DBTech VET

www.DBTechNet.org

Lifelong Learning Programme

Introduction to SQL Transactions

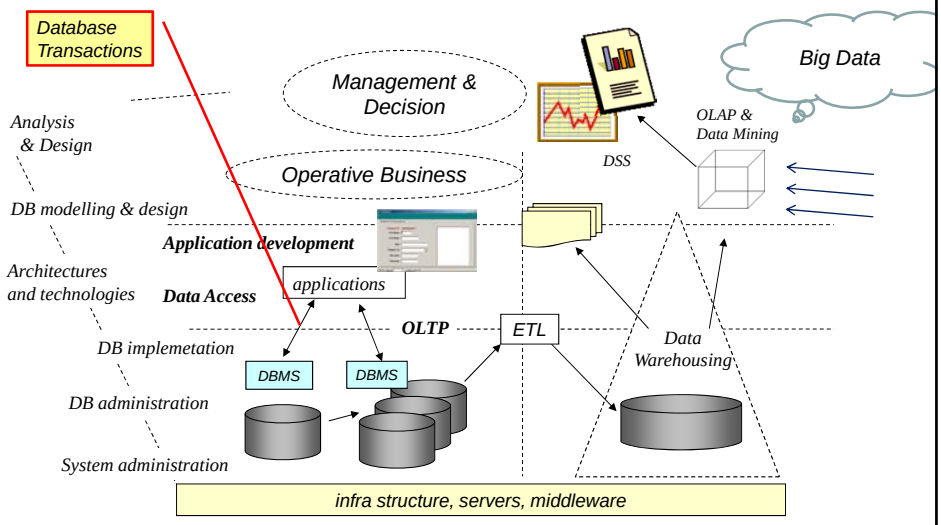
for teachers, trainers and application developers

martti.laiho@gmail.com

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.

DBTech Pro

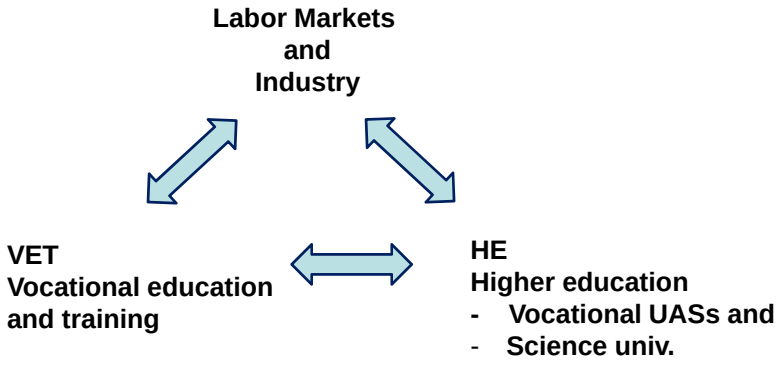
Areas in Database Technologies



Helia / Martti Laiho and Kari Silpiö

Context of our project

Europe 2020 strategy:



Martti Laiho 3

Problems and need for Transactions

- Today society, infra structures, business, and every day life of citizens are **dependant on ICT** and software using **OLTP databases**, which provide the most **reliable services** for storing and retrieving the needed data
- However, **inproper access** to database services results in **erroneous or missing data** causing difficulties, lost business, etc
  - **Missing orders, shipments, payments, ..**
  - **Double-bookings, double-invoicing, ..**
  - **Delays, erroneous information, ..**
  - **even catastrophes**
- Professional use of database services avoids these problems accessing database only by **well-designed database transactions** which are the basic building blocks of fault-tolerant applications

Martti Laiho 4

DBTech VET Teachers

Trends and worries

- Students avoiding database technologies
- Teachers avoiding transaction programming subjects
- Transaction programming skills and competence decreasing in the industry

Motivation, knowledge, skills and competence have to be improved – it starts from education!

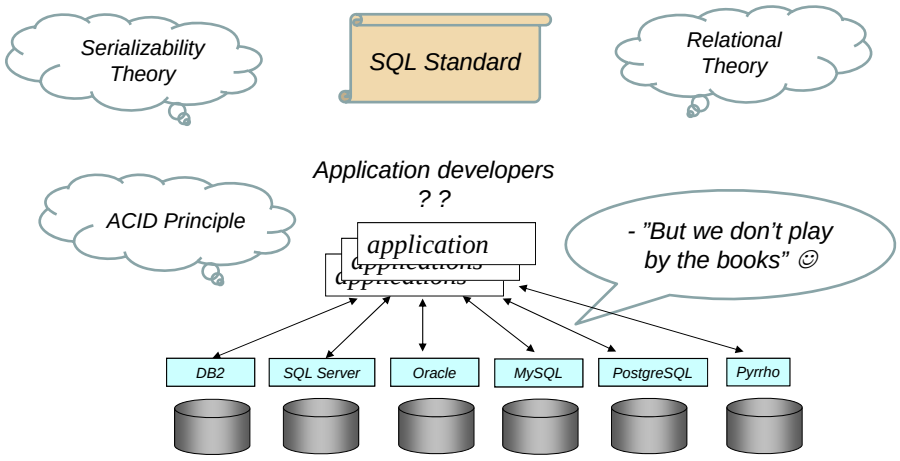
DBTech VET Teachers

- Focus on vocational education and **industry needs**
- Education of application developers in **reliable use of the current DBMS products, knowledge, skills and competence**

- Pedagogy: **Learning by experimenting and verifying**
- Motto: **Zero tolerance** for incorrect data!

We need to get industry, teachers and students to understand that transaction programming is **important and *exciting!***

OLTP - Theories and Practice?



Differently behaving products

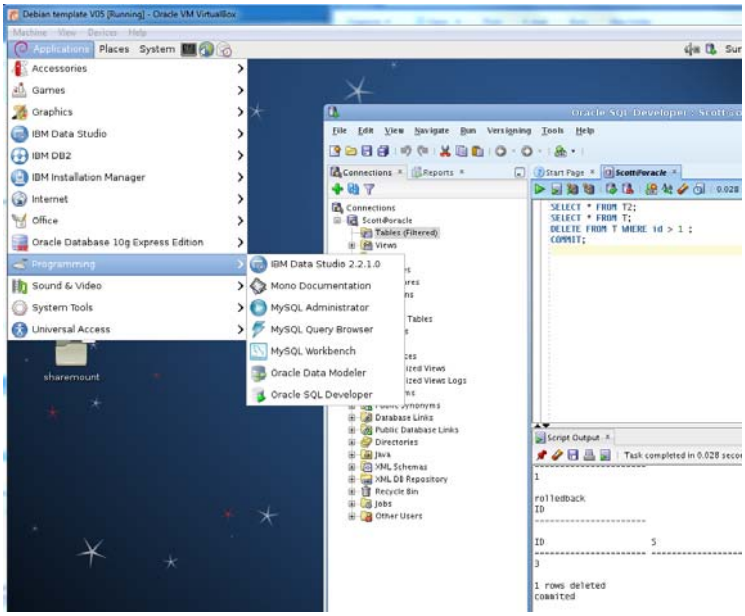
- As default in AUTOCOMMIT mode / transactional mode?
- Implicit or explicit starts of transactions
- Implicit COMMIT on DDL ?
- Default isolation level ?
- What is considered as Error or Warning ?
  - Value truncation, value overflow, …
- Error in command
  - Rolls back the command, compound command
  - Rolls back the command and discards commands until end of transaction
  - Rolls back the transaction
- Concurrency control mechanisms, …
- **New versions** may change the behavior of the DBMS product

Contents

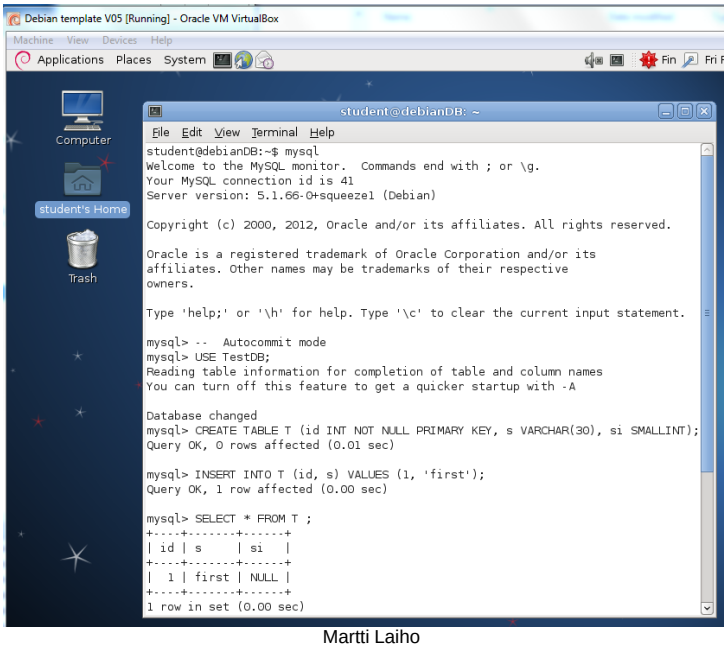
Part 1
- Database laboratory: DB2, Oracle, MySQL/InnoDB, PostgreSQL,...
- **Concepts**:
  - SQL-server, SQL-client, SQL-session
  - Client/Server dialogue: request, result, diagnostics
- **SQL transaction**
  - Autocommit mode, Implicit/explicit start of transaction
  - Commit: new consistent state, durability
  - Rollback: atomicity, transaction recovery
  - Consistency: constraints, diagnostics, exception handling
  - Diagnostics: SQLcode, SQLSTATE, ..
- **Single-user experiments**

Part 2
- **Concurrency problems**
- **ACID principle: isolation?**
- **Isolation levels**
- Concurrency Control Mechanisms: MGL, MVCC
- **Multi-user experiments**
- Some ”Best Practices”

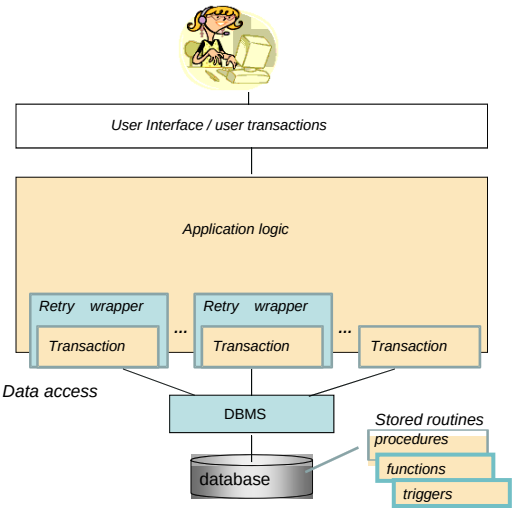
VirtualBox DebianDB



A sample MySQL test

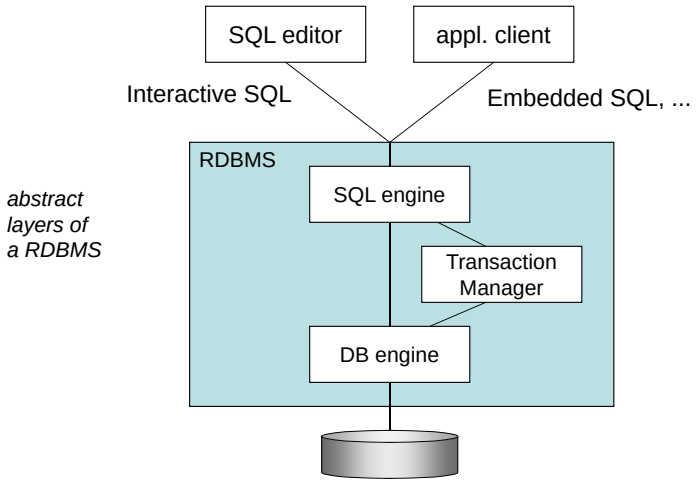


SQL Transactions in Reliable Applications



The abstract SQL API

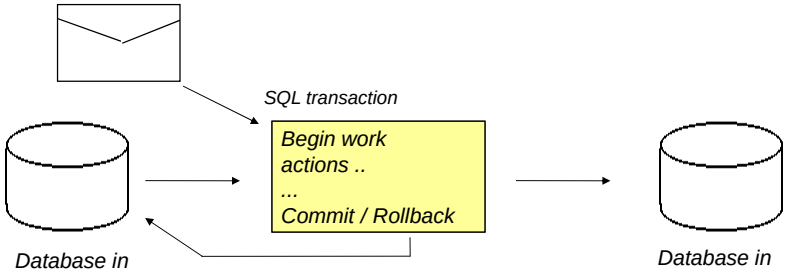
and the basic Client/Server architecture



SQL Transaction

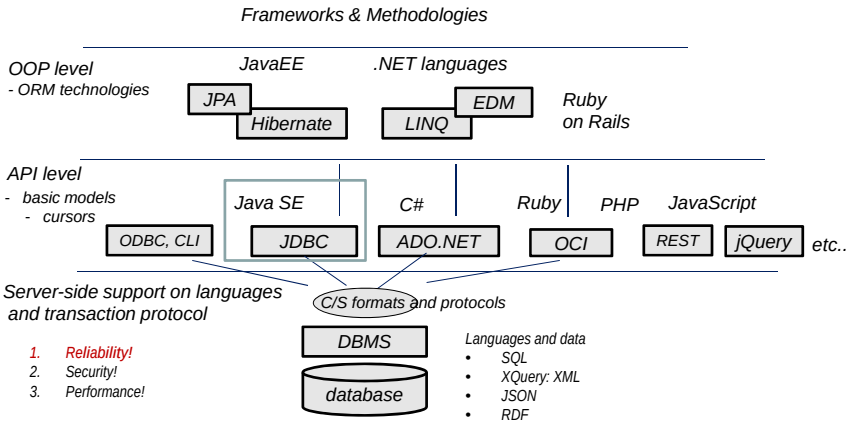
Context:

"Business transaction
=> Use case
=> User transaction => Sequence of SQL transactions

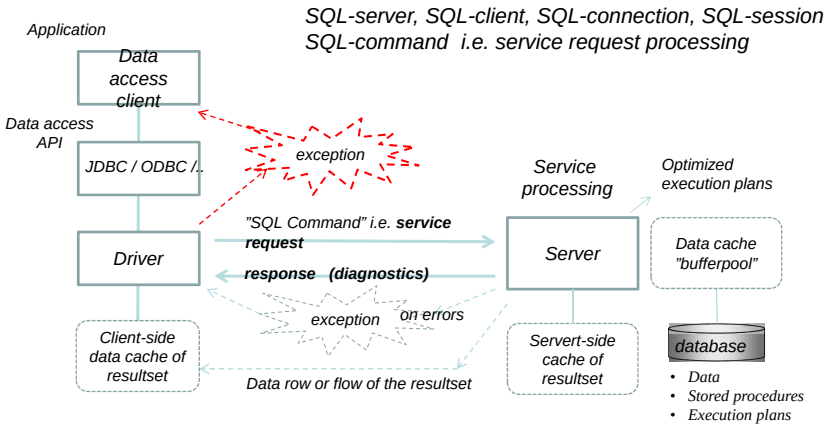


Is a Logical Unit of Work (LUW)
Ideally with the ACID properties
- Atomicity
- Consistency
- Isolation
- Durability

A Map on Data Access Technologies



Client / Server Dialogues



Client protocols:
- Shared Memory
- TCP/IP
- named pipes

Diagnostics: SQLcode, SQLSTATE

ISO SQL-89 SQLcode: *Integer:*

100 No data
0 successful execution
< 0 errors

ISO SQL-92 SQLSTATE: *String of 5 characters:*

	class	*subclass*	
Successful execution	0 0	0 0 0	
Warning	0 1	n n n	
No data	0 2	0 0 0	
. . .			
Transaction rollback	4 0	0 0 0	
		0 0 1	Serialization failure
		0 0 2	Integrity constraint violation
		0 0 3	Statement completion unknown
		0 0 4	Triggred action exception

etc - lots of standardized and implementation dependent codes

ISO SQL:1999 Get Diagnostics …

List of diagnostic items, including SQLSTATE and number of rows. Only few implementations this far

Structures for using Diagnostics

DB2 SQL:

```
<SQL statement>
IF (SQLSTATE <> '00000') THEN
     <error handling>
END IF;
```

Oracle PL/SQL:

```
BEGIN
     <processing>
EXCEPTION
WHEN <exception name> THEN
     <exception handling>;
...
WHEN OTHERS THEN
     err_code := sqlcode;
     err_text := sqlerrm;
     <exception handling>;
END;
```

compare with Java:

```
... throws SQLexception {
...
try {
     ...
     <JDBC statement(s)>
}
catch (SQLException ex) {
     <exception handling>
}
```

Transact-SQL of SQL Server:

```
BEGIN TRY
     <T-SQL statement(s)>
END TRY
BEGIN CATCH
     <exception handling based on
          ERROR_NUMBER(),
          ERROR_SEVERITY(),
          ERROR_STATE(),
          ERROR_PROCEDURE(),
          ERROR_LINE(),
          ERROR_MESSAGE()> ;
END CATCH;
```

ISO SQL: SET TRANSACTION

Source: Melton & Simon "SQL:1999"

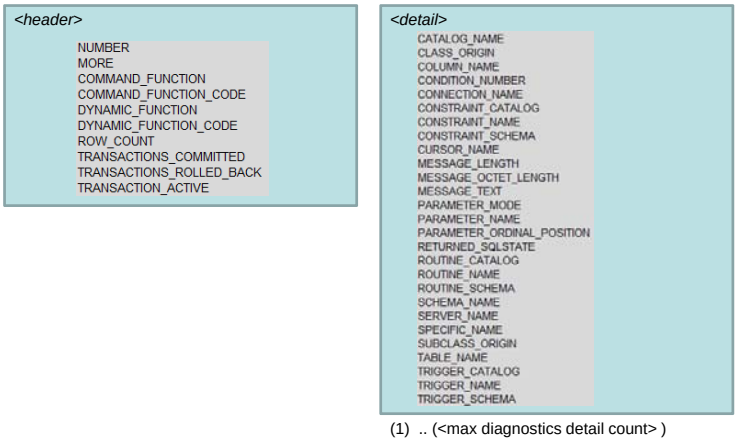
SET [LOCAL] TRANSACTION <mode>, …
<mode> ::= [READ ONLY | READ WRITE] |
[READ UNCOMMITTED |
READ COMMITTED |
REAPEATABLE READ |
SERIALIZABLE] |
[DIAGNOSTICS SIZE <integer>]

SET TRANSACTION tunes the attributes for following transaction.
It cannot be used in an active transaction.

*Diagnostics per SQL command consists of **header** and condition **details**.*
Diagnostics size defines for how many condition details per SQL command the server will reserve space in the diagnostics area in the transaction context.

xx

DIAGNOSTICS Items



(1) .. (<max diagnostics detail count>)

<SQL statement> ;
GET DIAGNOSTICS <target> = <item> [, . . .]
If SQLSTATE = . . .

SQL GET DIAGNOSTICS

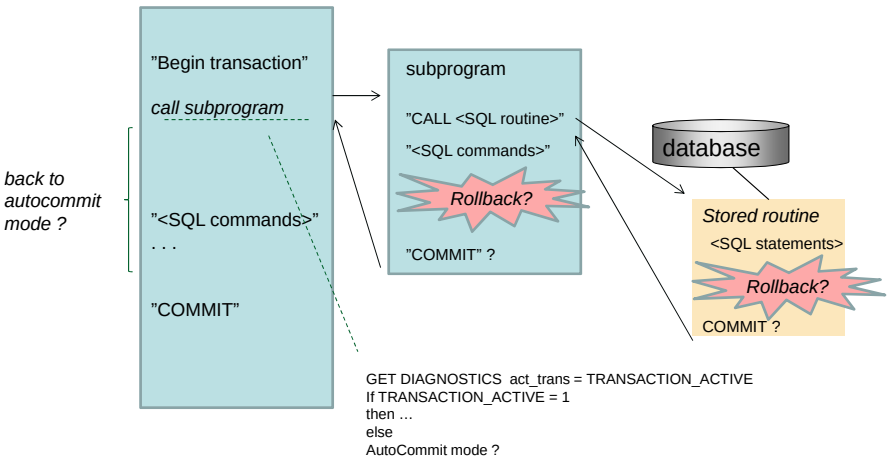
Example of getting diagnostics in MySQL 5.6:

```
INSERT INTO T (id, s) VALUES (2, NULL);
INSERT INTO T (id, s) VALUES (2, 'Hi, I am a duplicate');
mysql> INSERT INTO T (id, s) VALUES (2, 'Hi, I am a duplicate');
ERROR 1062 (23000): Duplicate entry '2' for key 'PRIMARY'
```

```
GET DIAGNOSTICS @rowcount = ROW_COUNT;
GET DIAGNOSTICS CONDITION 1
        @sqlstate = RETURNED_SQLSTATE,
        @sqlcode = MYSQL_ERRNO ;
SELECT @sqlstate, @sqlcode, @rowcount;
mysql> SELECT @sqlstate, @sqlcode, @rowcount;
+-----------+----------+-----------+
| @sqlstate | @sqlcode | @rowcount |
+-----------+----------+-----------+
| 23000     |     1062 |        -1 |
+-----------+----------+-----------+
1 row in set (0.00 sec)
```

xx

Potential errors



Diagnostics

	SQL-89	SQL-92	X/Open SC V2 1996	SQL/CLI	SQL:1999	SQL:2011	Mimer	DB2 LUW 9.5 /	Oracle 11g1	SQL Server 2012	MySQL 5.6.4	PostgreSQL 8.4	Pyrrho 4.8	Rdb7 7.1
SQLCA			Y					ESQL	ESQL	ESQL/C		ECPG		
SQLCODE	Y	Y	Y	Y				Y	Y	@@error error_number()				
SQLSTATE		Y	Y	Y	Y	Y		Y	Y	error_state()			Y	
FOUND												y		
GET DIAGNOSTICS	no	no	Y		Y	Y		Y	no	no	Y	Y	Y	Y
<statement info> i.e. diag. Header														
<target>=<st.info item>[, ...]														
<satement information item name>														
NUMBER				Y	Y	Y	Y				Y			
MORE				Y	Y	Y	Y							
COMMAND_FUNCTION					Y	Y	Y						Y	
COMMAND_FUNCTION_CODE					Y	Y							Y	
DYNAMIC_FUNCTION					Y	Y	Y							
DYNAMIC_FUNCTION_CODE					Y	Y								
ROW_COUNT				Y	Y	Y	Y	Y		@@rowcount	Y	Y	Y	Y
TRANSACTIONS_COMMITTED				Y	Y	Y							Y	Y
TRANSACTIONS_ROLLED_BACK				Y	Y	Y							Y	Y
TRANSACTION_ACTIVE				Y	Y	Y	Y							Y
product extensions:														
ACCESS_MODE														Y
CALLING_ROUTINE														Y
CONNECTION_NAME														Y
CURRENT_ROW														Y
GLOBAL_TRANSACTION														Y
ISOLATION_LEVEL														Y
DB2_RETURN_STATUS								Y						
DB2_SQL_NESTING_LEVEL								Y						
RESULT_OID												Y		
SQL/CLI extensions:														
RETURNCODE				Y										
<condition info> i.e. detail(s)														
EXCEPTION			Y		Y	no	Y	Y			no	no	no	Y
CONDITION			no		no	Y		no			Y	no	no	
<condition nr> <target>=<c.i.item> [, ...]														
<condition information item name>														
CATALOG_NAME			Y	Y	Y	Y	Y				Y		Y	
CLASS_ORIGIN			Y	Y	Y	Y	Y				Y		Y	
COLUMN_NAME			Y	Y	Y	Y	Y				Y			
CONDITION_NUMBER			Y	Y	Y	Y	Y							

Condition Handlers

In stored routines

DECLARE <condition name> CONDITION
[FOR SQLSTATE <value>] ;

DECLARE { **CONTINUE | EXIT | UNDO** }
HANDLER FOR
{ **SQLSTATE <value>**
| <condition name>
| SQLEXCEPTION
| SQLWARNING
| NOT FOUND
}
{ **<SQL statement> | <compoud statement>** } ;

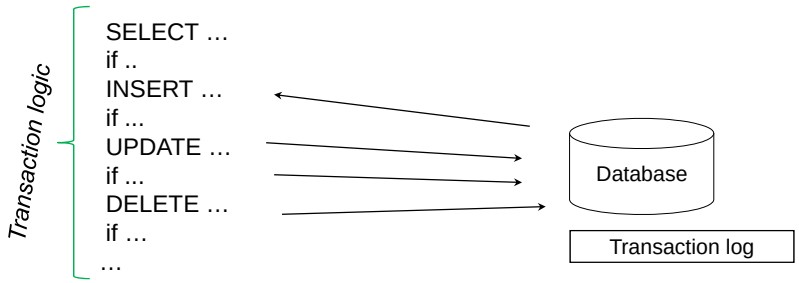
SIGNAL {**<condition name> | SQLSTATE <value>**} ;

.. Condition Handlers

```
CREATE PROCEDURE BankTransfer (IN fromAcct INT,
                               IN toAcct INT,
                               IN amount INT,
                               OUT msg VARCHAR(100))
LANGUAGE SQL MODIFIES SQL
P1: BEGIN
    DECLARE EXIT HANDLER FOR NOT FOUND  BEGIN ROLLBACK;
            SET msg = CONCAT('missing account ', fromAcct); END;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION   BEGIN ROLLBACK;
            SET msg = CONCAT('negative balance (?) in ', fromAcct); END;
    UPDATE Accounts SET balance = balance - amount WHERE acctID = fromAcct;
--
    DECLARE EXIT HANDLER FOR NOT FOUND   BEGIN ROLLBACK;
             SET msg = CONCAT('missing account ', toAcct); END;
    UPDATE Accounts SET balance = balance + amount WHERE acctID = toAcct;
    COMMIT;
    SET msg = 'committed';
END P1
```


SQL Transaction

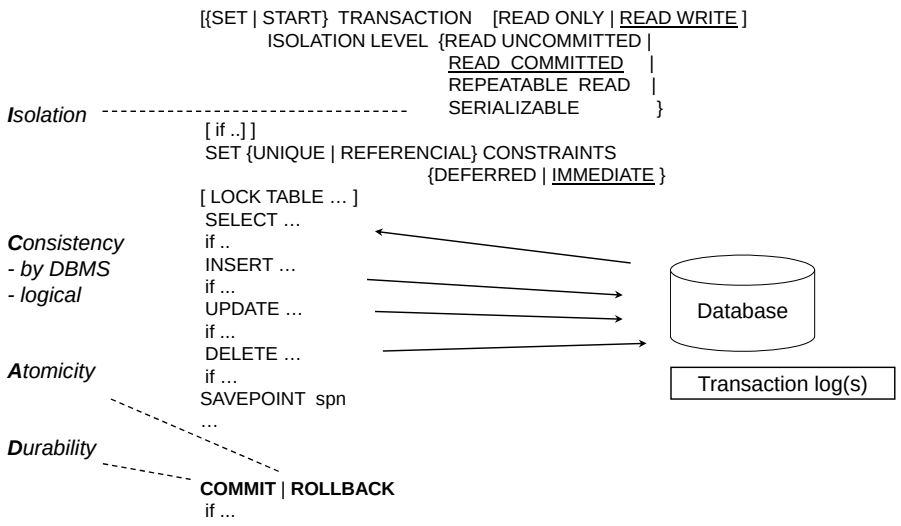
<implicit start> or <explicit start>



COMMIT | ROLLBACK

<explicit start> ::= BEGIN WORK
| BEGIN TRANSACTION
| START TRANSACTION

ACID SQL transaction

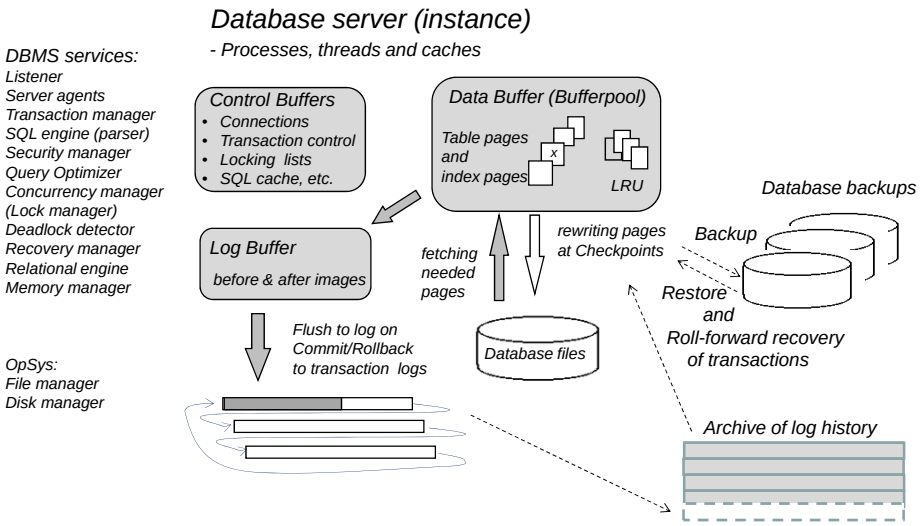


ROLLBACK

- i.e. automatic transaction recovery is based on use of transaction history which saves addresses and ”before images” of all changed / deleted rows
- For inserted rows the ”before image” is empty
- In ROLLBACK operation the server simply restores the before images of all rows affected by the transaction back to the original addresses

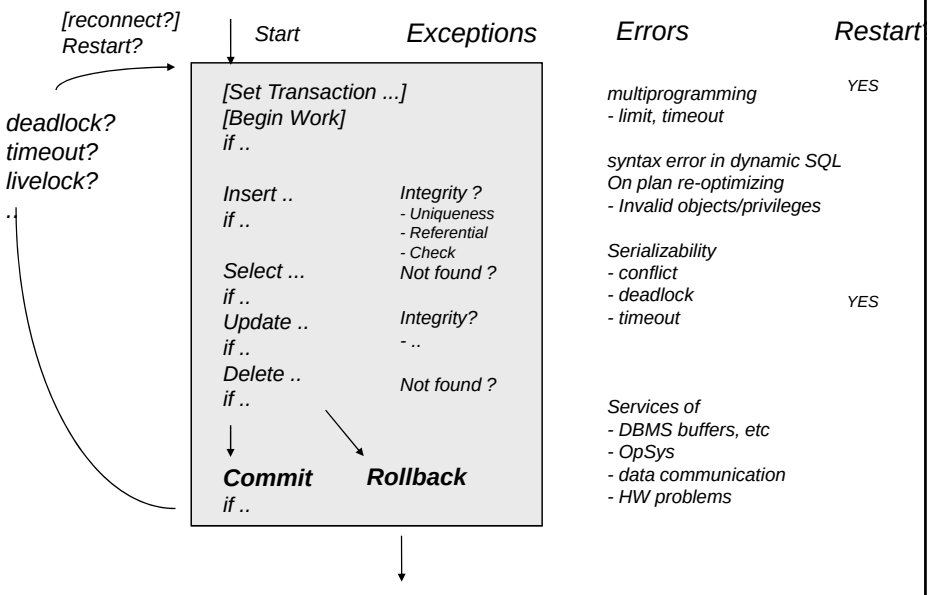
- For more details, see the presentation ”Basics of SQL Transactions”

A generic overview of a database server



Martti Laiho

Diagnostics needed after every SQL command



Martti Laiho 1998-2009

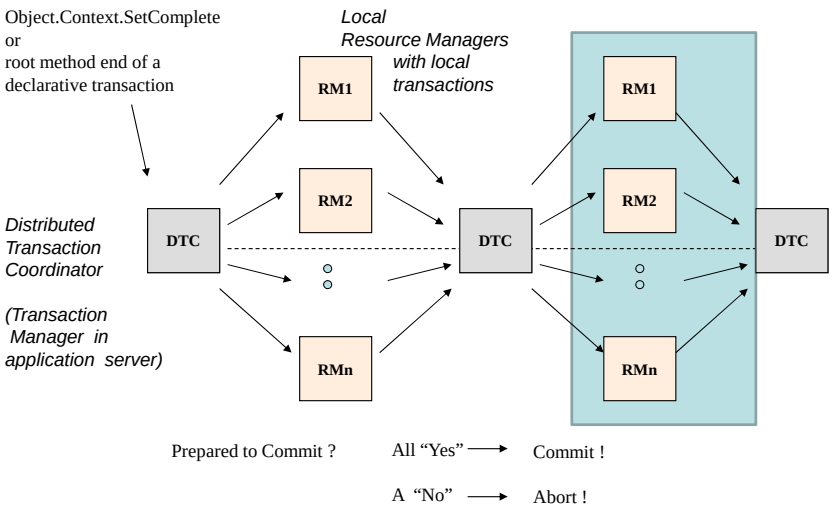
SQL Transaction Models

- **Flat transaction**
  - ACID properties
    - Atomicity (all or nothing!)
    - Consistency (integrity constraints)
    - Isolation (based on MGLCC, 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

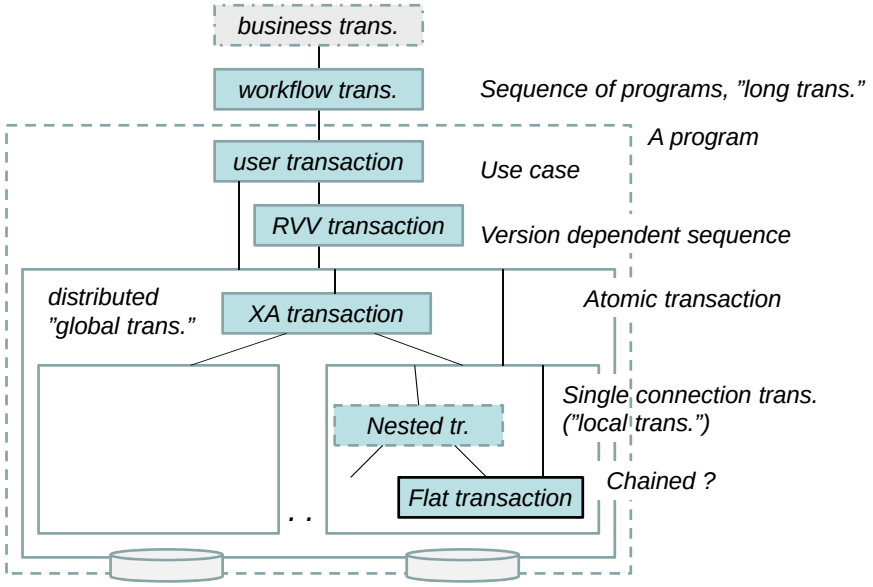
- Chained transactions

Distributed Transaction Two-Phase Commit (2PC)

(XA transaction)



Hierarchy of Transaction Concepts



..Differently behaving products

- As default in AUTOCOMMIT mode ?
- Implicit or explicit starts of transactions
- Implicit COMMIT on DDL ?
- Default isolation
- What is considered as error or Warning ?
  - Value truncation, value overflow, …
- Error in command
  - Rolls back the command, compound command
  - Rolls back the command and discards commands until end of transaction
  - Rolls back the transaction
- Concurrency control mechanisms
- Concurrency conflict resolutions: automatic ROLLBACK ?

ISO/SQL xacts and product implementations

	ANSI/ISO SQL: 2006	DB2 LUW 9.7	Oracle 12.1	SQL SERVER 2012	MySQL/InnoDB 5.6	PostgreSQL 9.2	Pyrrho 4.8
autocommit (server-side)	n/a	n/a	n/a	yes	yes	yes	yes
Transaction Limits							
explicit start	yes	n/a	n/a	yes	yes	yes	yes
implicit start	yes	yes	yes	(configurable)	(configurable)	n/a	n/a
COMMIT	yes	yes	yes	yes	yes	yes	yes
implicit commit on DDL	n/a	n/a	yes	n/a	yes	n/a	n/a
ROLLBACK	yes	yes	yes	yes	yes	yes	yes
implicit rollback on concurrency conflict (deadlock)	(yes)	yes	no (exception raised)	yes	yes	no (xaction invalidated)	yes, at commit
implicit rollback on error	left open	n/a	n/a	(configurable)	n/a	no (xaction invalidated)	yes
SAVEPOINT	yes	yes	yes	yes	yes	yes	n/a
ROLLBACK TO SAVEPOINT	yes	yes	yes	yes	yes	yes	n/a
RELEASE SAVEPOINT	yes	yes	yes	n/a	yes	yes	n/a
Isolation levels							
READ UNCOMMITTED	yes	UR	n/a	yes	yes	n/a migrate to "read latest committed"	n/a
"read latest committed"	n/a	CS (currently committed)	"read committed"	(configurable)	"read committed"	"read committed"	n/a
READ COMMITTED	yes	CS	n/a	yes	n/a	n/a migrate to snapshot	n/a
REPEATABLE READ	yes	RS	n/a	yes	n/a	n/a migrate to snapshot	n/a
snapshot	n/a	n/a	"serializable"	(configurable)	"repeatable read"	"serializable"	"serializable"
SERIALIZABLE	yes	RR	explicit locking	yes	yes	explicit locking	"serializable"

Single-user Transaction Experiments

- Students start their private copies of DebianDB
- Teacher demonstrates the first steps
  making sure that all students can repeat every step
  getting started with the experiment
- The same DBMS product is selected to be studied,
  - for example MySQL/InnoDB
- A single SQL session is started in a terminal window
- Students make notes of the transaction experiments or experiences are discussed

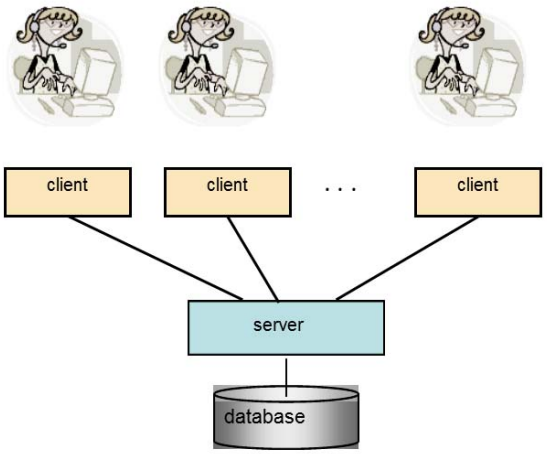
Experiments with help of the instructor

Exercises in the SQL Transactions Handbook

- 1.1 Autocommit mode / Transactional mode: Rollback
- 1.2 Start transaction ?
- 1.3 Autocommit off => Transactional mode
- 1.4 Commit on DDL: Create, Alter, Drop ?
- 1.5 Errors / warnings, Rollback on error ?
  Diagnostics ?
- 1.6 Check constraint, Bank transfer transactions
- 1.7 Unit of Recovery
  - Transaction recovery Database recovery
    up to the latest committed transaction

Part 2:

Competing Transactions in Multi-user Environment



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 (MGL, LSCC) 0% ..100%
  - Multi-Versioning (MVCC) %?
  - Cursor level concurrency control,
    SELECT .. FOR UPDATE
- Client-side concurrency control
  - Row Version Verification (RVV) aka. "Optimistic Locking"

Concurrency Problems

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

1 Lost Update Problem (solved?)

2 Uncommitted Dependency Problem (Dirty Read)

3 Inconsistent Analysis Problems

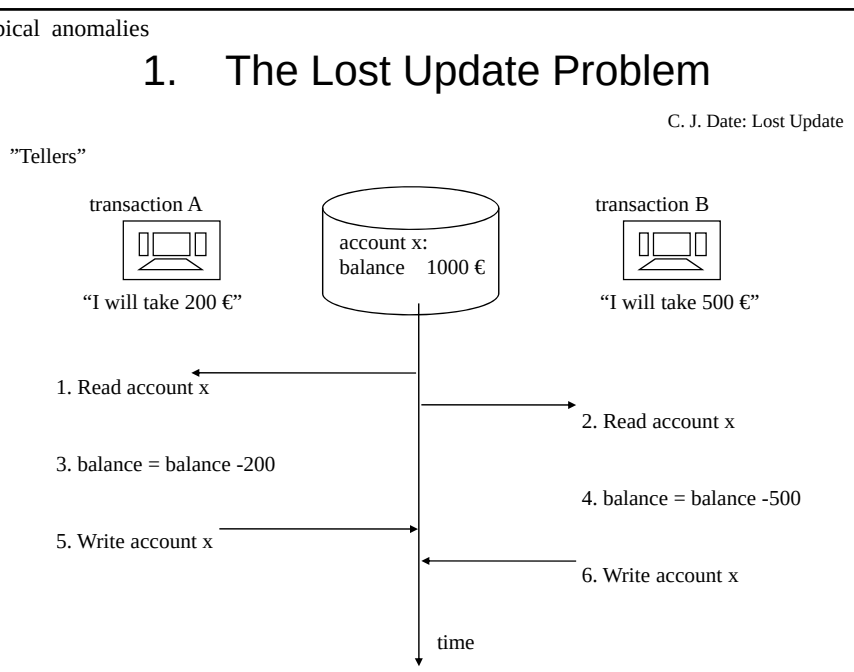
a) Decreasing Read Set (Non-repeatable Read)

b) Increasing Read Set (Phantoms)

..Typical anomalies

1. The Lost Update Problem

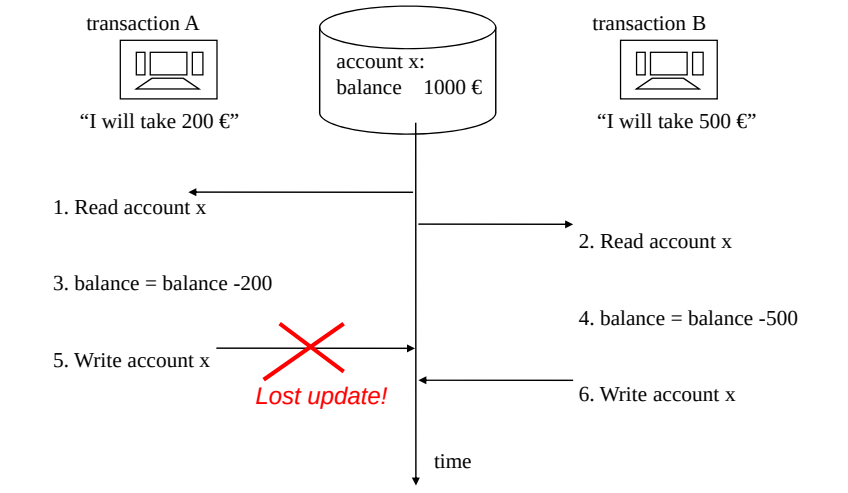
C. J. Date: Lost Update



..Typical anomalies

1. The Lost Update Problem

C. J. Date: Lost Update



Concurrency Control by S- and X-locks

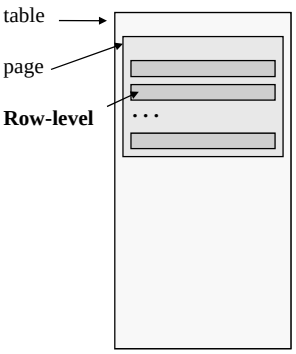
Compatibility of S and X locks:

Lock of transaction A to object o

Lock request of transaction B to object o	Shared	eXclusive
Shared	**Grant**	**Wait !**
eXclusive	**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:



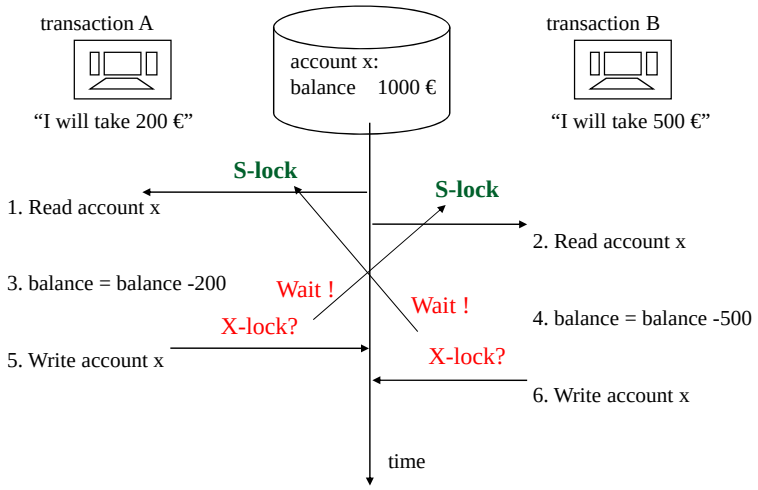
..Typical anomalies

1. The Lost Update Problem

- Applying the locking scheme:

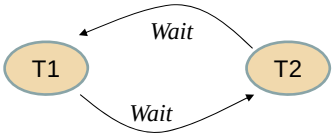
C. J. Date: Lost Update

”Tellers”



Deadlock

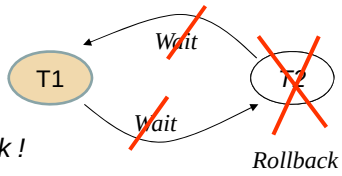
A Cycle of Lock Waits



Modern DBMS systems will detect the deadlock in some seconds (deadlock detection) and solve the waiting cycle

- *selecting the victim*
- *making automatic Rollback (not Oracle)*
- *send error message to the application*

=> *Application must react on the deadlock !*



Martti Laiho

Martti Laiho 1998

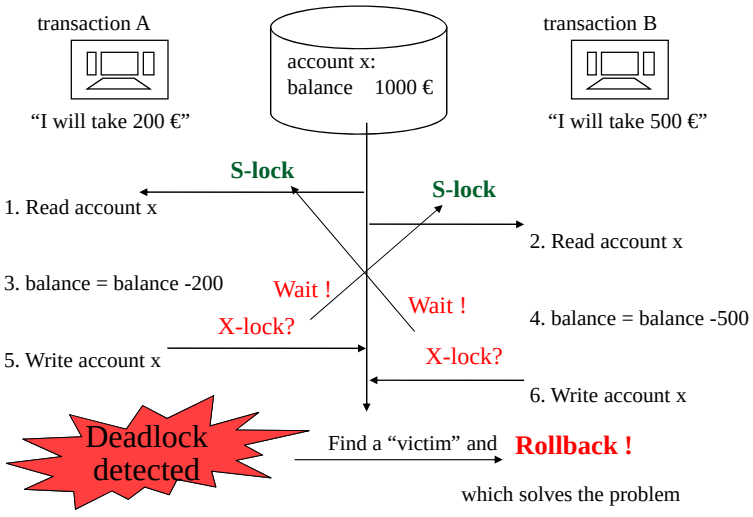
..Typical anomalies

1. The Lost Update Problem

- Applying the locking scheme:

C. J. Date: Lost Update

”Tellers”



Martti Laiho 46

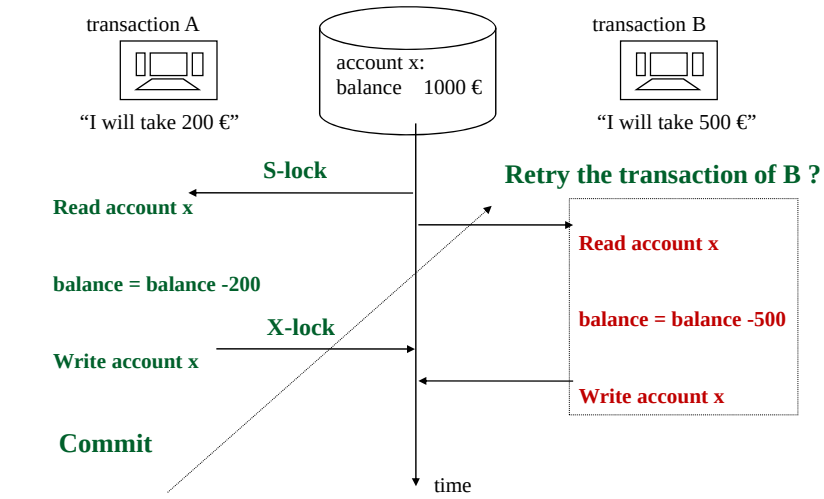
..Typical anomalies

1. The Lost Update Problem

- solved by locking scheme:

C. J. Date: Lost Update

”Tellers”



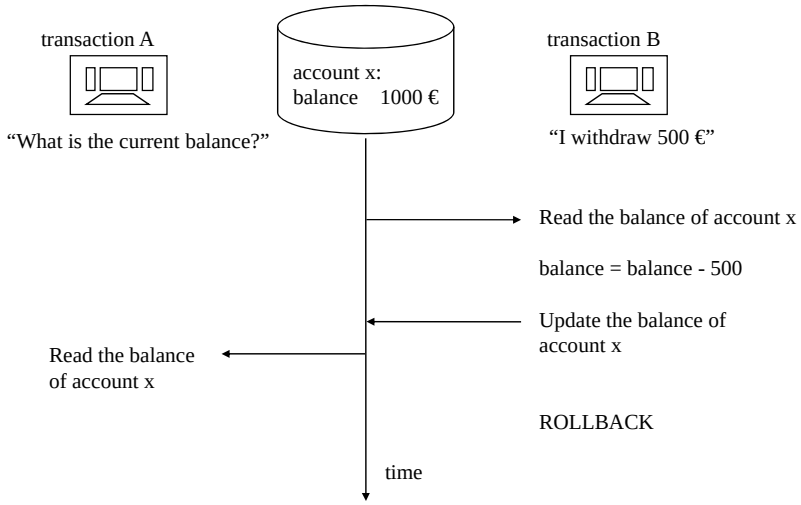
Martti Laiho

Martti Laiho 1998 47

..Typical anomalies

Dirty Read

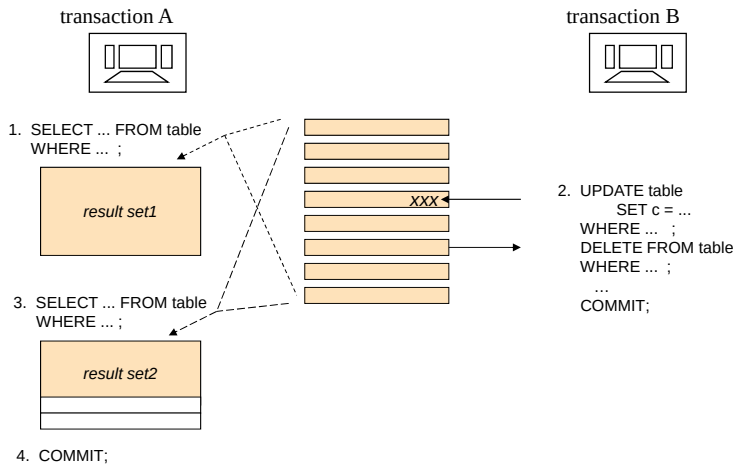
C.J. Date



Martti Laiho 48

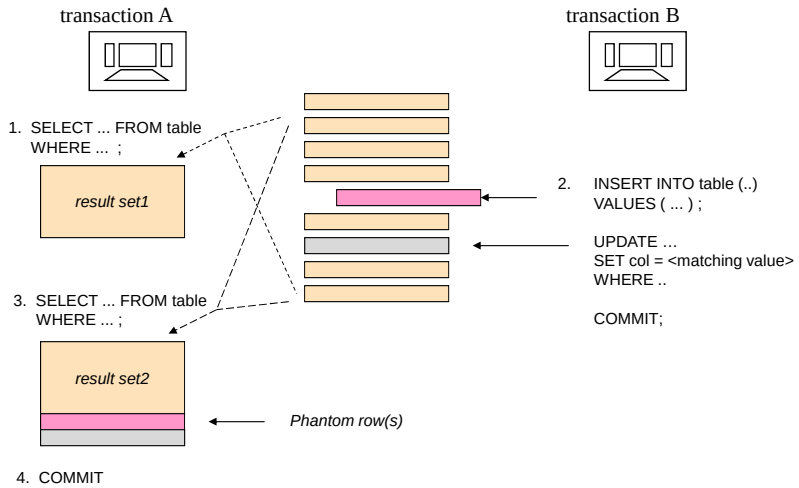
Non-Repeatable Read

C.J. Date



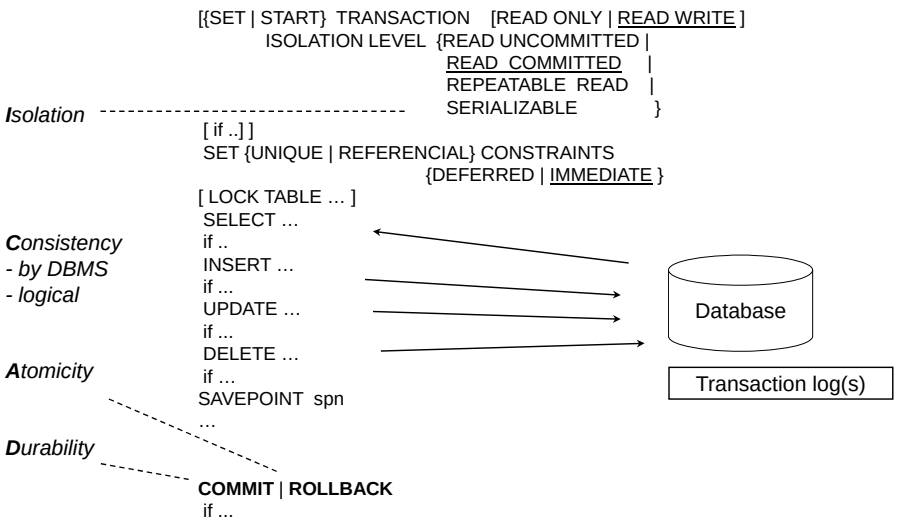
Martti Laiho

Phantom Read



Martti Laiho

ACID SQL transaction



Isolation Levels of ISO SQL

Isolation Level: \ Anomalies:	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

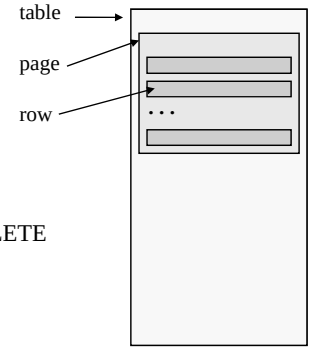
Isolation levels can be explained by objects and duration in S-locking
preventing only the transaction itself against certain anomalies,
but can't prevent concurrent transactions from dirty reads, etc
i.e. can't provide strict isolation as defined by Haerder and Reuter

Locking Scheme Concurrency Control (LSCC)

Compatibility of S and X locks

Lock request of transaction B to object o	Lock of transaction A to object o: **S**hared	eX**X**clusive
Shared	**Grant**	**Wait !**
eX**X**clusive	**Wait !**	**Wait !**

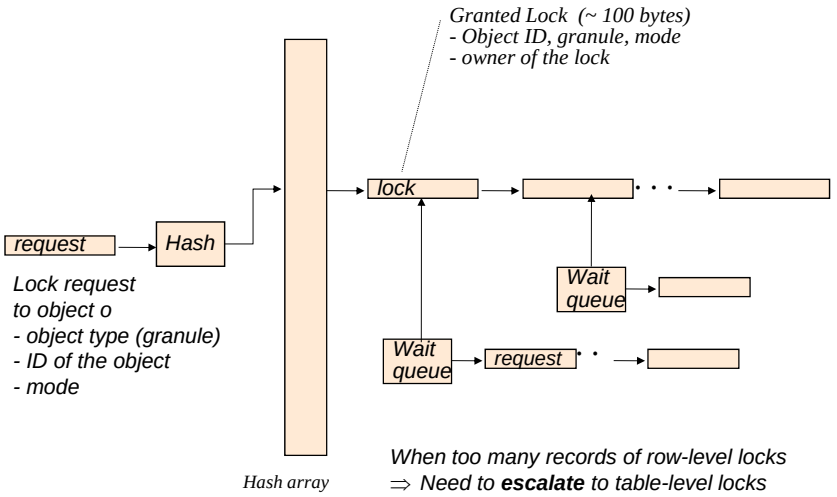
Locking granularity:



Implicit locking by DBMS on INSERT, UPDATE, DELETE
- 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**

Explicit locking:
Table-level: LOCK TABLE <table> IN <locking mode>
Row-level: SELECT .. FROM <table> WHERE <search condition> **FOR UPDATE**

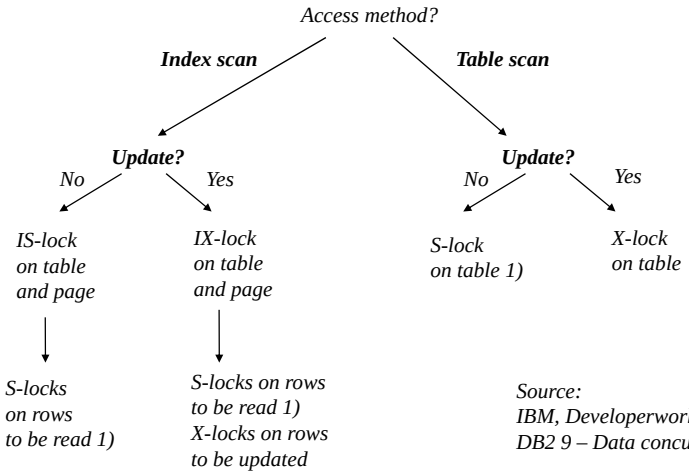
Management of Lock Records and Requests



When too many records of row-level locks
*⇒ Need to **escalate** to table-level locks*
=> Less work, longer waiting / deadlocking

*Lock wainting time can be controlled by **TIMEOUT**s*

Locking Mode selected by the Optimizer



Source:
IBM, Developerworks, Sanders
DB2 9 – Data concurrency

1) depending on the isolation level

Martti Laiho 2007

Multi-Granular Locking (MGL) scheme

- Sample variants of lock compatibility matrices

Lock granules:

database

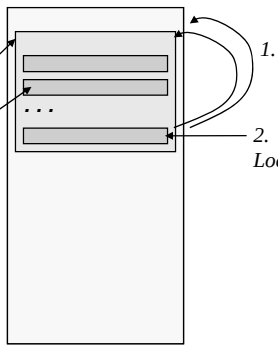
(tablespace)

table

(extent)

page

row



1. Intent locks
IS for S on row
IX for X on row

2.
Lock on row

Lock requested:	Lock already granted to some other process				
	IS	IX	S	SIX	X
IS	grant	grant	grant	grant	wait
IX	grant	grant	wait	wait	wait
S	grant	wait	grant	wait	wait
SIX	grant	wait	wait	wait	wait
X	wait	wait	wait	wait	wait

SIX = S + IX

Lock requested:	Lock already granted to some other process			
	none	S	U	X
S	grant	grant	grant[3]	wait
U	grant	grant	wait	wait
X	grant	wait	wait	wait

***S**hared locks (S) allow reading.*
*e**X**clusive locks (X) allow writing and are kept up to end of transaction eliminating lost updates.*

Other locks on index ranges, schemas

either GRANT or CNVT

Compatibility Matrix of SQL Server Locks

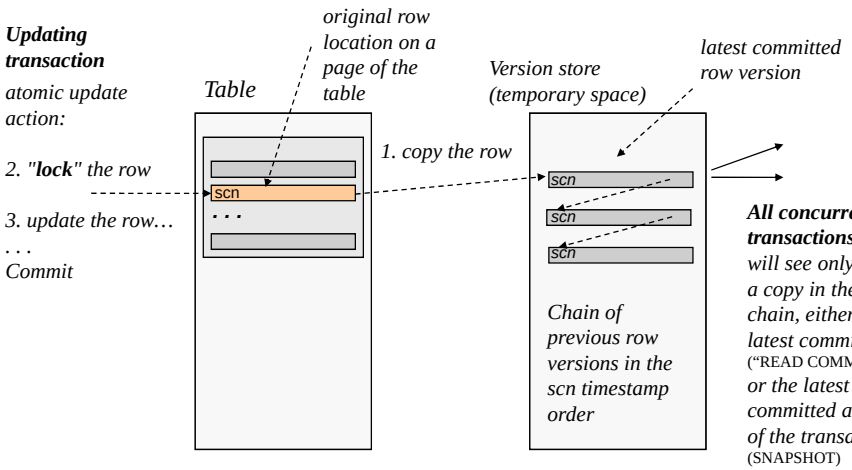
	NL	SCH-S	SCH-M	S	U	X	IS	IU	IX	SIU	SIX	UIX	BU	RS-S	RS-U	RI-N	RI-S	RI-U	RI-X	RX-S	RX-U	RX-X
NL	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
SCH-S	N	N	C	N	N	N	N	N	N	N	N	N	N	I	I	I	I	I	I	I	I	I
SCH-M	N	C	C	C	C	C	C	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
S	N	N	C	N	N	C	N	N	C	N	C	C	C	N	N	N	N	N	C	N	N	C
U	N	N	C	N	C	C	N	C	C	C	C	C	C	N	C	N	N	C	C	N	C	C
X	N	N	C	C	C	C	C	C	C	C	C	C	C	C	C	N	C	C	C	C	C	C
IS	N	N	C	N	N	C	N	N	N	N	N	N	C	I	I	I	I	I	I	I	I	I
IU	N	N	C	N	C	C	N	N	N	N	N	C	C	I	I	I	I	I	I	I	I	I
IX	N	N	C	C	C	C	N	N	N	C	C	C	C	I	I	I	I	I	I	I	I	I
SIU	N	N	C	N	C	C	N	N	C	N	C	C	C	I	I	I	I	I	I	I	I	I
SIX	N	N	C	C	C	C	N	N	C	C	C	C	C	I	I	I	I	I	I	I	I	I
UIX	N	N	C	C	C	C	N	C	C	C	C	C	C	I	I	I	I	I	I	I	I	I
BU	N	N	C	C	C	C	C	C	C	C	C	C	N	I	I	I	I	I	I	I	I	I
RS-S	N	I	I	N	N	C	I	I	I	I	I	I	I	N	N	C	C	C	C	C	C	C
RS-U	N	I	I	N	C	C	I	I	I	I	I	I	I	N	C	C	C	C	C	C	C	C
RI-N	N	I	I	N	N	N	I	I	I	I	I	I	I	C	C	N	N	N	N	C	C	C
RI-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	N	N	N	C	C	C	C
RI-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	N	N	C	C	C	C	C
RI-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	N	C	C	C	C	C	C
RX-S	N	I	I	N	N	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-U	N	I	I	N	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C
RX-X	N	I	I	C	C	C	I	I	I	I	I	I	I	C	C	C	C	C	C	C	C	C

Key

N	No Conflict	SIU	Share with Intent Update
I	Illegal	SIX	Shared with Intent Exclusive
C	Conflict	UIX	Update with Intent Exclusive
		BU	Bulk Update
NL	No Lock	RS-S	Shared Range-Shared
SCH-S	Schema Stability Locks	RS-U	Shared Range-Update
SCH-M	Schema Modification Locks	RI-N	Insert Range-Null
S	Shared	RI-S	Insert Range-Shared
U	Update	RI-U	Insert Range-Update
X	Exclusive	RI-X	Insert Range-Exclusive
IS	Intent Shared	RX-S	Exclusive Range-Shared
IU	Intent Update	RX-U	Exclusive Range-Update
IX	Intent Exclusive	RX-X	Exclusive Range-Exclusive

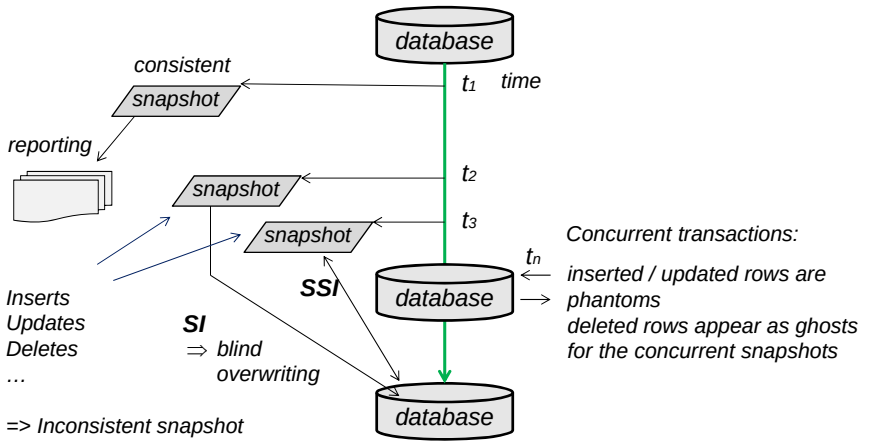
For more information see:
SQL Server Books Online

Multi-Version Concurrency Control (MVCC)



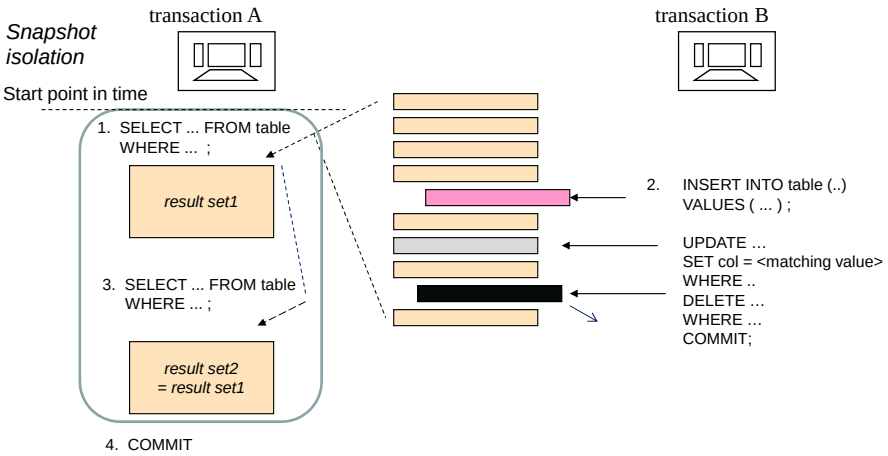
Locking stamp *on the row record:*
Oracle: system change number (scn)
SQL Server: transaction sequence number (xsn) - see K Delaney's eBook
InnoDB: ?
PostgreSQL: ?

Phantoms & ghosts in snapshot isolation (SI)

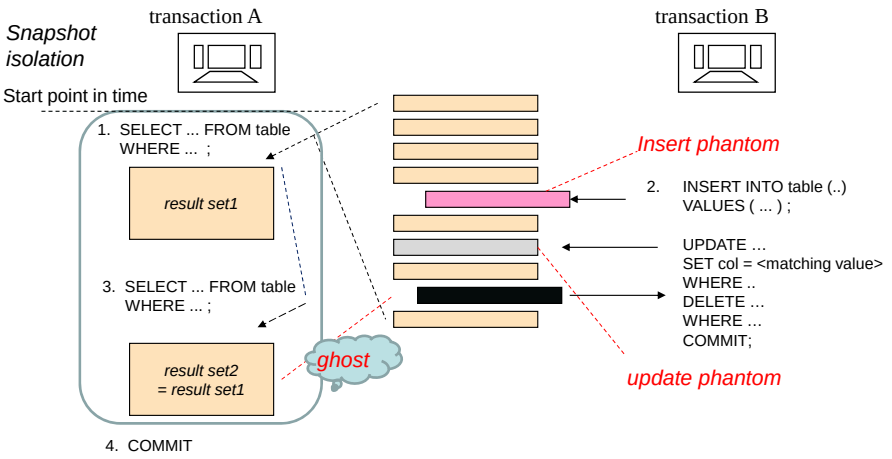


SI = snapshot isolation
SSI = ”serializable” snapshot isolation (using version verification)

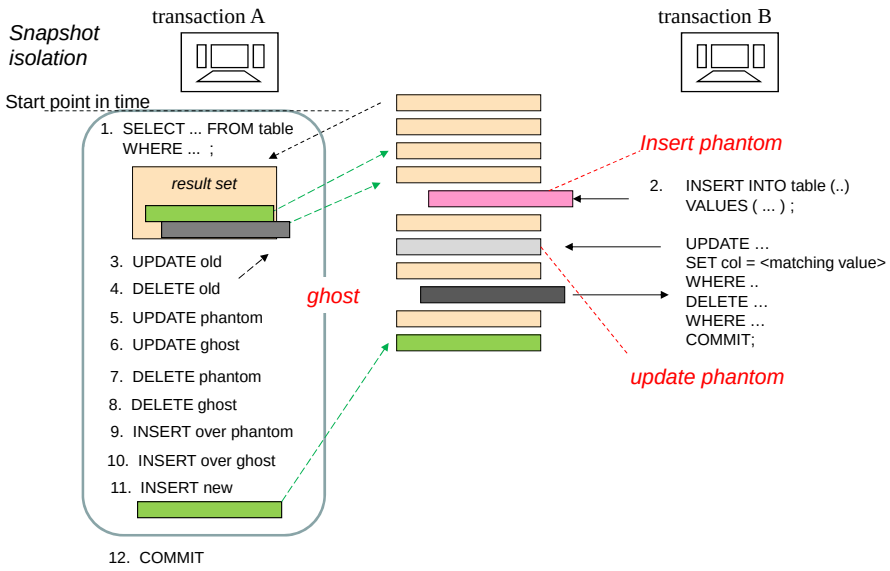
Snapshot (start point in time)



Phantoms and Ghosts of Snapshot



Inconsistencies of Snapshot

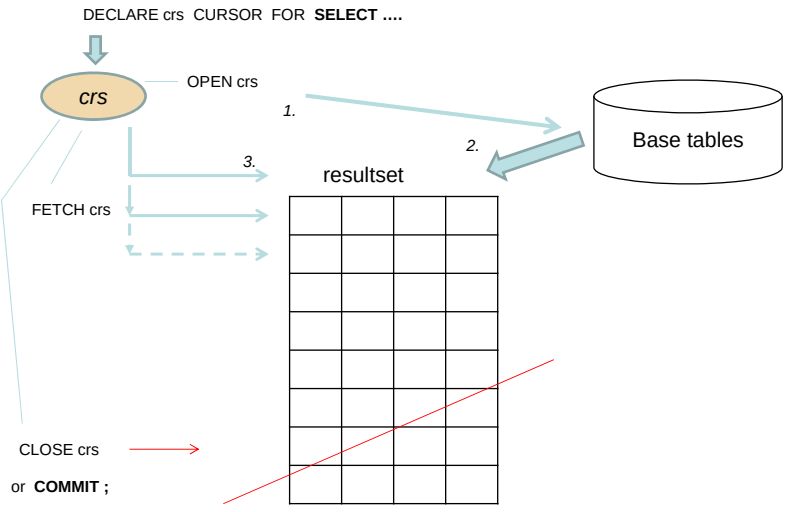


Cursor Processing

- 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)

..Cursor Processing



Multi-user Transaction Experiments

- Students start their private copies of DebianDB
- Teacher demonstrates the first steps
  making sure that all students can repeat every step
  getting started with the experiment
- The same DBMS product is selected to be studied,
  - for example MySQL/InnoDB
- Two concurrent SQL sessions are started in separate terminal windows
- Students make notes of the transaction experiments or experiences are discussed

Experiments on concurrency

Exercises in the SQL Transactions Handbook

- Exercise 2.1 Lost Update problem
- Exercise 2.2 SELECT-UPDATE scenarios a) and b)
- Exercise 2.3 UPDATE – UPDATE scenarios in opposite order
  => deadlock
- Exercise 2.4 Dirty Read problem
- Exercise 2.5 Non-Repeatable Read
- Exercise 2.6 Insert-Phantom Problem
- Exercise 2.7 A SNAPSHOT study with different kinds of Phantoms

A Well-designed SQL Transaction

- Is an atomic, logical unit of work that either transfers the database from a consistent state to another consistent state – or all its actions need to be rolled back
- Is a short dialogue with the database server performing data retrieval and/or data update task of some use case
- Does not contain any user intervention during the transaction
- Checks carefully diagnostics of the received data access services
- Handles the generated data access exceptions
- May contain transaction logic which depends on the received data or diagnostics
- Depending on the logic is restarted on concurrency or connection failures, but avoiding livelocks

Evaluation

- Students fill the DBTech VET lab evaluation survey
- Students fill the Multi-choice questions
- Teacher fills the DBTech VET Lab summary and returns it to the local(?) coordinator