | IMMEDIATE }

[LOCK TABLE …]
SELECT …
if ..
INSERT …
if ...
UPDATE …
if ...
DELETE …
if …
SAVEPOINT spn
…

Consistency
- by DBMS
- logical



Transaction log

Atomicity

Durability

COMMIT | **ROLLBACK**
if ...

Transaction Patterns

Dependency models of **Single-connection SQL transactions**

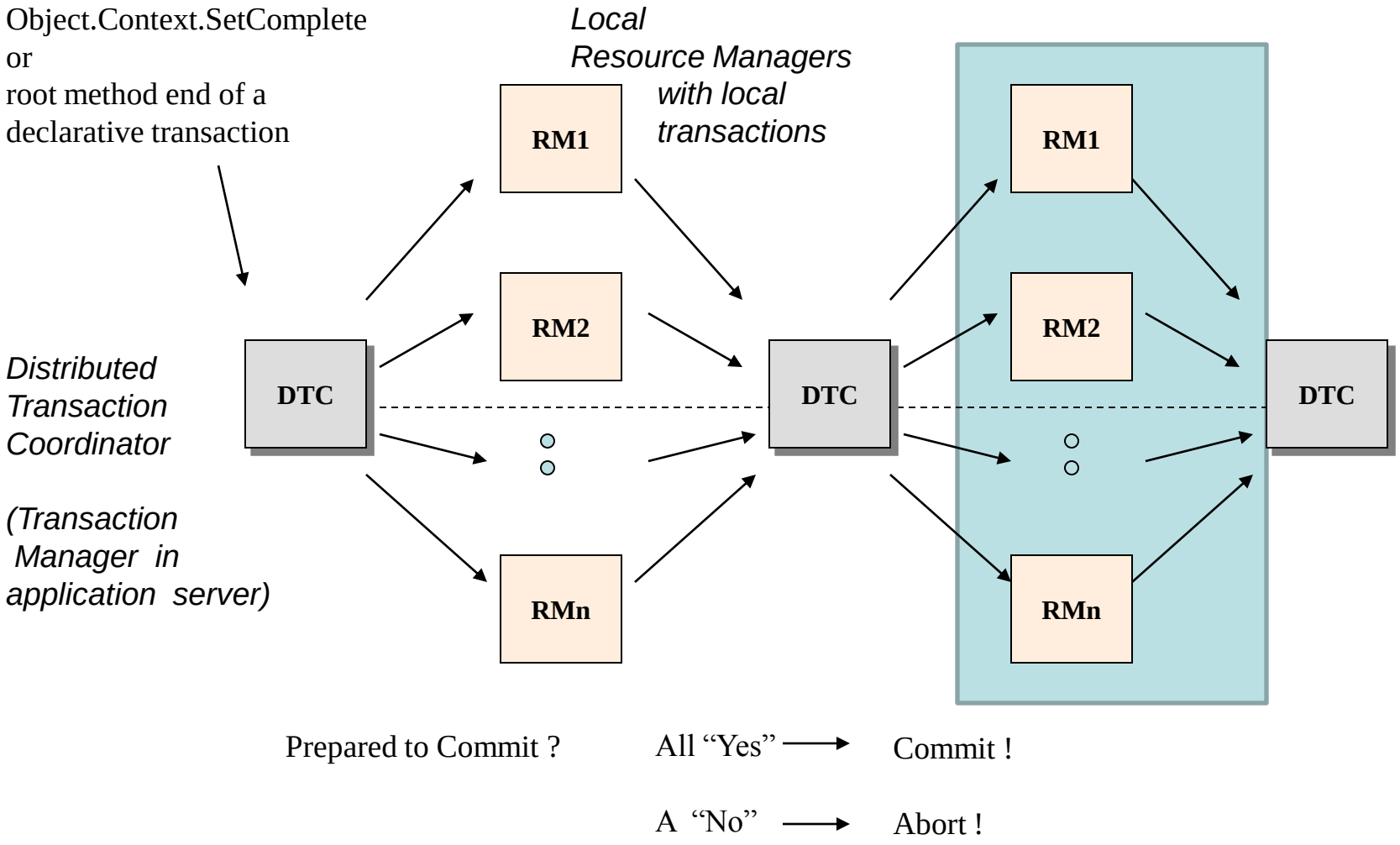
- Flat transactions
  - typical practice
- Nested transactions, Autonomous transactions
- Chained transactions (ISO SQL only?)

Dependency models of **Multi-connection transactions**

- Distributed transactions
  - "**global** transaction" (XA transaction, JTA transaction) consisting of parallel connections with **local** SQL transactions
- Long-transactions (see "hierarchy of transaction concepts")
  - sequence of transactions
    - typical case of a use transaction
  - "very long": workflow models

Distributed Transaction Two-Phase Commit (2PC)

(XA transaction)



Concurrency Control Technologies

- SQL standard defines Isolation Levels for transaction context based on anomalies, without concerning the technologies

- Concurrency Control Implementations tuned by Isolation Levels:
  - Optimistic Concurrency Control (OCC) 100% isolated
  - Locking Schemes (LSCC) 0% ..100%
  - Multi-Versioning (MVCC) %?
  - Cursor level concurrency control,
    SELECT .. FOR UPDATE

- Client-side
  - Row Version Verification (RVV) aka. "Optimistic Locking"

Concurrency Problems

Typical anomalies (C J Date, Milton, SQL-92)

1 Lost Update Problem

2 Uncommitted Dependency Problem (Dirty Read)

3 Inconsistent Analysis Problems

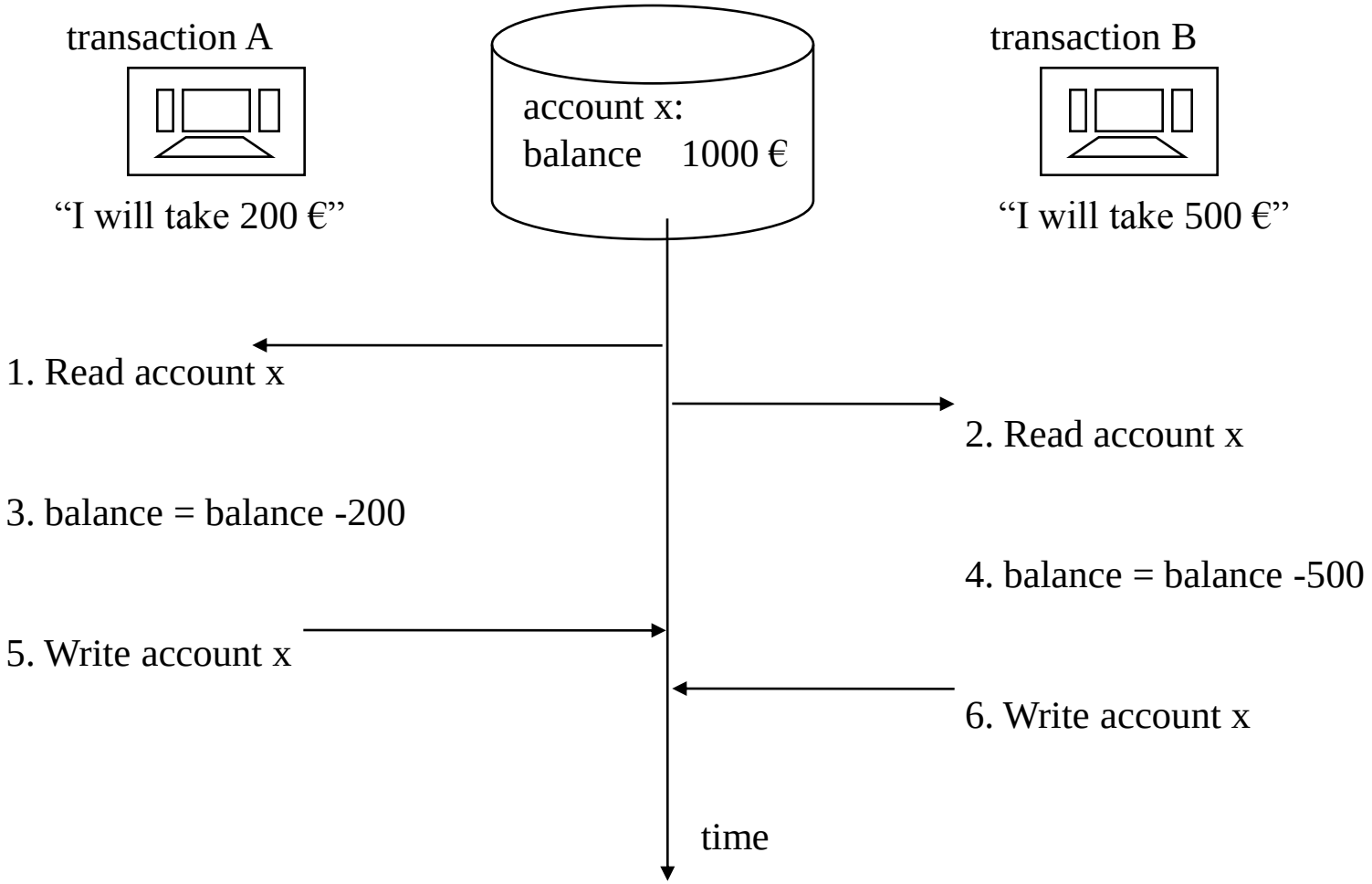
a) Decreasing Read Set (Nonrepeatable Read)

b) Increasing Read Set (Phantoms)

1. The Lost Update Problem

C. J. Date: Lost Update

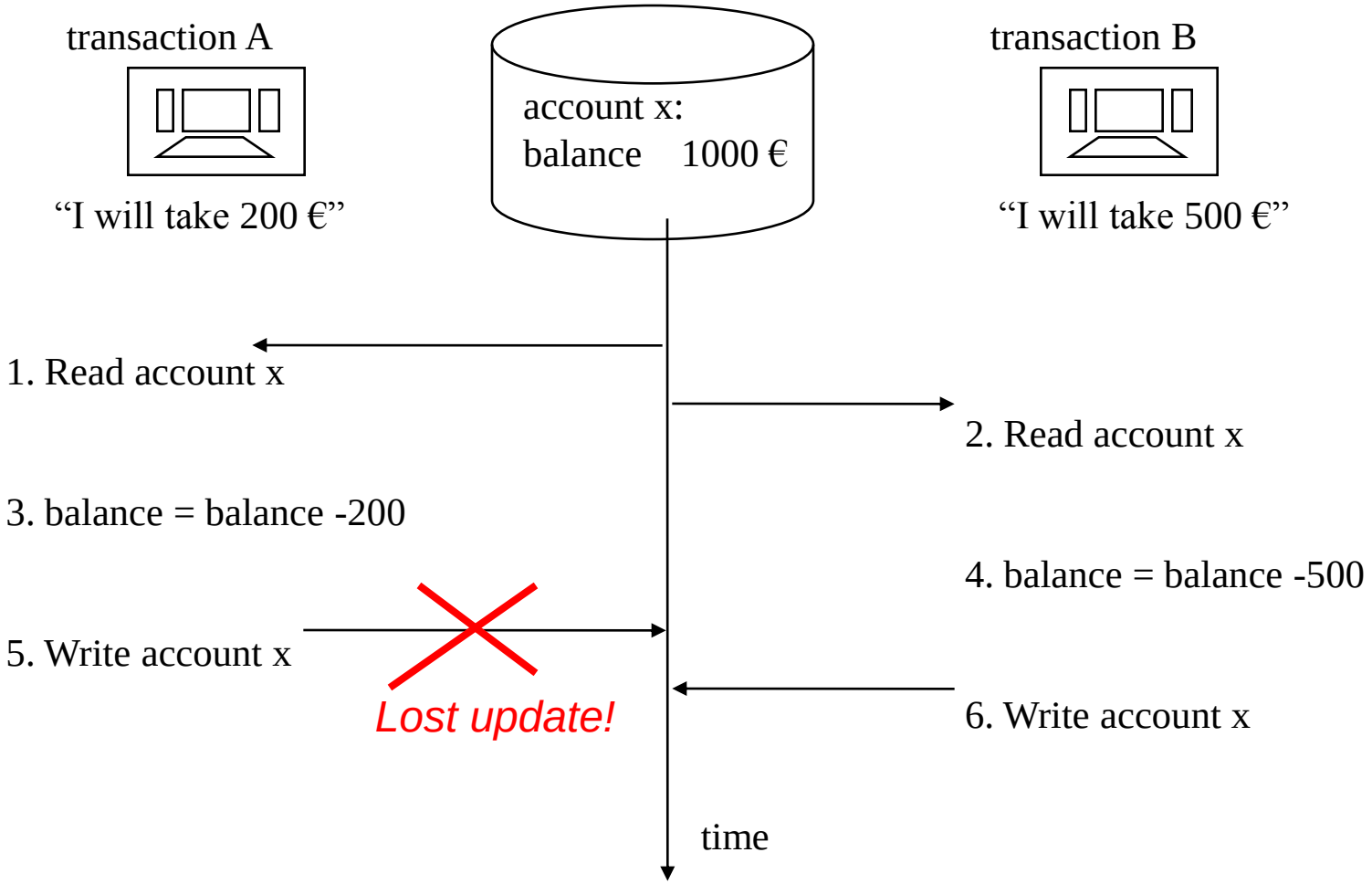
”Tellers”



1. The Lost Update Problem

C. J. Date: Lost Update

”Tellers”



Locking Scheme Concurrency Control (LSCC)

Compatibility of S and X locks

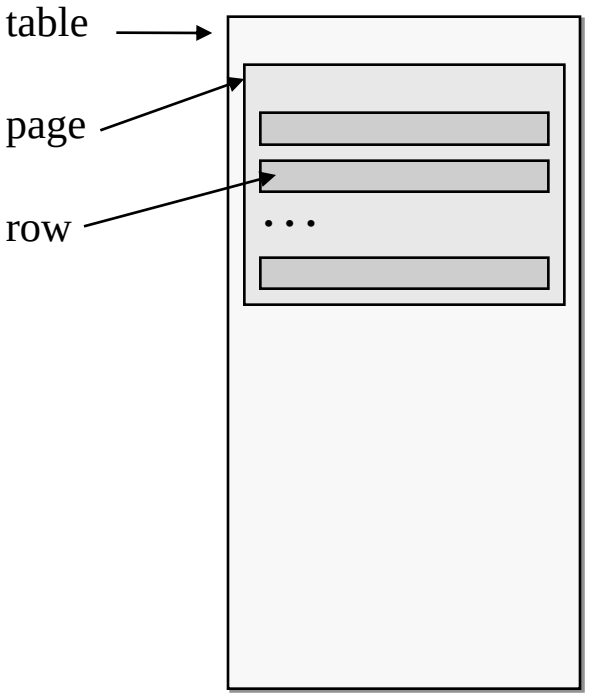
Lock of transaction A to object o

Lock request of transaction B to object o

	Shared	e**X**clusive
Shared	**Grant**	**Wait !**
e**X**clusive	**Wait !**	**Wait !**

- S-lock grants read access to object
- X-lock grants write access to object
- X-lock request after getting S-lock is called as lock promotion

Locking granularity:

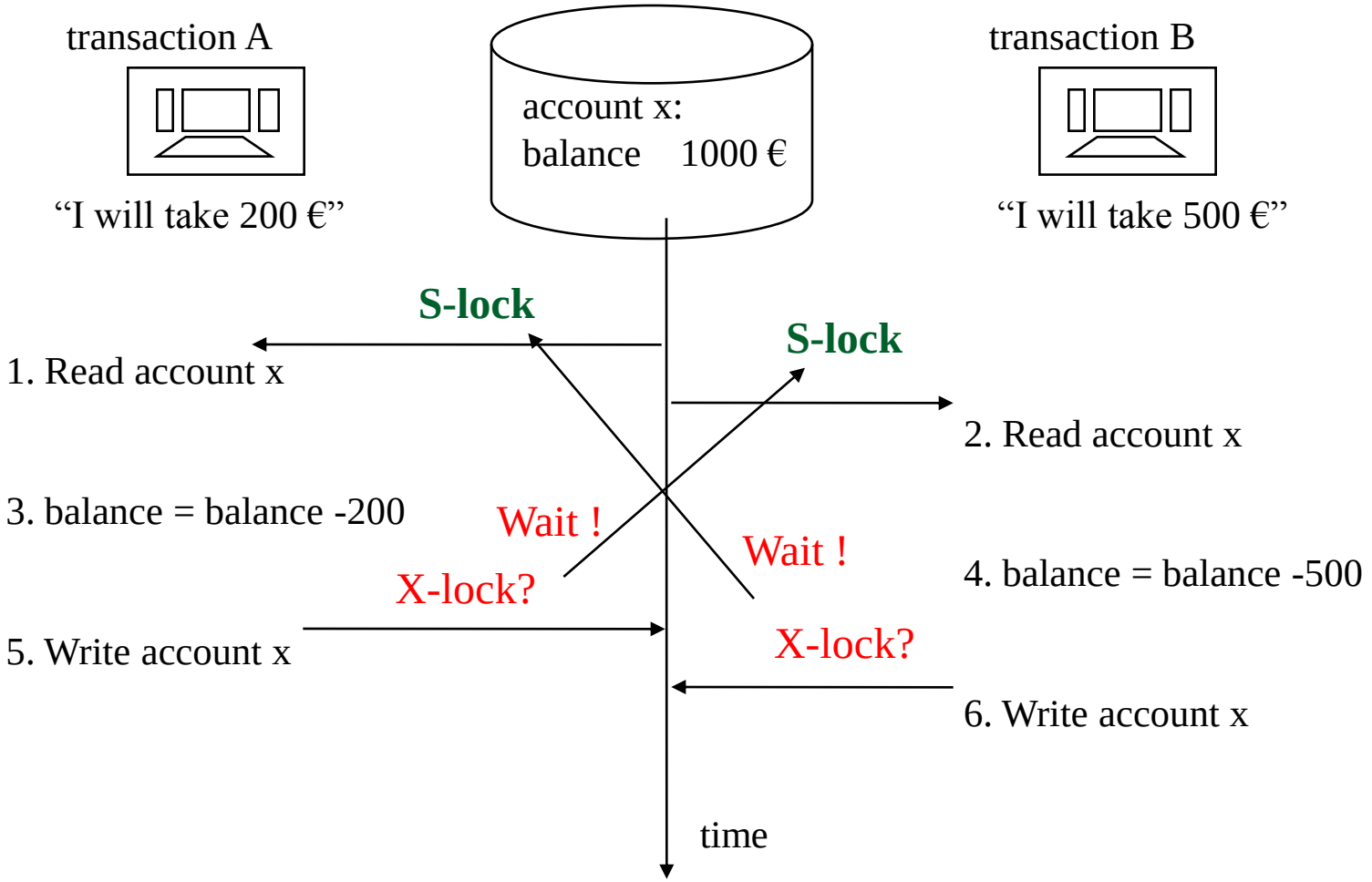


1. The Lost Update Problem

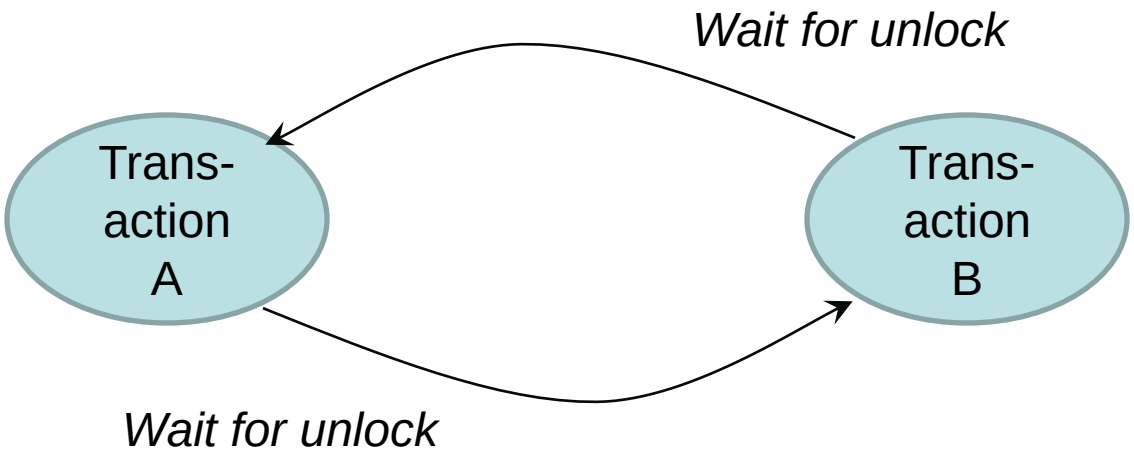
C. J. Date: Lost Update

- Applying the locking scheme:

”Tellers”



Wait for Cycle of the Lock Waits

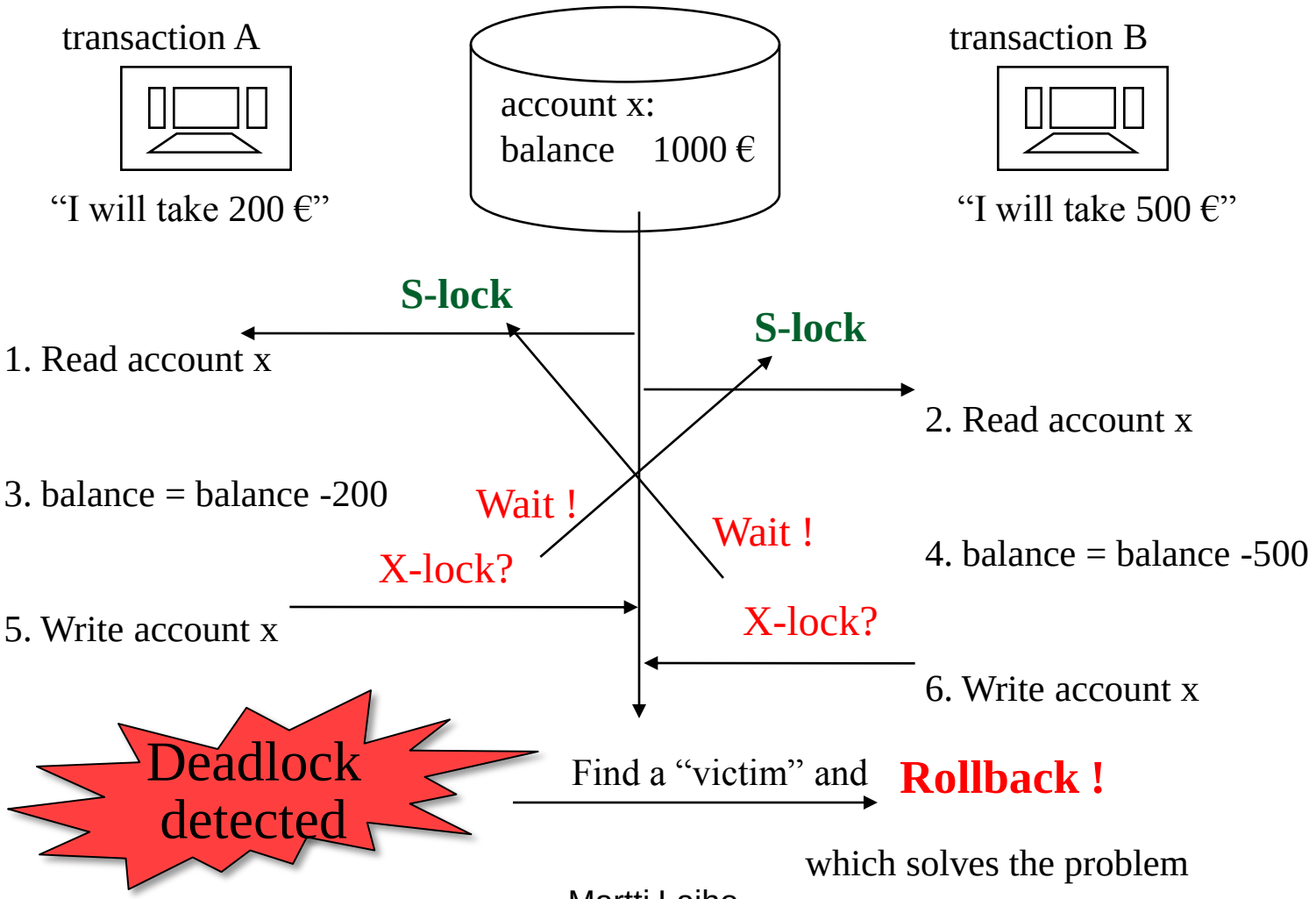


1. The Lost Update Problem

C. J. Date: Lost Update

- Applying the locking scheme:

”Tellers”

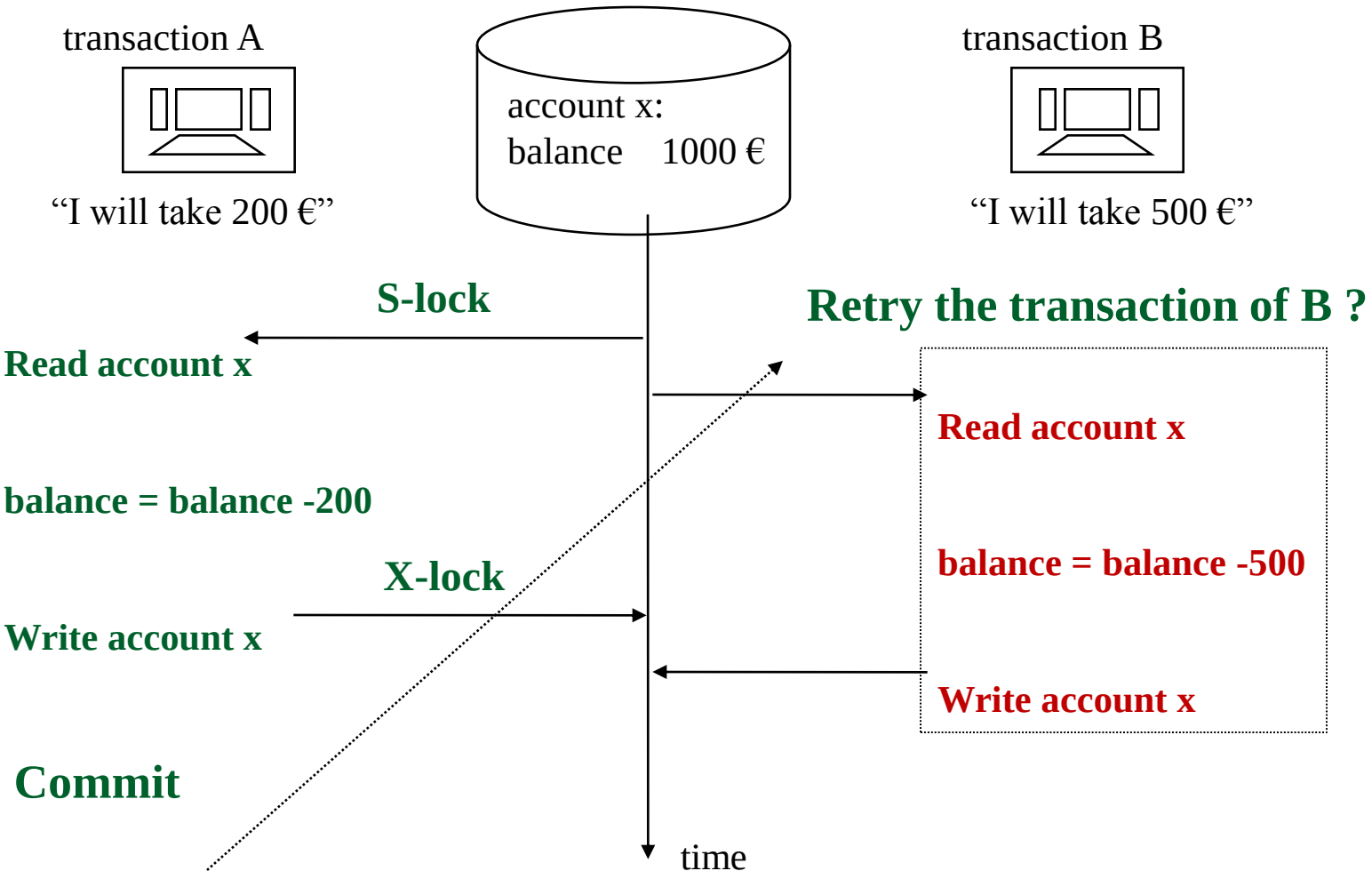


1. The Lost Update Problem

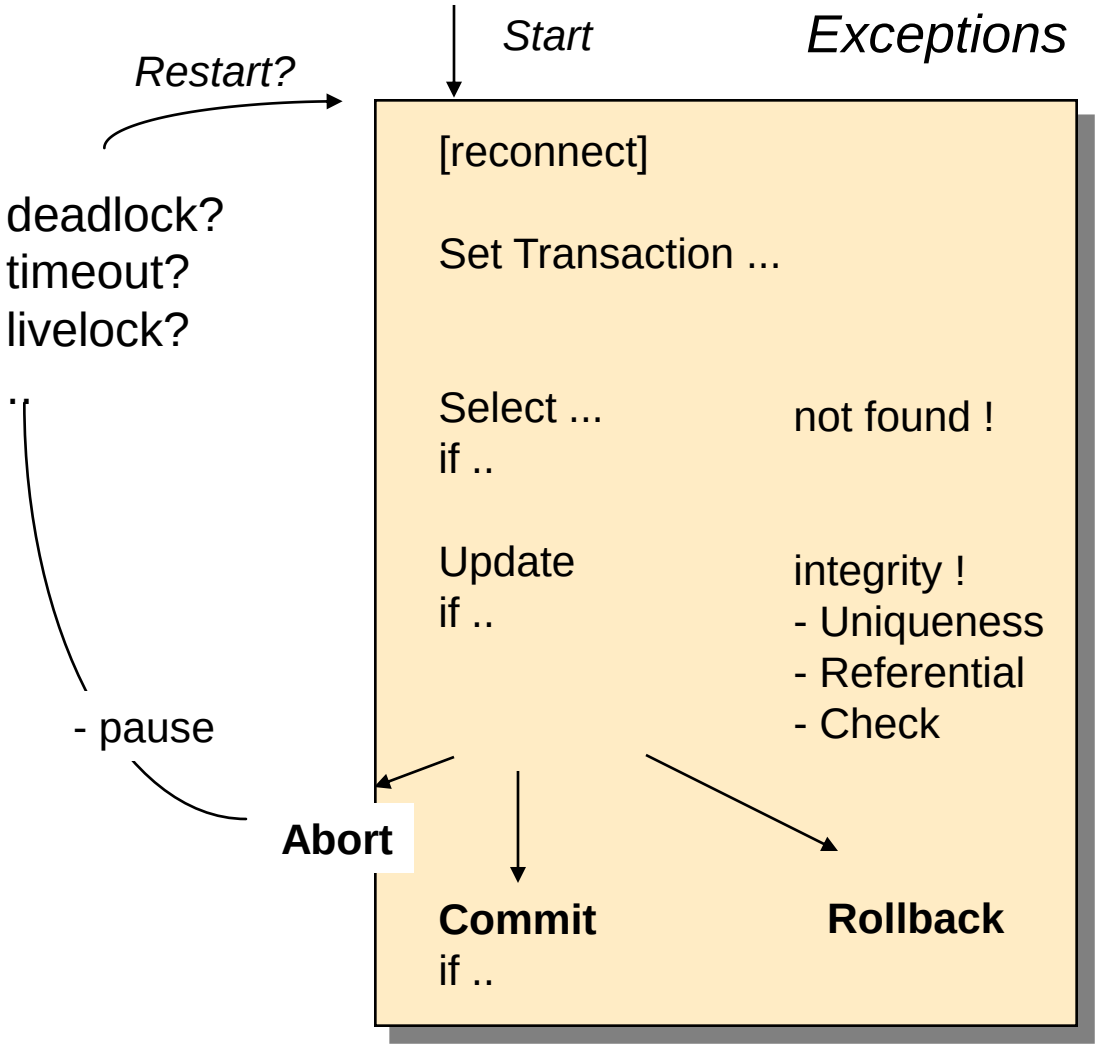
- solved by locking scheme:

C. J. Date: Lost Update

”Tellers”



Retryer Pattern



Errors	*Restart?*
multiprogramming - limit, timeout	YES
syntax error in dynamic SQL	
reoptimising - objects/privileges	
serializability - conflict - deadlock - timeout	YES
services of - DBMS - buffers, etc - OpSys - data communication - HW	YES?

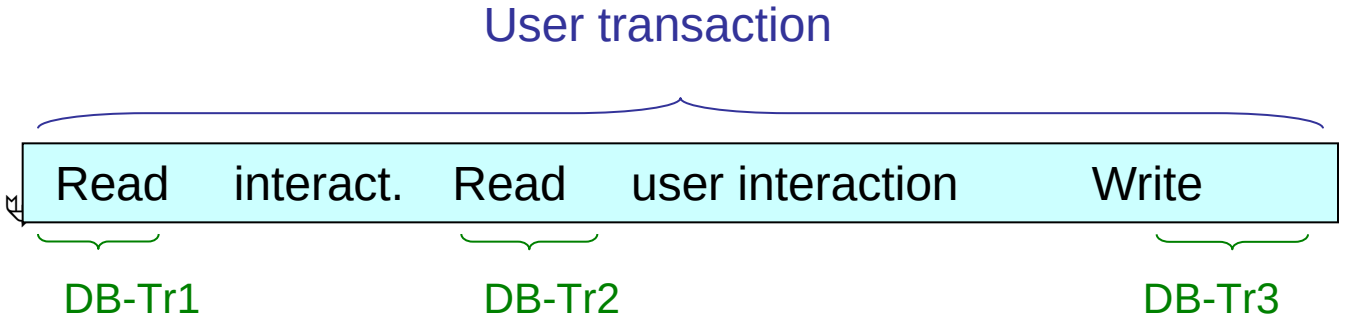
SQL-92: Isolation Levels

Isolation Level: \ Phenomena:	**Lost Update**	**Dirty Read**	**NonRepeatable Read**	**Phantoms**
READ UNCOMMITTED	NOT possible	possible !	possible !	possible !
READ COMMITTED	NOT possible	NOT possible	possible !	possible !
REPEATABLE READ	NOT possible	NOT possible	NOT possible	possible !
SERIALIZABLE	NOT possible	NOT possible	NOT possible	NOT possible

*The **Lost Update Problem (case 1)***
in context of a single transaction is considered solved!

SQL transactions in a User Transaction

- User transaction usually spans multiple DB-transactions
  - DBMS Transaction Manager cannot help to ensure ACID properties in that sequence



- Locking is inappropriate for i.e. Web applications
  - Because of
    - New stateless connections, but locking is bound to a connection
    - Unlimited number of concurrent users
    - User may go for a coffee break in the middle of a transaction
    - User may not run the user transaction up to the end

Losing Data due to Blind Write Problem

A sample run:

step	Process A	Accounts balance	Process B
1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;		
2		1 000 €	
3	SELECT balance INTO :balance FROM Accounts WHERE acctId = :id;		
4	**[COMMIT;]**		
5	newBalance = balance + 100;		UPDATE Accounts SET balance = balance + 200 WHERE acctId = :id;
6		1 200 €	COMMIT;
7	UPDATE Accounts SET balance = :newBalance WHERE acctId = :id;		
8	COMMIT;	1 100 €	

Blind Write based on stale data overwrites update of B

Lost Data

Well known problem, solved more than 30 years ago!

So, why should it be reconsidered? …

Update Types with respect to Blind Overwriting Problem

”Lost Update case 2”

- (Type -1): Blind Write based on stale data

  - High risk of lost updates

- Type 0: Sensitive / Cumulative Update (a := a + Δ)

  - No risk of lost updates

”Optimistic Locking”, ”Weak Locking”, ”RVV”

- Programming disciplines for avoiding lost updates

- Type 1: Update with RVV predicates

- Type 2: Select and conditional update with RVV predicates

Type 0 – avoiding Blind Overwriting

step	Process A	Accounts balance	Process B
1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;		
2		1 000 €	
3	SELECT balance INTO :balance FROM Accounts WHERE acctId = :id;		
4	(COMMIT;)		
5	~~newBalance = balance + 100;~~		UPDATE Accounts SET balance = balance + 200 WHERE acctId = :id;
6		1 200 €	COMMIT;
7	UPDATE Accounts SET balance = balance + 100 WHERE acctId = :id;		
8	COMMIT;	1 300 €	

Sensitive update →

(also a sensitive update)

Type 1 – avoiding Blind Overwriting

1. transaction

2. transaction

step	Process A	Accounts balance	Process B
1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;		
2		1 000 €	
3	SELECT balance, rowVersion INTO :balance, :rowVersion FROM Accounts WHERE acctId = :id;		
4	COMMIT;		
5	newBalance = balance + 100;		UPDATE Accounts SET balance = balance + 200 WHERE acctId = :id;
6		1 200 €	COMMIT;
7	UPDATE Accounts SET balance = :newBalance WHERE acctId = :id AND (rowVersion = :rowVersion);		
8	COMMIT;	1 200 €	

Update with RVV predicate

Need to test if the update succeeded !

Final result

Type 2 – avoiding Blind Overwriting

- Select with conditional update

Transaction 2

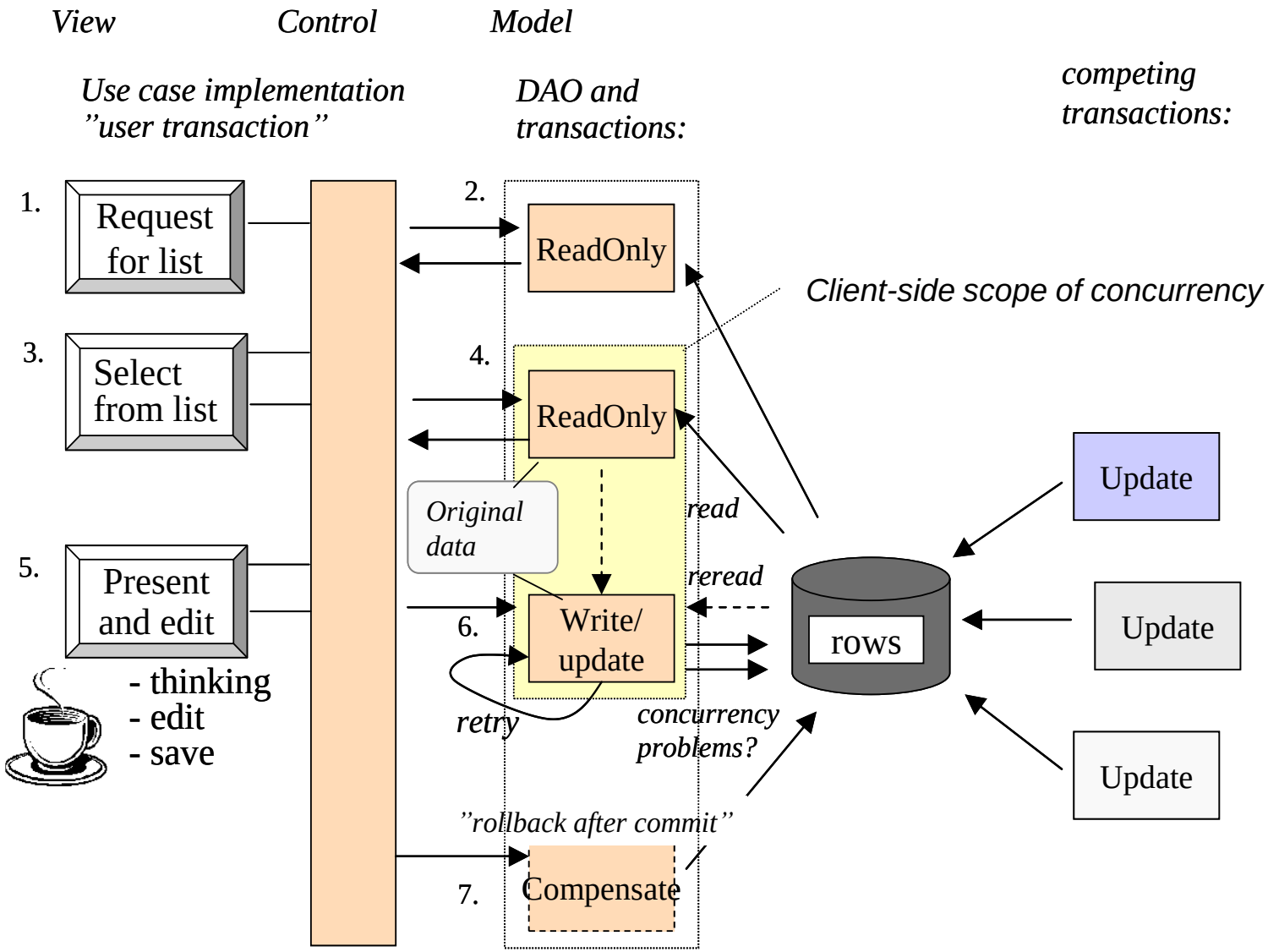
```
SET TRANSACTION ISOLATION LEVEL
    REPEATEBLE READ ;
SELECT rowVersion INTO :current_rowVersion
FROM Accounts
WHERE acctId = :id ;
...

if (current_rowVersion = old_rowVersion)
then
   UPDATE Accounts
   SET balance = :newBalance
   WHERE acctId = :id  AND
         :current_rowVersion = rowVersion;
```

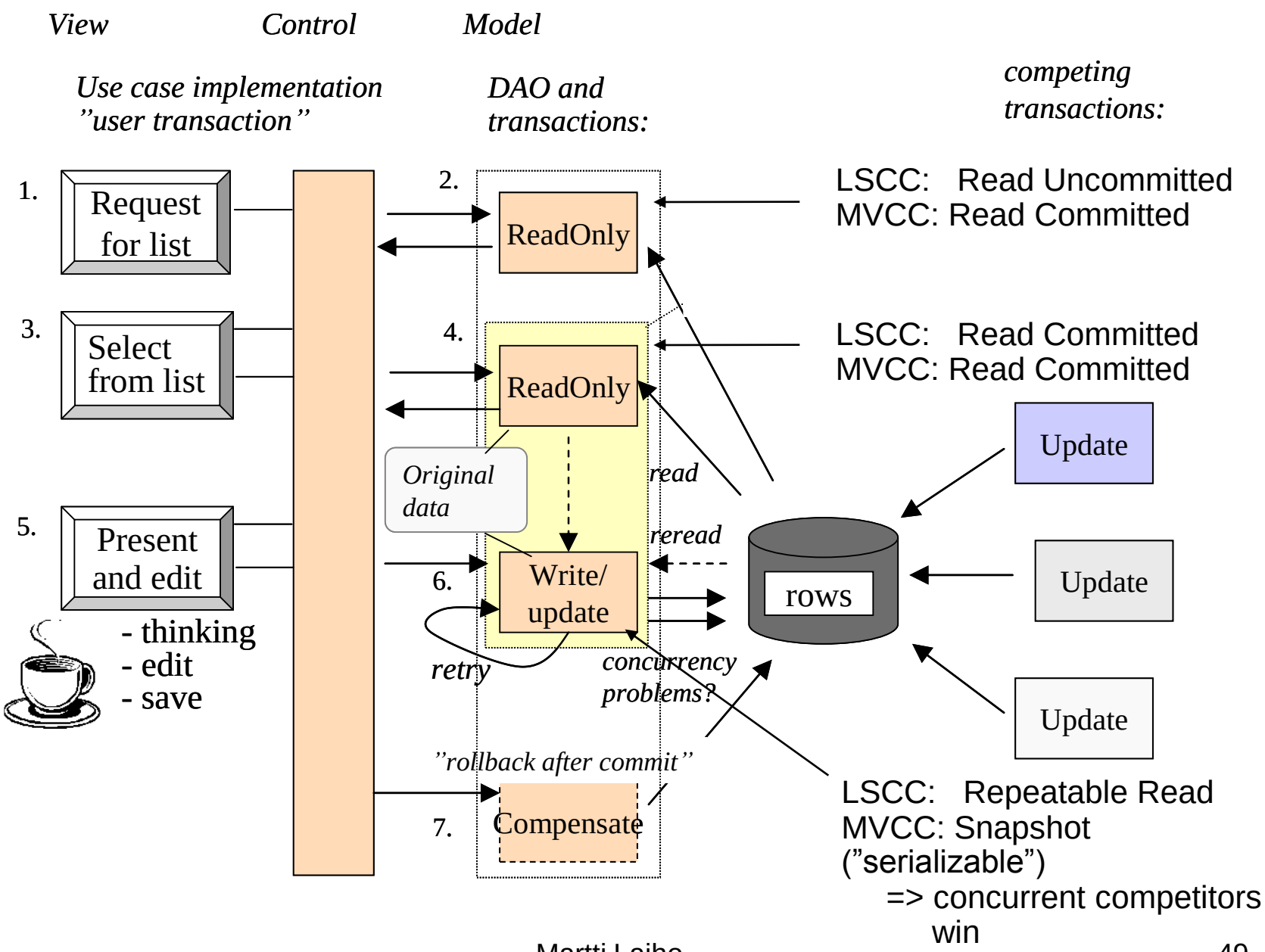
Conditional UPDATE

Need to test if the update succeeded !

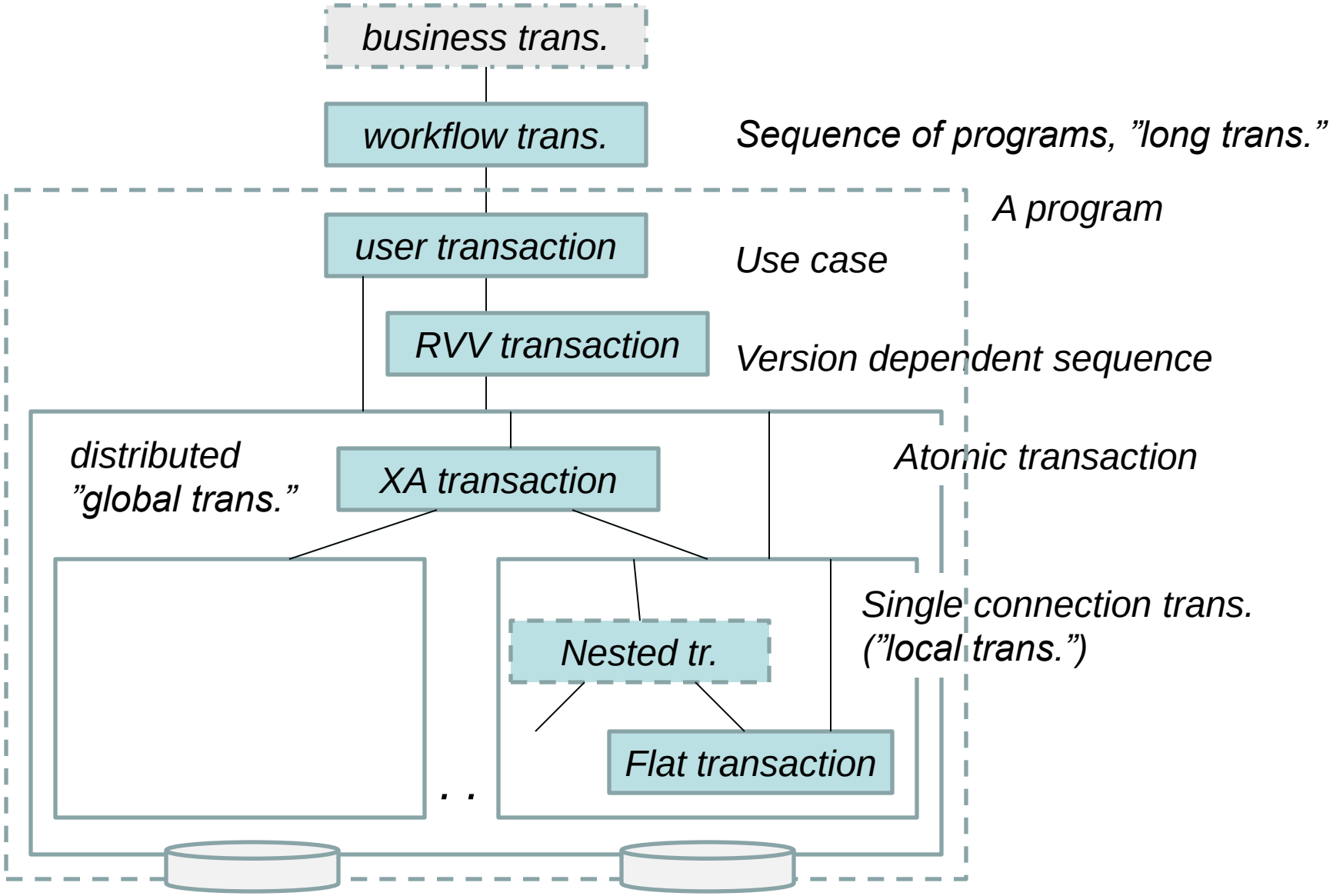
A Typical Use Case and its SQL Transactions



.. and its Isolation Levels



Hierarchy of Transaction Concepts



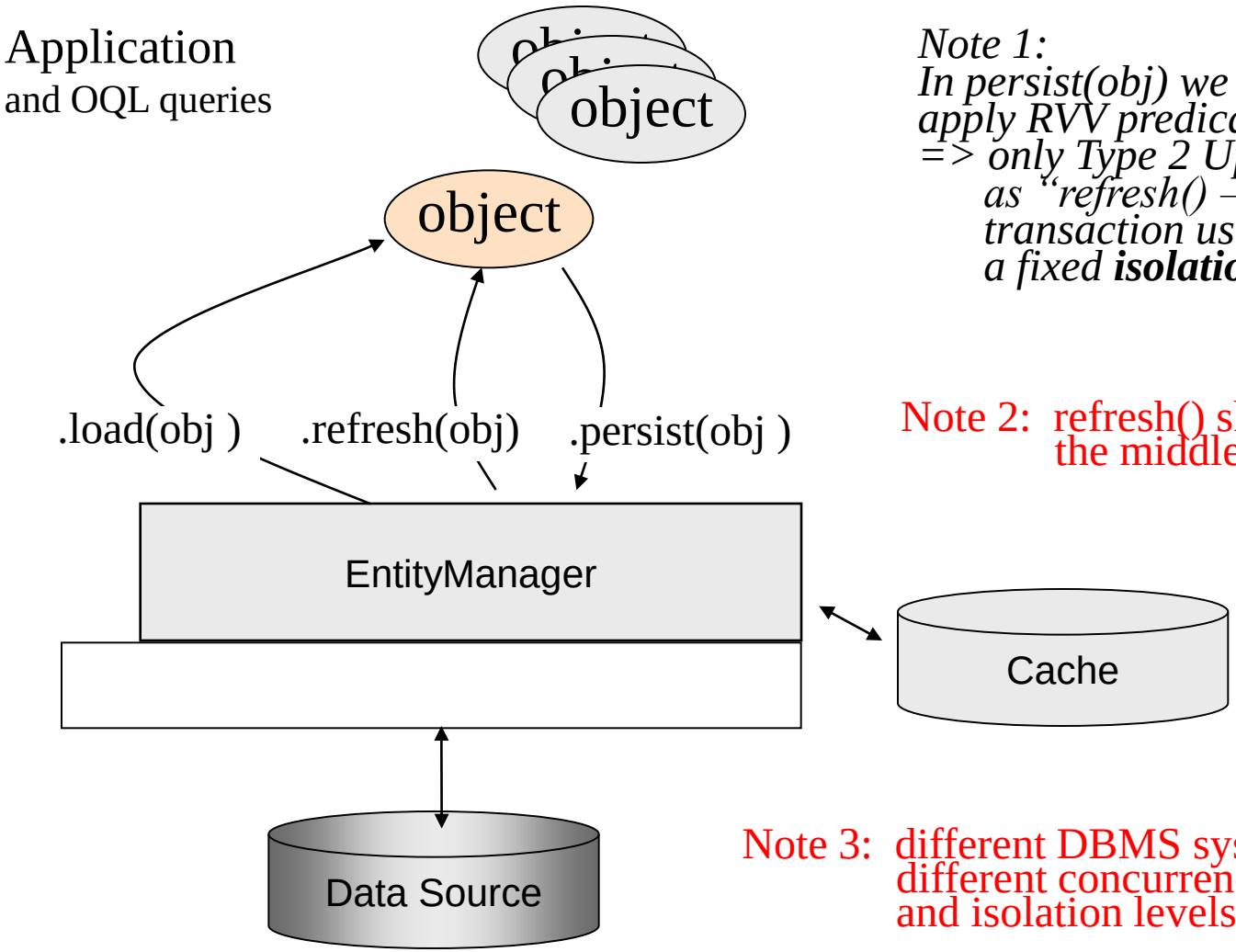
Object-Relational Mapping (ORM)

- Solves the paradigm mismatch between
  - Objects in Object-Oriented Programming (OOP) and
  - (”Relational”) SQL databases

- Concerns
  - Existing tables vs. generated tables
  - Middleware cache vs. DBMS bufferpool
  - Transactions with different Isolation levels ?
  - Can the Middleware hide the behavior of the DBMS ?

- ORM is the topic on Thursday 2010-05-06

ORM Stack of Services and APIs

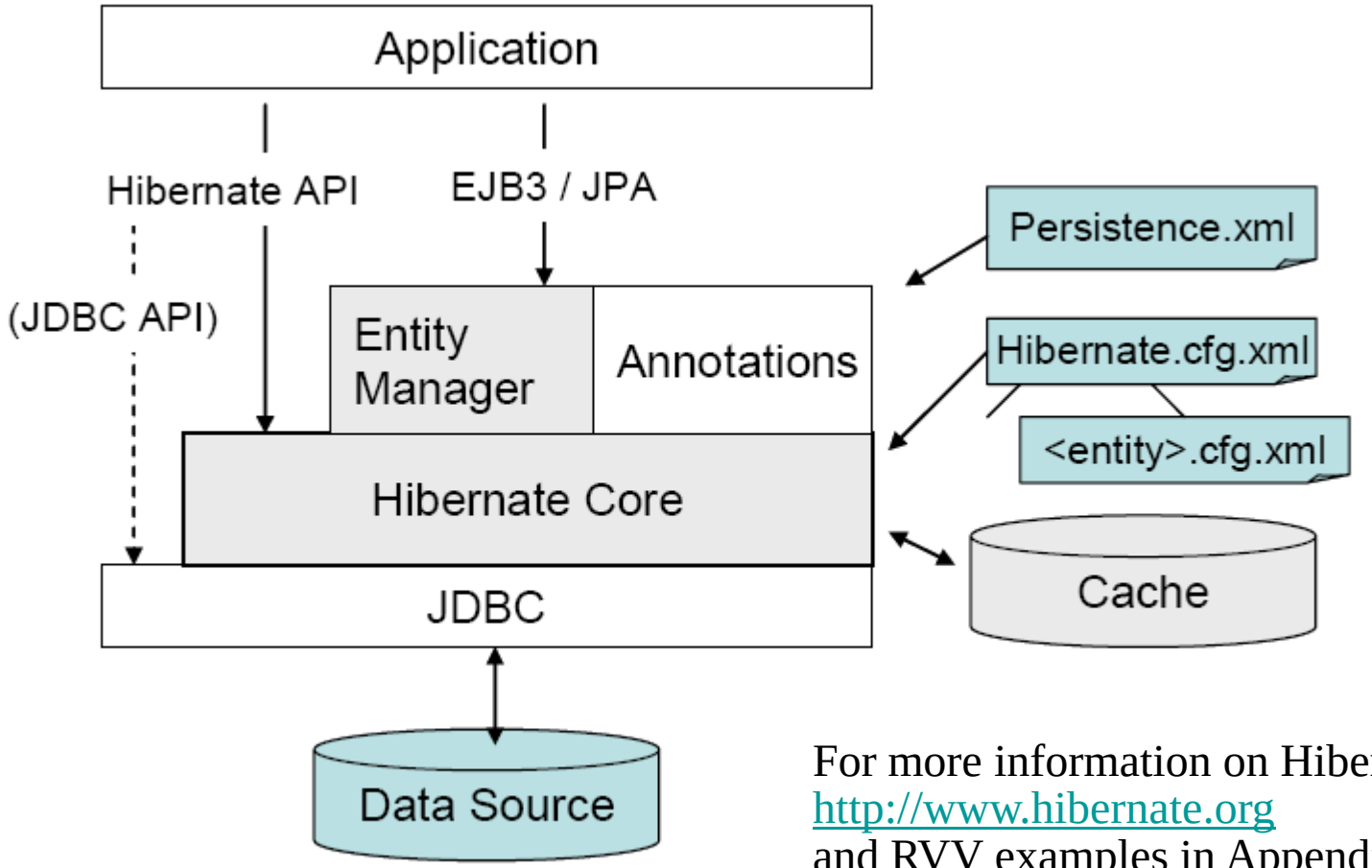


Note 1:
In persist(obj) we cannot apply RVV predicate
=> only Type 2 Update possible as "refresh() – persist()" transaction using a fixed ***isolation level***

Note 2: refresh() shall bypass the middleware cache!

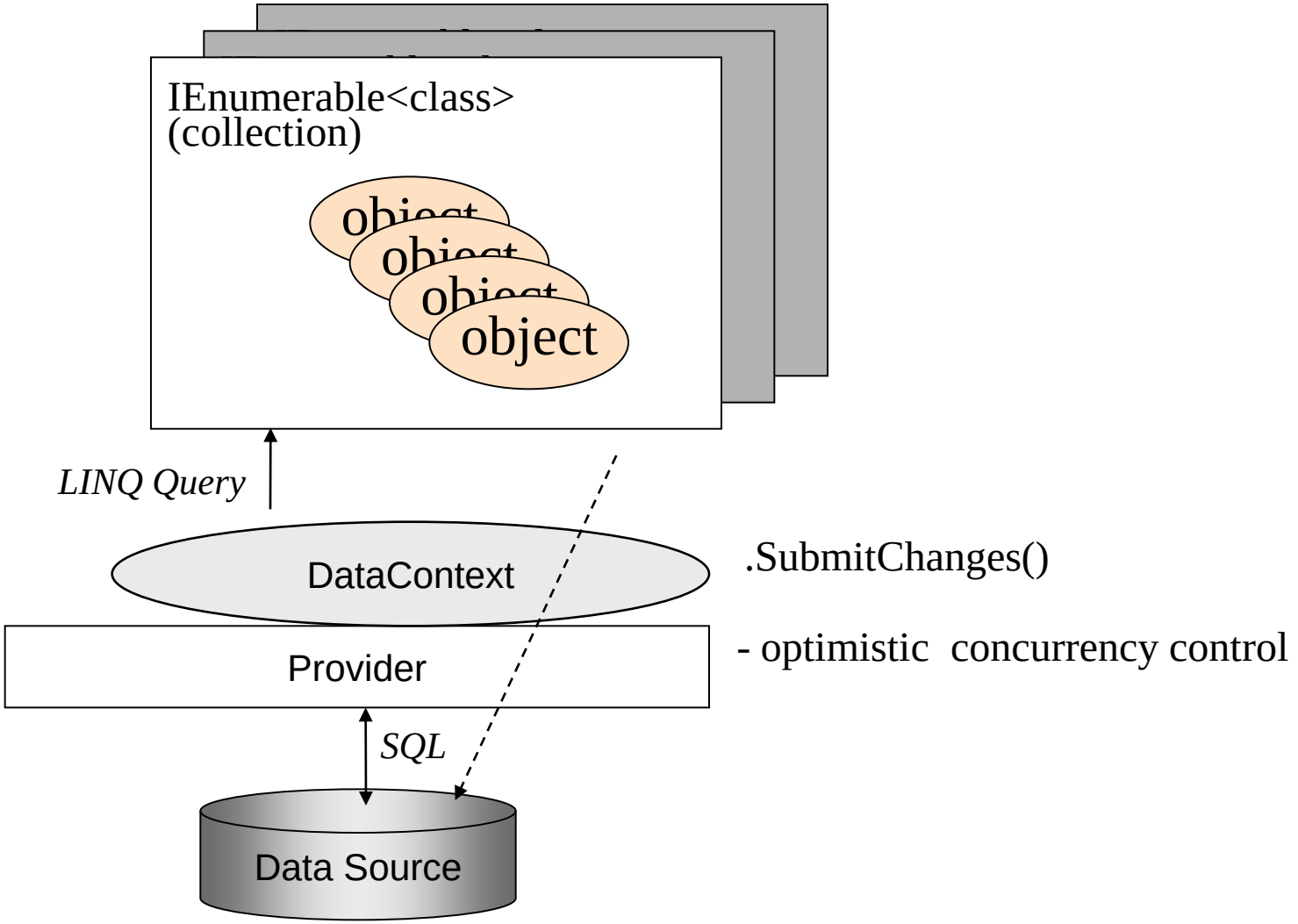
Note 3: different DBMS systems provide different concurrency services and isolation levels!

Hibernate Stack of Services and APIs

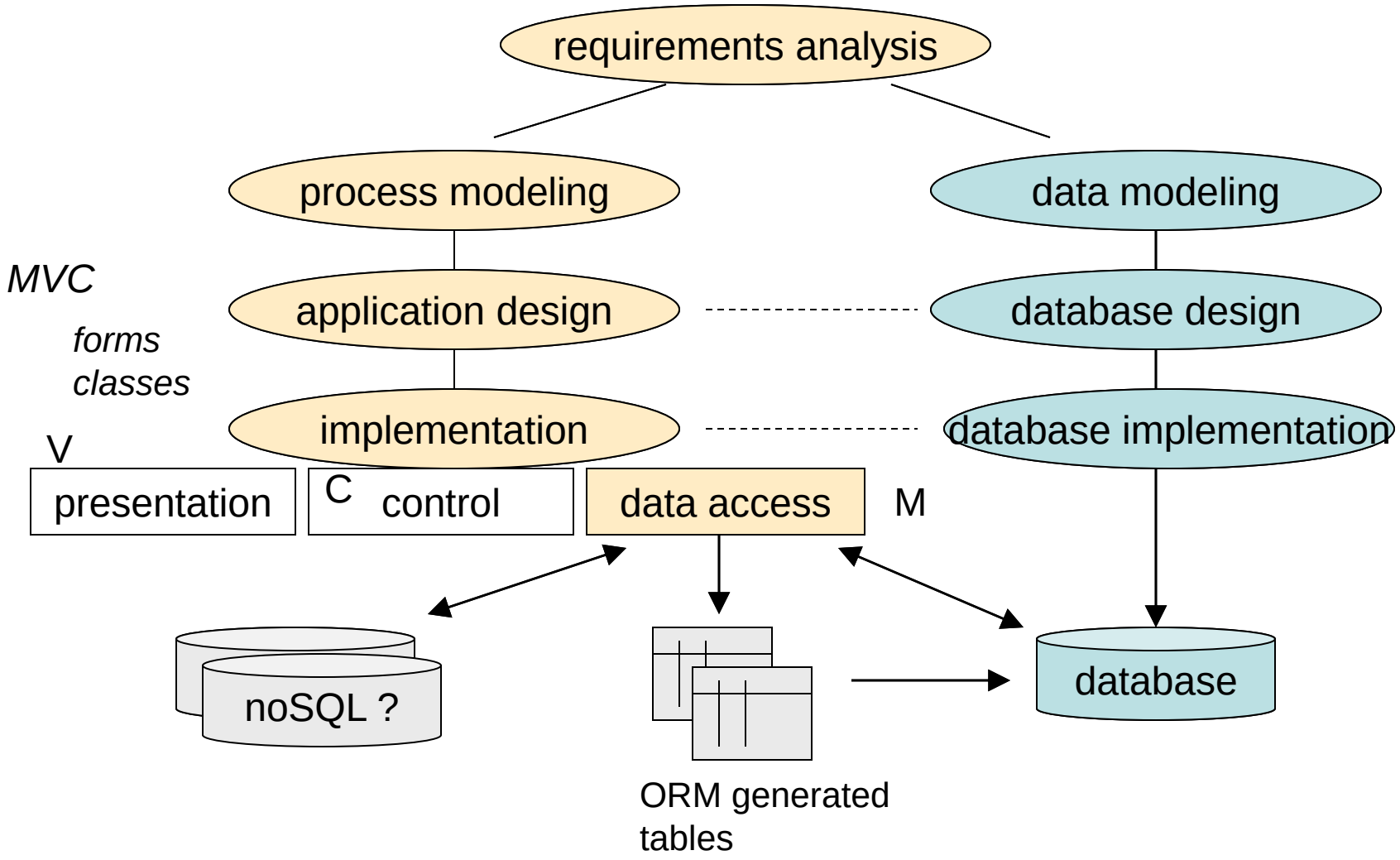


For more information on Hibernate see
http://www.hibernate.org
and RVV examples in Appendix 6 and 7

Persistence in LINQ to SQL

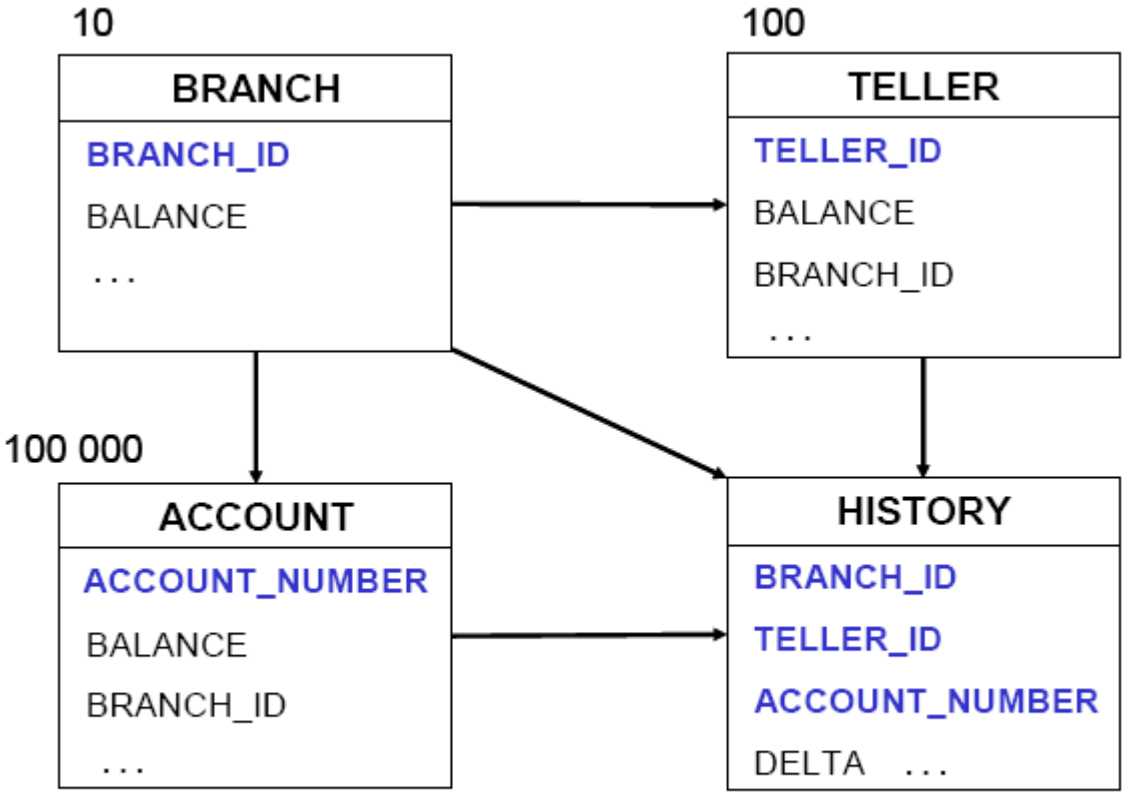


Methodologies and Implementations



Performance: TPCA Benchmark

See www.tpc.org



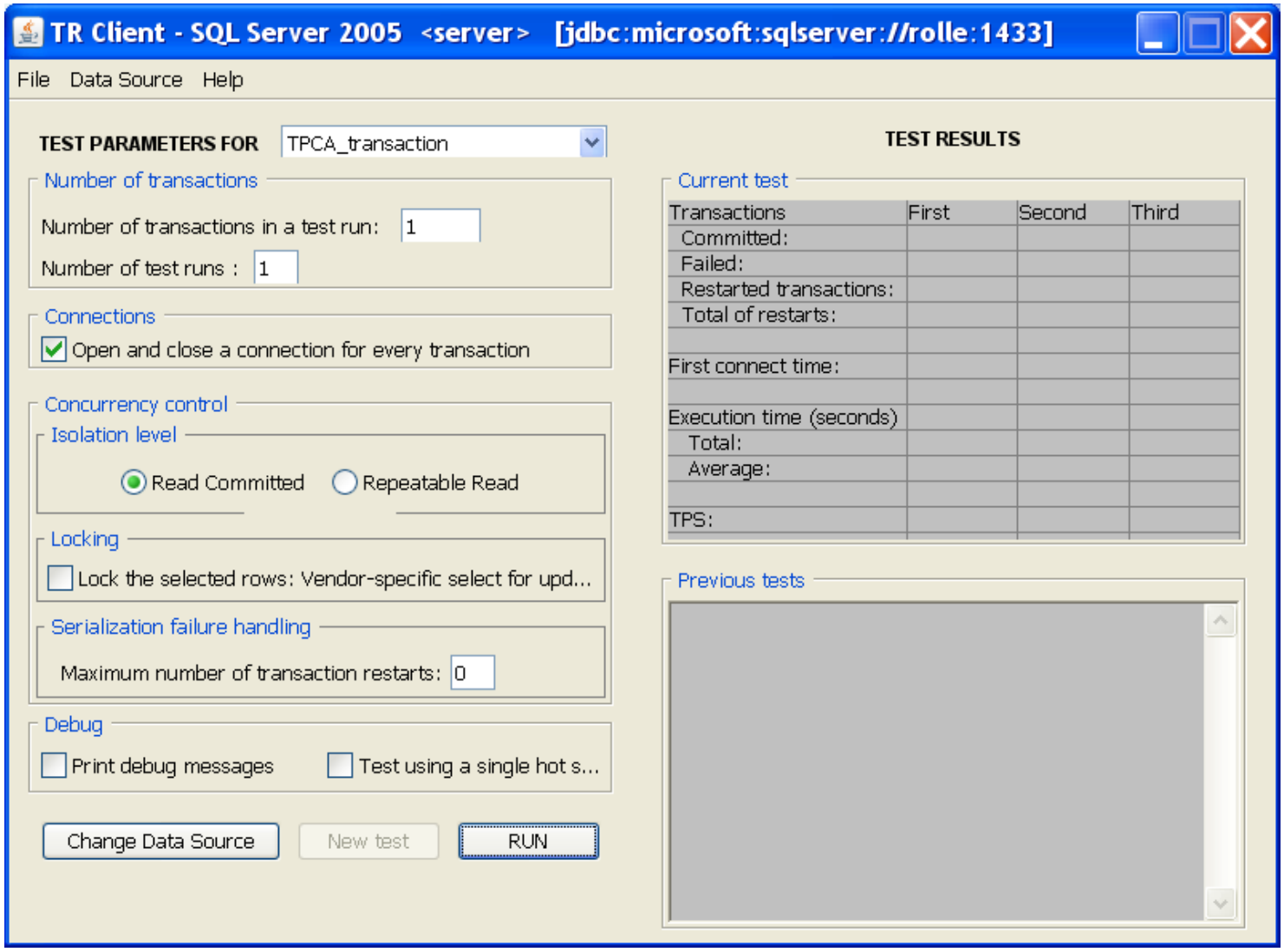
Tables of the TPCA BANK database of scale 1-tps

.. TPCA Transaction

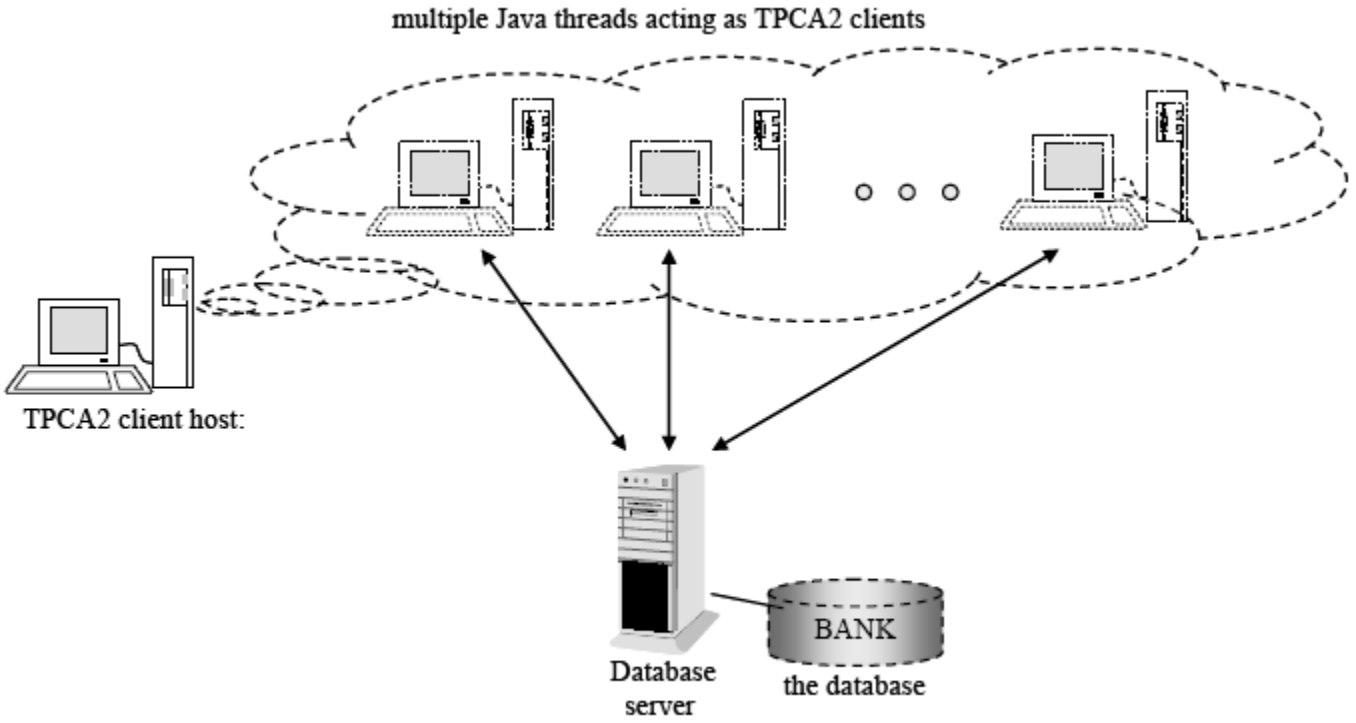
The simplified algorithm of TPCA transaction is the following:

1) Generate a random **ACCOUNT_NUMBER** and a random **TELLER_ID**.
2) BEGIN transaction
3) Select the **ACCOUNT** having the ACCOUNT_NUMBER, read its BALANCE and BRANCH_ID
4) Add 100 ("delta") to the BALANCE of the **ACCOUNT**
5) Add 100 to the BALANCE of the **TELLER** having the TELLER_ID
6) Add 100 to the BALANCE of the **BRANCH** having the BRANCH_ID
7) Insert a **HISTORY** record of this business transaction
8) COMMIT the transaction

GUI of TPCA Lab (DBTech Pro)



TPCA 2.1 (CCLab2)

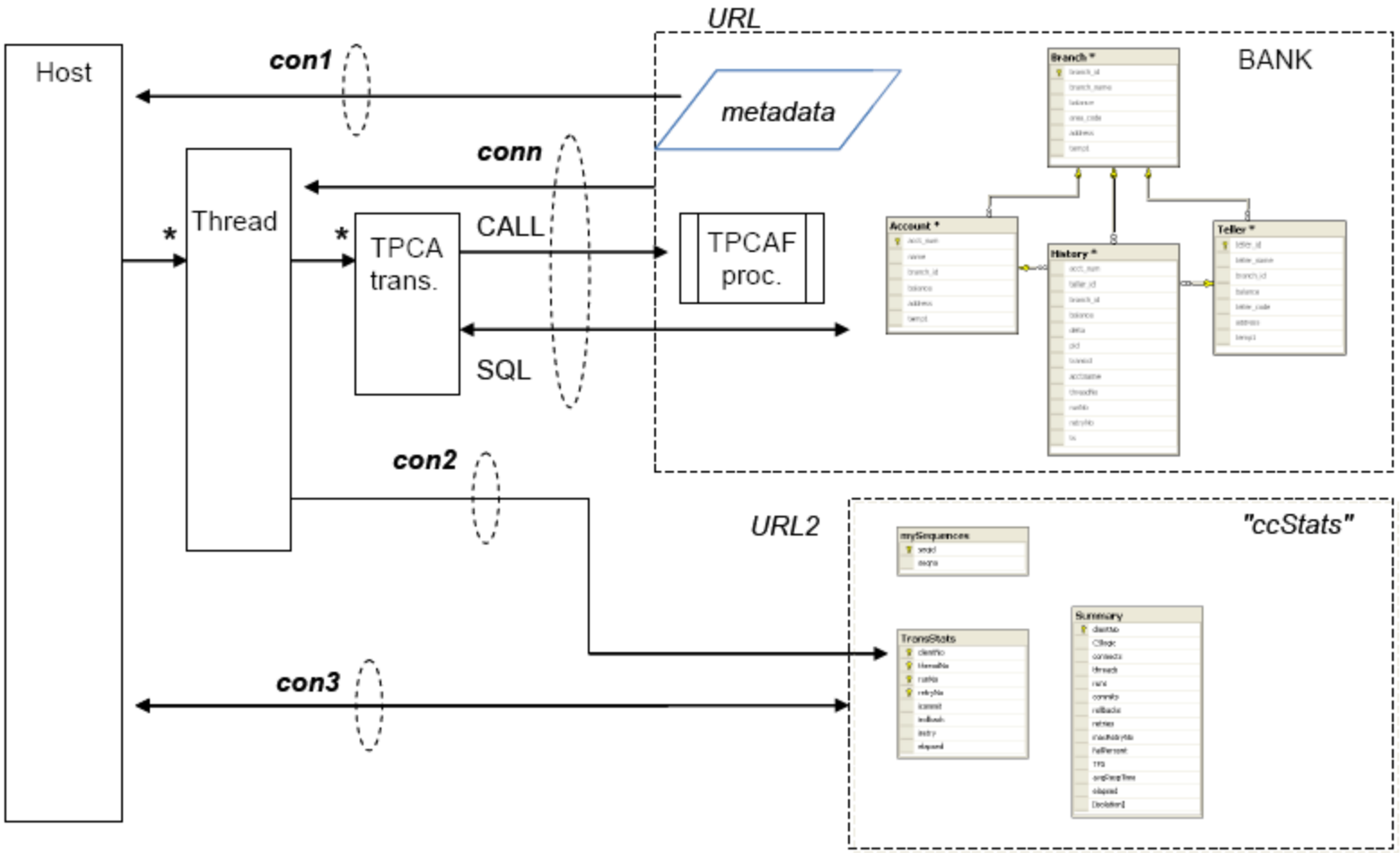


TPCA2 client host using multithreading for client connections

Parameters of TPCA 2.1 Client TpcaScales

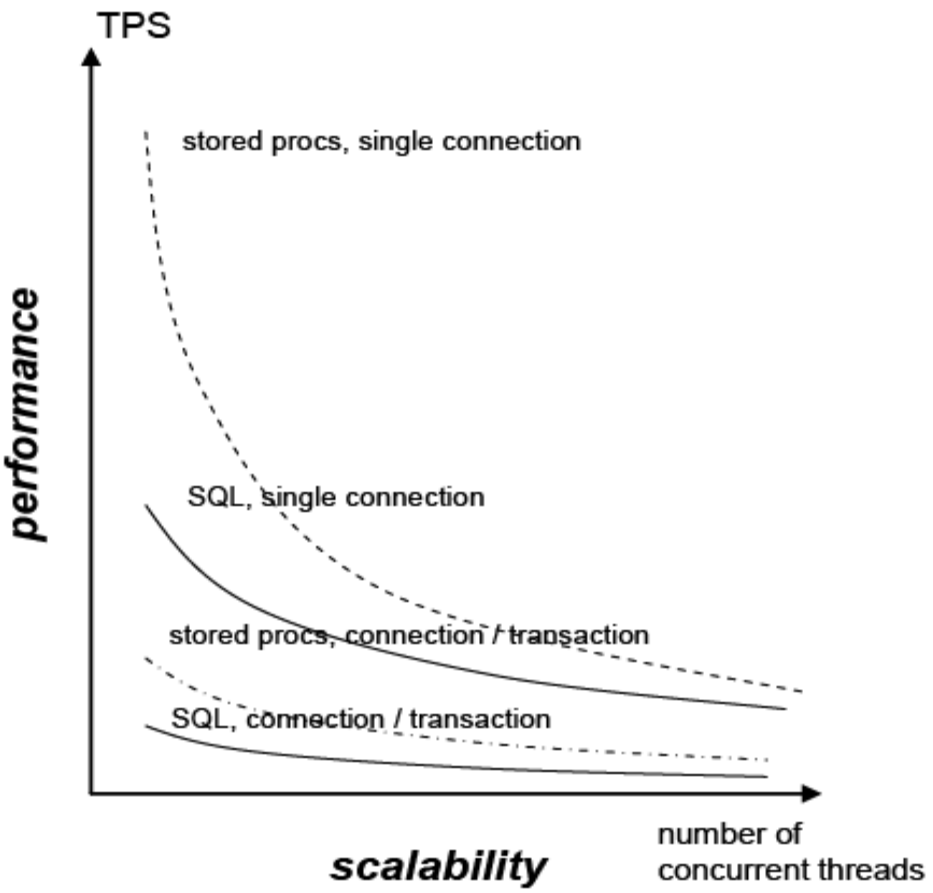
```
rem The Bank database:
SET DRIVER="com.ibm.db2.jcc.DB2Driver"
SET URL="jdbc:db2://localhost:50000/BANK"
SET USER=USER1
SET PSW=sql
rem CC Statistics database:
SET DRIVER2="com.microsoft.sqlserver.jdbc.SQLServerDriver"
SET URL2="jdbc:sqlserver://localhost;instance=SQLEXPRESS;DatabaseName=DB2BANK"
SET USER2=USER1
SET PSW2=sql
rem Parameters:
SET CONNECTS="1,2"
rem          1=connect/thread, 2=connect/transaction
SET CSLOGIC="1,2,3"
rem          1=statemets, 2=preparedStatements, 3=stored procedures
SET ISOLATION="2,3,4"
rem          JDBC isolation levels: 1=Read Uncommitted, ..
SET THREADS="1,2,3,4,5,6,7,8,9"
SET VERBOSE=1
SET RUNS=40
SET RETRIES=10
SET ROWLOCK=""
SET HOTSPOT=0
SET
CLASSPATH=.;TPCA2_SQL\sqljdbc4.jar;TPCA2_ORA\ojdbc14.jar;TPCA2_DB2\db2jcc4.jar
java TpcaScales %DRIVER% %URL% %USER% %PSW% %ISOLATION% %CONNECTS% %CSLOGIC%
%VERBOSE% %THREADS% %RETRIES% %RUNS% %ROWLOCK% %HOTSPOT% %DRIVER2% %URL2%
%USER2% %PSW2% >SQL_scales.txt
```

Classes, Connections and Tables



Classes and database connections of the TPCA2 client program

Expected Perfomance/Scalability Scenarios



Hypothetical Performance/Scalability scenarios of some DBMS on depending on number of concurrent TPCA transactions (and threads)