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.

Areas in Database Technologies
Context of our project

Europe 2020 strategy:

- Labor Markets and Industry
- VET Vocational education and training
- HE Higher education
  - Vocational UASs and
  - Science univ.

Problems and need for Transactions

- Today society, infrastructures, 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, improper 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.
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?

SQL Standard

Serializability Theory
Relational Theory

Application developers ??

- "But we don't play by the books"

Application

DB2 SQL Server Oracle MySQL PostgreSQL Pytho

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

Martti Laiho 7
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

• Concurrency problems

• ACID principle: isolation?

• Isolation levels

• Concurrency Control Mechanisms: MGL, MVCC

• Multi-user experiments

• Some "Best Practices"

Part 2

VirtualBox DebianDB
A sample MySQL test

SQL Transactions in Reliable Applications

User Interface / user transactions

Application logic

Retry wrapper

Transaction

Database

Stored routines

procedures

functions

triggers
The abstract SQL API

and the basic Client/Server architecture

SQL editor
appl. client

Interactive SQL
Embedded SQL, ...

abstract layers of a RDBMS

RDBMS

SQL engine
Transaction Manager
DB engine

SQL Transaction

Context:
*Business transaction
=> Use case
=> User transaction => Sequence of SQL transactions

Begin work actions ...
... Commit / Rollback

Database in consistent state

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

Database in consistent state
A Map on Data Access Technologies

Frameworks & Methodologies
- OOP level
  - ORM technologies
    - JPA
    - Hibernate
    - EDM
    - Ruby
- API level
  - basic models
    - cursors
  - SQL
  - XQuery: XML
  - JSON
  - RDF

Languages and data
- SQL
- XQuery: XML
- JSON
- RDF

Server-side support on languages and transaction protocol
1. Reliability
2. Security
3. Performance

Frameworks & Methodologies
- .NET languages
  - ADO.NET
  - C#
  - LINQ
- Java SE
- C#
- Ruby
- PHP
- JavaScript
- etc.

Client / Server Dialogues
- SQL-server, SQL-client, SQL-connection, SQL-session
- SQL-command i.e. service request processing

Optimized execution plans
- Data
- Stored procedures
- Execution plans

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:

<table>
<thead>
<tr>
<th>Class</th>
<th>Subclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0</td>
<td>Warning</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>No data</td>
</tr>
<tr>
<td>...</td>
<td>Transaction rollback</td>
</tr>
<tr>
<td>0 0 1 0 0</td>
<td>Serialization failure</td>
</tr>
<tr>
<td>0 0 2 0 0</td>
<td>Integrity constraint violation</td>
</tr>
<tr>
<td>0 0 3 0 0</td>
<td>Statement completion unknown</td>
</tr>
<tr>
<td>0 0 4 0 0</td>
<td>Triggered action exception</td>
</tr>
<tr>
<td>etc - lots of standardized and implementation dependent codes</td>
<td></td>
</tr>
</tbody>
</table>

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

Oracle PL/SQL:

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

Transact-SQL of SQL Server:

```sql
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

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.

DIAGNOSTICS Items

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

Example of getting diagnostics in MySQL 5.6:

```sql
INSERT INTO T (id, s) VALUES (2, NULL);
INSERT INTO T (id, s) VALUES (2, 'Hi, I am a duplicate');
```

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

```sql
GET DIAGNOSTICS @rowcount = ROW_COUNT;
GET DIAGNOSTICS CONDITION 1
    @sqlstate = RETURNED_SQLSTATE,
    @sqlcode = MYSQL_ERRNO;
SELECT @sqlstate, @sqlcode, @rowcount;
```

```sql
mysql> SELECT @sqlstate, @sqlcode, @rowcount;
+-----------+----------+-----------+
<table>
<thead>
<tr>
<th>@sqlstate</th>
<th>@sqlcode</th>
<th>@rowcount</th>
</tr>
</thead>
<tbody>
<tr>
<td>23000</td>
<td>1062</td>
<td>-1</td>
</tr>
</tbody>
</table>
+-----------+----------+-----------+
1 row in set (0.00 sec)
```

---

**Potential errors**

- **“Begin transaction”**
- **call subprogram**
- **“<SQL commands>”**
- **“COMMIT”**

- Rollback?
- COMMIT?

GET DIAGNOSTICS act_trans = TRANSACTION_ACTIVE
IF TRANSACTION_ACTIVE = 1
then ...
else
AutoCommit mode?

---

Martti Laiho 22
Diagnostics

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

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

Condition Handlers

In stored routines

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

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

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

Transaction logic
- SELECT ...
  if ...
- INSERT ...
  if ...
- UPDATE ...
  if ...
- DELETE ...
  if ...
- ...

COMMIT | ROLLBACK

<implicit start> or <explicit start>

<explicit start> ::= BEGIN WORK
| BEGIN TRANSACTION
| START TRANSACTION
ACID SQL transaction

\[
\begin{align*}
&\text{([SET | START] TRANSACTION [READ ONLY | READ WRITE] ISOLATION LEVEL [READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE])} \\
&\text{[if ...]} \\
&\text{SET [UNIQUE | REFERENTIAL] CONSTRAINTS [DEFERRED | IMMEDIATE]} \\
&\text{[ LOCK TABLE ... ]} \\
&\text{SELECT ...} \quad \text{if ...} \\
&\text{INSERT ...} \quad \text{if ...} \\
&\text{UPDATE ...} \quad \text{if ...} \\
&\text{DELETE ...} \quad \text{if ...} \\
&\text{SAVEPOINT spn} \\
&\text{COMMIT | ROLLBACK if ...}
\end{align*}
\]

Database

Transaction log(s)

**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

**Database server (instance)**
- Processes, threads and caches

**Control Buffers**
- Connections
- Transaction control
- Locking lists
- SQL cache, etc.

**Log Buffer**
before & after images

**Data Buffer (Bufferpool)**
Table pages and index pages
LRU

**Database files**

 rewritting pages at Checkpoints

**Backup**

 Restore and Roll-forward recovery of transactions

**Archive of log history**

---

**DBMS services**:
- Listener
- Server agents
- Transaction manager
- SQL engine (parser)
- Security manager
- Query Optimizer
- Concurrency manager (Lock manager)
- Deadlock detector
- Recovery manager
- Relational engine
- Memory manager

**OpSys**:
- File manager
- Disk manager

---

**Diagnostics needed after every SQL command**

- Start
- Exceptions
- Errors
- Restart

[reconnect?] Restart?

deadlock? timeout? livelock?

\[\text{[Set Transaction ...]}\]
\[\text{[Begin Work]}\]
if ..

- Insert ..
  if ..
- Select ..
  if ..
- Update ..
  if ..
- Delete ..
  if ..

- Commit
  if ..

\[\text{Rollback}\]

<table>
<thead>
<tr>
<th>Exceptions</th>
<th>Errors</th>
<th>Restart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity?</td>
<td>multiprogramming - limit, timeout</td>
<td>YES</td>
</tr>
<tr>
<td>- Uniqueness</td>
<td>syntax error in dynamic SQL</td>
<td></td>
</tr>
<tr>
<td>- Referential</td>
<td>On plan re-optimizing - Invalid objects/privileges</td>
<td></td>
</tr>
<tr>
<td>- Check</td>
<td>Serializability - conflict - deadlock - timeout</td>
<td>YES</td>
</tr>
<tr>
<td>Not found?</td>
<td>Services of - DBMS buffers, etc</td>
<td></td>
</tr>
<tr>
<td>- Referential</td>
<td>- OpSys - data communication - HW problems</td>
<td></td>
</tr>
</tbody>
</table>

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
    - ACID compromising 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*

- Object.Context.SetComplete or root method end of a declarative transaction
- Distributed Transaction Coordinator (Transaction Manager in application server)

![Diagram of Distributed Transaction Two-Phase Commit (2PC)](image)

- Local Resource Managers with local transactions
- Prepared to Commit?
  - All “Yes” → Commit!
  - A “No” → Abort!
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

<table>
<thead>
<tr>
<th>Feature</th>
<th>A/B/S/O</th>
<th>RDB</th>
<th>LUNI 3.7</th>
<th>Oracle 12c</th>
<th>SQL SERVER 2012</th>
<th>MySQL/InnoDB</th>
<th>PostgreSQL</th>
<th>Pythia 6.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>isolation (server-side)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>explicit start</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>implicit start</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>(configurable)</td>
<td>(configurable)</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>COMMIT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>ROLLBACK</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>implicit rollback on concurrency conflict (deadlock)</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>exception raised</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>implicit rollback on error</td>
<td>left open</td>
<td>n/a</td>
<td>n/a</td>
<td>(configurable)</td>
<td>n/a</td>
<td>(transaction rolled back)</td>
<td>yes</td>
<td>n/a</td>
</tr>
<tr>
<td>SAVEPOINT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>ROLLBACK TO SAVEPOINT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>RELEASE SAVEPOINT</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td>yes</td>
<td>yes</td>
<td>n/a</td>
<td>n/a</td>
</tr>
</tbody>
</table>

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)
1. The Lost Update Problem

"Tellers"

```
<table>
<thead>
<tr>
<th>Transaction A</th>
<th>Account x: balance 1000 €</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;I will take 200 €&quot;</td>
<td></td>
</tr>
<tr>
<td>1. Read account x</td>
<td></td>
</tr>
<tr>
<td>3. balance = balance -200</td>
<td></td>
</tr>
<tr>
<td>5. Write account x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction B</th>
<th>Account x: balance 1000 €</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;I will take 500 €&quot;</td>
<td></td>
</tr>
<tr>
<td>2. Read account x</td>
<td></td>
</tr>
<tr>
<td>4. balance = balance -500</td>
<td></td>
</tr>
<tr>
<td>6. Write account x</td>
<td></td>
</tr>
</tbody>
</table>
```

Lost update!
Concurrent Control by S- and X-locks

Compatibility of S and X locks:

<table>
<thead>
<tr>
<th>Lock of transaction A to object o</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Shared</strong></td>
</tr>
<tr>
<td><strong>Shared</strong></td>
</tr>
<tr>
<td>eXclusive</td>
</tr>
</tbody>
</table>

- 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

1. The Lost Update Problem

"Tellers"

- Applying the locking scheme:

<table>
<thead>
<tr>
<th>Action</th>
<th>Transaction A</th>
<th>Transaction B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Read account x</td>
<td>S-lock</td>
<td>S-lock</td>
</tr>
<tr>
<td>2. Read account x</td>
<td>S-lock</td>
<td>S-lock</td>
</tr>
<tr>
<td>3. balance = balance -200</td>
<td>Wait !</td>
<td>Wait !</td>
</tr>
<tr>
<td>4. balance = balance -500</td>
<td>Wait !</td>
<td>Wait !</td>
</tr>
<tr>
<td>5. Write account x</td>
<td>X-lock?</td>
<td>X-lock?</td>
</tr>
<tr>
<td>6. Write account x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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!

1. The Lost Update Problem

- Applying the locking scheme:

  "Tellers"

  transaction A
  “I will take 200 €”

  1. Read account x
  3. balance = balance -200
  5. Write account x

  account x: balance 1000 €

  transaction B
  “I will take 500 €”

  2. Read account x
  4. balance = balance -500
  6. Write account x

  Deadlock detected

  Find a “victim” and Rollback!

  which solves the problem

Martti Laiho 1998
1. The Lost Update Problem

- solved by locking scheme:

"Tellers"

```
account x:
balance 1000 €
```

`transaction A`
```
“I will take 200 €”
```

`S-lock`

Read account x

`balance = balance -200`

Write account x

`X-lock`

Commit

`transaction B`
```
“I will take 500 €”
```

Retry the transaction of B?

```
Read account x
```

`balance = balance -500`

```
Write account x
```

..Typical anomalies

Dirty Read

```
account x:
balance 1000 €
```

`transaction A`
```
“What is the current balance?”
```

Read the balance of account x

```
balance = balance - 500
```

Update the balance of account x

ROLLBACK

..Typical anomalies
**Non-Repeatable Read**

C.J. Date

1. `SELECT ... FROM table WHERE ... ;`

2. `UPDATE table SET c = ... WHERE ... ;`  
   `DELETE FROM table WHERE ... ;`  
   ...  
   `COMMIT;`

3. `SELECT ... FROM table WHERE ... ;`

4. `COMMIT;`

**Phantom Read**

Martti Laiho

1. `SELECT ... FROM table WHERE ... ;`

2. `INSERT INTO table (...) VALUES ( ... );`
   `UPDATE ... SET col = <matching value> WHERE ... ;`
   `COMMIT;`

3. `SELECT ... FROM table WHERE ... ;`

4. `COMMIT`
ACID SQL transaction

\[
\{ \text{SET} | \text{START} \} \text{ TRANSACTION} \quad \{ \text{READ ONLY} | \text{READ WRITE} \}
\]

\[
\text{ISOLATION LEVEL} \quad \{ \text{READ UNCOMMITTED} | \text{READ COMMITTED} | \text{REPEATABLE READ} | \text{SERIALIZABLE} \}
\]

\[
\{ \text{if...} \}
\]

\[
\text{SET} \quad \{ \text{UNIQUE} | \text{REFERENCIAL} \} \text{ CONSTRAINTS} \quad \{ \text{DEFERRED} | \text{IMMEDIATE} \}
\]

\[
\{ \text{LOCK TABLE ...} \}
\]

\[
\text{SELECT} \quad \ldots
\]

\[
\text{if...}
\]

\[
\text{INSERT} \quad \ldots
\]

\[
\text{if...}
\]

\[
\text{UPDATE} \quad \ldots
\]

\[
\text{if...}
\]

\[
\text{DELETE} \quad \ldots
\]

\[
\text{SAVEPOINT} \quad \text{spn}
\]

\[
\text{COMMIT} | \text{ROLLBACK}
\]

\[
\text{Database}
\]

\[
\text{Transaction log(s)}
\]

Isolation Levels of ISO SQL

<table>
<thead>
<tr>
<th>Isolation Level</th>
<th>Lost Update</th>
<th>Dirty Read</th>
<th>Nonrepeatable Read</th>
<th>Phantoms</th>
</tr>
</thead>
<tbody>
<tr>
<td>READ UNCOMMITTED</td>
<td>NOT possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>READ COMMITTED</td>
<td>NOT possible</td>
<td>NOT possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>REPEATABLE READ</td>
<td>NOT possible</td>
<td>NOT possible</td>
<td>NOT possible</td>
<td>Possible</td>
</tr>
<tr>
<td>SERIALIZABLE</td>
<td>NOT possible</td>
<td>NOT possible</td>
<td>NOT possible</td>
<td>NOT possible</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>Shared</th>
<th>Exclusive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock of transaction A to object o</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Shared</td>
<td>Grant</td>
<td>Wait !</td>
</tr>
<tr>
<td>Exclusive</td>
<td>Wait !</td>
<td>Wait !</td>
</tr>
</tbody>
</table>

Lock request of transaction B to object o

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

Granted Lock (~ 100 bytes)
- Object ID, granule, mode
- owner of the lock

Lock waiting time can be controlled by TIMEOUTs

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

Lock waiting time can be controlled by TIMEOUTs
Locking Mode selected by the Optimizer

Access method?

Index scan
- Update?
  - No
    - IS-lock on table and page
  - Yes
    - IX-lock on table and page
- S-locks on rows to be read
- X-locks on rows to be updated

Table scan
- Update?
  - No
    - S-lock on table
  - Yes
    - X-lock on table

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

1) depending on the isolation level

Multi-Granular Locking (MGL) scheme

- Sample variants of lock compatibility matrices

<table>
<thead>
<tr>
<th>Lock requested</th>
<th>IS</th>
<th>IX</th>
<th>S</th>
<th>IX</th>
</tr>
</thead>
<tbody>
<tr>
<td>IS</td>
<td>grant</td>
<td>grant</td>
<td>grant</td>
<td>wait</td>
</tr>
<tr>
<td>IX</td>
<td>grant</td>
<td>grant</td>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>S</td>
<td>grant</td>
<td>wait</td>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>IX</td>
<td>grant</td>
<td>wait</td>
<td>wait</td>
<td>wait</td>
</tr>
<tr>
<td>IX</td>
<td></td>
<td></td>
<td></td>
<td>wait</td>
</tr>
<tr>
<td>X</td>
<td>wait</td>
<td>wait</td>
<td>wait</td>
<td>wait</td>
</tr>
</tbody>
</table>

SIX = S + IX

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

2. Lock on row

Other locks on index ranges, schemas

Martti Laiho  2007
Compatibility Matrix of SQL Server Locks

<table>
<thead>
<tr>
<th>Key</th>
<th>Compatibility Matrix of SQL Server Locks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SHARE</td>
</tr>
<tr>
<td>2</td>
<td>SHARE WITH INTENTION EXCLUSIVE</td>
</tr>
<tr>
<td>3</td>
<td>EXCLUSIVE</td>
</tr>
</tbody>
</table>

For more information see:
SQL Server Books Online

Multi-Version Concurrency Control (MVCC)

1. copy the row
2. "lock" the row
3. update the row...
... Commit

Original row location on a page of the table

Version store (temporary space)

Latest committed row version

All concurrent transactions will see only a copy in the chain, either the latest committed ("READ COMMITTED") or the latest committed at start of the transaction (SNAPSHOT)

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

Martti Laiho
Phantoms & ghosts in snapshot isolation (SI)

Inserted / updated rows are phantoms
Deleted rows appear as ghosts
for the concurrent snapshots

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

Martti Laiho

Snapshot (start point in time)

Snapshot isolation
Start point in time

1. SELECT ... FROM table
   WHERE ... ;

   result set1

3. SELECT ... FROM table
   WHERE ... ;

   result set2 = result set1

4. COMMIT

transaction A

2. INSERT INTO table (...) VALUES (...);
   DELETE ...
   WHERE ...
   COMMIT;

transaction B

Martti Laiho

1.11.2014
Phantoms and Ghosts of Snapshot

1. SELECT ... FROM table WHERE ... ;

2. INSERT INTO table (..) VALUES ( ... ) ;

3. SELECT ... FROM table WHERE ... ;

4. COMMIT

Start point in time

Snapshot isolation

result set1

transaction A

result set2 = result set1

ghost

transaction B

Insert phantom

update phantom

Inconsistencies of Snapshot

1. SELECT ... FROM table WHERE ... ;

2. INSERT INTO table (...) VALUES ( ... ) ;

3. UPDATE old

4. DELETE old

5. UPDATE phantom

6. UPDATE ghost

7. DELETE phantom

8. DELETE ghost

9. INSERT over phantom

10. INSERT over ghost

11. INSERT new

12. COMMIT

Start point in time

Snapshot isolation

result set

transaction A

ghost

transaction B

Insert phantom

update phantom

Martti Laiho 61

Martti Laiho 62
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)

```
DECLARE crs CURSOR FOR
    SELECT ....
OPEN crs
1.
2.
3.
FETCH crs
resultset
CLOSE crs
or COMMIT ;
```
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