Prof. DI Dr. Erich Gams

Prepared Statements, Transactions

JDBC

informationssysteme htl-wels

Übersicht ☛ Was lernen wir?

- Cursor
- Scrollable Resultset
- ACID
- Transaktionen

Cursor

- You access the data in a ResultSet object through a **cursor**.
- This **cursor is a pointer that points to one row of data in the ResultSet**. Initially, the cursor is **positioned** before the **first row**.
- The method **ResultSet.next** moves the cursor to the next row.

Scrollable ResultSet

- „ A **ResultSet** object is a table of data representing a database result set, which is usually generated by executing a statement that queries the database“

- By default a ResultSet Interface is **Non-Scrollable**
- In non-scrollable ResultSet we can move only in forward direction (that means from first record to last record).
- **Scrollable ResultSet:** Cursor can move both forward and backward direction. Scrollable ResultSet cursor can move randomly

Scrollable ResultSet : resultSetType

- *Statement createStatement(int resultSetType, int resultSetConcurrency);*
- *Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability);*

- *ResultSet.TYPE_FORWARD_ONLY*
  - *JDBC 1 Funktionalität, Cursor zum nächsten Satz, Default*
- *ResultSet.TYPE_SCROLL_INSENSITIVE*
  - *Cursor kann frei bewegt und positioniert werden, Änderungen werden nach Abrufen des Resultsets nicht berücksichtigt*
- *ResultSet.TYPE_SCROLL_SENSITIVE*
  - *Cursor kann frei bewegt und positioniert werden, Änderungen durch andere Transaktionen, während das ResultSet offen ist, sind sichtbar*

Scrollable ResultSet: *resultSetConcurrency*

- *Statement createStatement(int resultSetType, int resultSetConcurrency);*
- *Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability);*

- *ResultSet.CONCUR_READ_ONLY*
  - *keine Änderungen möglich*
- *ResultSet.CONCUR_UPDATABLE*
  - *Änderungen in der Ergebnismenge möglich*

Scrollable ResultSet: *resultSetHoldability*

- *Statement createStatement(int resultSetType, int resultSetConcurrency);*
- *Statement createStatement(int resultSetType, int resultSetConcurrency, int resultSetHoldability);*

- *ResultSet.HOLD_CURSORS_OVER_COMMIT*
  - *ResultSet bleibt offen, auch wenn commited wurde*
- *ResultSet.CLOSE_CURSORS_AT_COMMIT*
  - *ResultSet wurde geschlossen, wenn commit auftritt*

- Gute Quelle (http://docs.oracle.com/javase/tutorial/jdbc/basics/retrieving.html)

Scrollable ResultSet

next()	Bewegt den Datensatzcursor zum nächsten Datensatz
previous()	Bewegt den Datensatzcursor zum vorherigen Datensatz
first()	Bewegt den Datensatzcursor zum ersten Datensatz
last()	Bewegt den Datensatzcursor zum letzten Datensatz
afterLast()	Bewegt den Datensatzcursor hinter den letzten Datensatz
beforeFirst()	Bewegt den Datensatzcursor vor den ersten Datensatz
absolute(int n)	Bewegt den Datensatzcursor auf den *n*-ten Datensatz
relative(int n)	Bewegt den Datensatzcursor relativ zur momentanen Position

Prepared Statement

Select * from kunden where name=“Maier“ and vorname=“Hugo“

Connection conn=….

PreparedStatement ps=conn.prepareStatement(“select * from kunden where name = ? and vorname =?“);

ps.setString(1,“Meier“);

ps.setString(2,“Hugo“);

ResultSet rs = ps.executeQuery();

Was ist SQL-Injection?

Transaktionen

- Ist ein **Verbund von mehreren SQL-Anweisungen**, welche **atomar** ausgeführt werden.
  - Entweder alle oder keine.
  - Überführung von einem konsistenten Zustand in den nächsten.

ACID (Klassische Datenbanken)

- **Atomarität (Abgeschlossenheit)**
  - Sequenz von Daten-Operationen entweder ganz oder gar nicht ausgeführt
  - Konsistenz heißt, dass eine Sequenz von Daten-Operationen nach Beendigung einen konsistenten Datenzustand hinterlässt
- **Konsistenz**
  - Konsistenter Datenzustand -> Sequenz von Operationen -> konsistenter Datenzustand
- **Isolation (Abgrenzung)**
  - Keine Beeinflussung nebenläufiger Daten-Operationen, Transaktionen laufen ungestört von anderen Transaktionen ab (es können nicht die selben Daten von verschiedenen Transaktionen bearbeitet werden).
- **Dauerhaftigkeit**
  - Dauerhaftigkeit: Mit Commit erfolgreich abgeschlossene Änderungen sind persistent

Transaktionen
Methoden des Interfaces Connection

- `void commit( )`
  - schreibt Änderungen in die Datenbank fest
- `void rollback( )`
  - macht Änderungen einer Transaktion rückgängig
- `void setAutoCommit(boolean autoCommit)`
  - schaltet Auto-Commit-Modus ein (true) oder aus.
- `void setTransactionIsolation(int level)`
  - festlegen von **Isolationsstufen** bei konkurrierenden Zugriffen auf die gleiche Ressource

Transaktionen - Ablauf

```
Try
{
        con.setAutoCommit(false);
        stmt.executeUpdate("UPDATE Kunden SET NAME = ‚Lehmann' ");
        stmt.executeUpdate("UPDATE KUNDEN SET Nr = 13 WHERE Nr =12");
        con.commit( );
        con.setAutoCommit(true);
}

catch (SQLException e)
{
        con.rollback();
}
```

Transaktionen

Oder

```
statement.executeUpdate("BEGIN TRANSACTION");

statement.executeUpdate(" UPDATE Kunden SET NAME =
,Lehmann‘ "); // ...

statement.executeUpdate("COMMIT");
```

Dirty Read, Non-repeatable Read (Stale Data), Phantom Read

- Wenn Transaktionen dem ACID-Prinzip entsprechen sollen, müssen sie serialisiert nacheinander durchgeführt werden.
- Aus Performance-Gründen wird hiervon oft abgewichen und ein niedrigerer "Transaction Isolation Level" eingestellt.
- Das kann zu folgenden Fehlern führen:

Dirty Read, Non-repeatable Read (Stale Data), Phantom Read

- **Dirty Read:**
  Es können von anderen Transaktion geschriebene Daten gelesen werden, für die noch kein "Commit" erfolgte und die eventuell per "Rollback" zurückgesetzt werden.
- **Non-repeatable Read (Stale Data):**
  Während einer laufenden Transaktion können Daten von anderen Transaktionen geändert und committed werden, so dass in der laufenden Transaktion ein zweites Auslesen zu anderen Daten führt. (Mehrere Aktionen, während denen sich etwas ändert)
- **Phantom Read:**
  Eine erste Transaktion liest über eine "Where-Klausel" eine Liste von Datensätzen. Eine zweite Transaktion fügt weitere Datensätze hinzu (inkl. Commit). Wenn die erste Transaktion wieder über die gleiche "Where-Klausel" Datensätze liest oder bearbeitet, gibt es mehr Datensätze als vorher.

Isolationsstufen

TRANSACTION_NONE	Transaktionen sind abgeschaltet oder werden nicht unterstützt
TRANSACTION_READ_UNCOMMITED	Transaktionen können Änderungen anderer Transaktionen lesen, bevor diese mit Commit bestätigt oder durch ein Rollback rückgängig gemacht wurden.
TRANSACTION_READ_COMMITED	Es dürfen nur mit Commit bestätigte Werte gelesen werden.
TRANSACTION_REPEATABLE_READ	Sichert, das bei einem wiederholten Lesen innerhalb einer Transaktion immer der gleiche Wert gelesen wird.
TRANSACTION_SERIALIZABLE	Eine Transaktion kann davon ausgehen, dass sie allein ausgeführt wird und dass es eine Reihenfolge der Transaktionen gibt (serialisierte Ausführung).

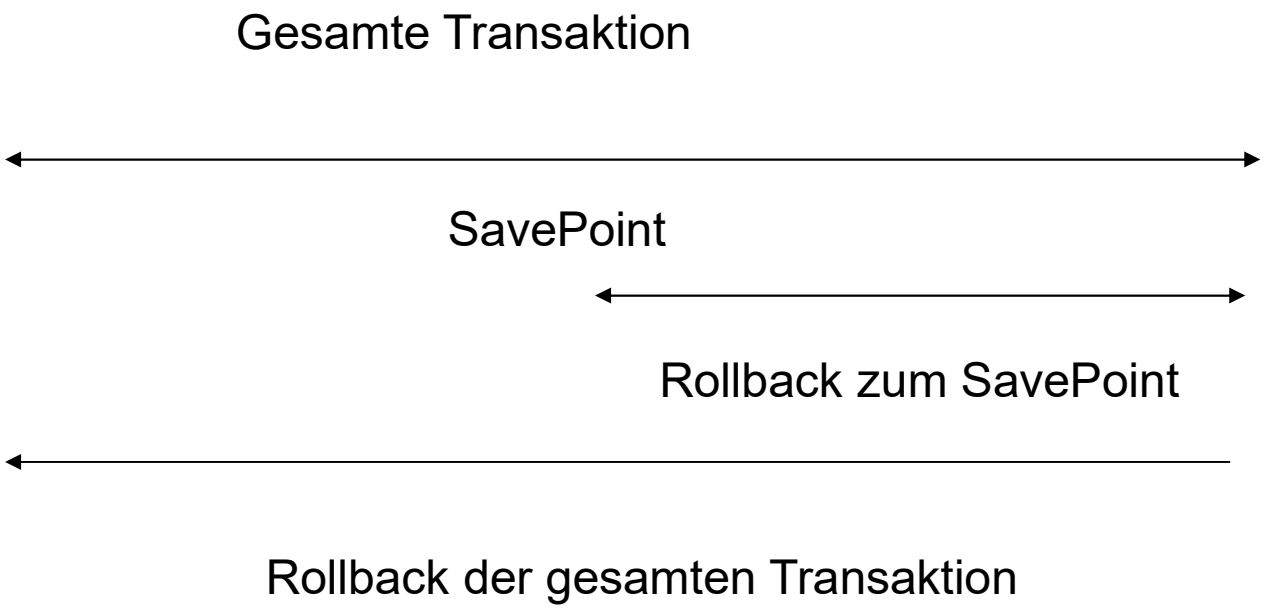
Isolationsstufen

- **Read Uncommitted:**
  - Geringste Isolation, höchste Performance,
    es können Dirty Reads, Non-repeatable Reads und Phantom Reads auftreten
- **Read Committed:**
  - Es gibt keine Dirty Reads mehr, aber es gibt weiterhin Non-repeatable Reads und Phantom Reads
- **Repeatable Read:**
  - Keine Dirty Reads und keine Non-repeatable Reads, aber weiterhin Phantom Reads
- **Serializable:**
  - Keine Dirty Reads, keine Non-repeatable Reads und keine Phantom Reads,
    höchste Isolation, geringste Performance

Transaktionen, Savepoints

- nach mehreren Operationen könne Save Points gesetzt werden.
  - Klasse: `SavePoint`
  - Methode: `setSavePoint( )`
- Transaktionen müssen bei einem Rollback nicht ganz rückgängig gemacht werden, sondern nur bis zu einem Save Point.
  - `rollback(SavePoint sp)`

Sicherungspunkte



Sicherungspunkte

```
Connection conn = …….

Statement stmt = conn.createStatement( );

conn.setAutoCommit(false);

stmt.execute(……);

……

SavePoint sp1 = conn.setSavePoint(„Sicherung1“);

stmt=execute(……);

……

conn.rollback(sp1);  // Rollback zum 1. Sicherungspunkt

conn.commit( );  // Anweisungen bis zum 1. Sicherungspunkt schreiben
```

Aufgabe

- database: playlists
  - movies: title, director, company, year, running time, cast
  - playlist_movies: playlisttitle, movietitle, positionnr
  - playlist: playlisttitle, genre, description
- This data should be outsourced. (Class: Properties)

```
Driver=org.hsqldb.jdbcDriver
url=jdbc:hsqldb:file:.\hsqldb\data\;shutdown=true
User=sa
Password=
```

Aufgabe Resultset

- Erzeuge ein Befehlszeilenprogramm, das neue Filme in die Filmedatenbank einfügen oder bestehende löschen kann.
- Befehle: insert <Daten>, delete <Schlüssel>, leer bedeutet Liste ausgeben
- Hinweise:
  - Nimm die Änderungen im ResultSet vor und schreibe Sie dann in die DB.
  - Verwende eine Properties-Datei für datenbankspezifische Daten.

Aufgabe Transaktion

- Lese mittels Transaktion eine neue Playlist samt der darin befindlichen Filme aus einer Textdatei ein.
- Es können auch mehrere Playlists in der Datei sein.
- Es soll entweder eine gesamte Playlist inkl. der Filme oder, sollte etwas fehlschlagen, nichts in die DB eingetragen werden. (Verwende Transaktionen!)

Aufgabe Transactionlevel

- Finde heraus welches Transactionlevel von deiner Datenbank unterstützt wird.
  - Default Transaction Isolation
  - Actual Transaction Isolation
  - Actual AutoCommit
  - Die Levels im einzelnen: TRANSACTION_READ_UNCOMMITTED, TRANSACTION_READ_COMMITTED, TRANSACTION_REPEATABLE_READ, TRANSACTION_SERIALIZABLE
- Für die Informationen brauchst du die Klassen Connection und DatabaseMetaData und deren Methoden

Auf los geht‘s los ;-)