## A: Vývojový proces JDBC aplikácie

### Registrácia triedy Driver

Tento krok je nepovinný, pretože JDBC 4.0 nevyžaduje priamu registráciu Drivera. V predchádzajúcich verziách bolo možné zaregistrovať Driver dvoma spôsobmi:

* Registrácia pomocou metódy forName() triedy Class. Trieda Driver sa dynamicky načíta pomocou tejto techniky:

Class.forName("oracle.jdbc.driver.OracleDriver");

* Registrácia pomocou metódy registerDriver() triedy DriverManager:

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

### Vytvorenie spojenia

Databáza môže byť lokálna alebo niekde v sieti. V oboch prípadoch je postup pripojenia sa je rovnaký, databázu možno nájsť pomocou adresy URL. Po načítaní ovládača je možné vytvoriť spojenie podľa nasledujúcich pokynov:

Connection connection = DriverManager.*getConnection*(*URL*);

Ukážka 1

Kde URL je reťazec spojenia, ktorého formát adresy závisí od DMBS a ovládača JDBC.

URL má tri časti; dve časti sú oddelené dvojbodkou. Syntax URL je nasledujúca:

<protocol>:<sub-protocol>:<data-source-details>

Časť <protocol> je vždy nastavená na jdbc. Časť <sub-protocol> je špecifická pre dodávateľa. Časť <datasource-details> je špecifická pre systém, ktorá sa používa na lokalizáciu databázy. V niektorých prípadoch je možné v tejto poslednej časti adresy URL špecifikovať aj niektoré vlastnosti spojenia. [1] Nasleduje príklad pripojenia URL na pripojenie k SQLite databáze:

private static final String *URL* = "jdbc:sqlite:chinook.db";

Ukážka 2

### Vytvorenie príkazu

Po vytvorení spojenia je možné komunikovať s databázou. Rozhrania Statement, CallableStatement a PreparedStatement z knižnice JDBC definujú metódy, ktoré umožňujú odosielať príkazy SQL a prijímať údaje z databázy. Použitie JDBC Statement je nasledovné:

Ukážka 3

Statement statement = connection.createStatement();

### Vykonanie dotazov a spracovanie výsledkov

Po vytvorení instancie Statement, je možné vykonávať dotazy (query). Existujú viacero typov query. Niektoré z nich sú nasledovné:

* Query na aktualizáciu/vloženie tabuľky do databázy.
* Query na načítanie údajov.

Metóda executeQuery() sa používa na vykonávanie dotazov na získanie hodnôt z databázy. Táto metóda vracia objekt ResultSet, ktorý možno použiť na získanie všetkých záznamov tabuľky.

Ukážka 4

...  
String **sql** = **"SELECT \* FROM employees"**;  
ResultSet **resultSet** = statement.executeQuery(**sql**);  
...

Metóda executeUpdate(query) sa používa na vykonávanie dotazov na aktualizáciu/vkladanie/odstránenie.

Ukážka 5

...  
String **sql** = **"INSERT INTO artists(Name) VALUES ('Zsolt Kiss')"**;  
**int result** = statement.executeUpdate(**sql**);  
System.***out***.println(**"Records inserted: "** + **result** + **"\n"**);  
...

Rovnaké výsledky je možné dosiahnuť pomocou PreparedStatement následne:

...  
String **sql** = **"INSERT INTO artists(Name) VALUES (?)"**;  
PreparedStatement **preparedStatement** = connection.prepareStatement(**sql**);  
**preparedStatement**.setString(1, **"Zsolt Kiss"**);  
**int result** = **preparedStatement**.executeUpdate();  
System.***out***.println(**"Records inserted: "** + **result** + **"\n"**);  
...

Ukážka 6

### Uzatvorenie spojenia

Po vykonaní požadovaných operácií je možné spojenie uzavrieť. Uzavretím spojenia sa automaticky uzavrú objekty Statement a ResultSet. Na uzavretie spojenia sa používa metóda close() rozhrania Connection. Nižšie je znázornená nasledujúcim spôsobom:

Ukážka 7

connection.close();

Kde connection je objekt, ktorý bol vytvorený v Ukážke 1.

## B: Gramatika SQL parsera a podporované formáty

grammar JFSQL;  
  
root  
 : statement ( SCOL )? EOF  
 ;  
  
statement  
 : createTable  
 | createDatabase  
 | delete  
 | dropTable  
 | dropDatabase  
 | insert  
 | select  
 | update  
 ;  
  
createTable  
 : CREATE TABLE tableName OPEN\_PAR columnDefinition ( COL columnDefinition )\* CLOSE\_PAR  
 ;  
  
createDatabase  
 : CREATE DATABASE databaseUrl  
 ;  
  
delete  
 : DELETE FROM tableName( WHERE expr )?  
 ;  
  
dropTable  
 : DROP TABLE tableName  
 ;  
  
dropDatabase  
 : DROP DATABASE databaseUrl  
 ;  
  
insert  
 : INSERT INTO tableName ( OPEN\_PAR columnName ( COL columnName )\* CLOSE\_PAR )? VALUES valuesInParentheses (COL valuesInParentheses )\*  
 ;  
  
select  
 : SELECT columnName ( COL columnName )\* FROM tableName ( WHERE expr )?  
 ;  
  
update  
 : UPDATE tableName SET columnName EQ value ( COL columnName EQ value )\* ( WHERE expr )?  
 ;  
  
columnDefinition  
 : columnName columnType  
 ;  
  
columnType  
 : INTEGER  
 | REAL  
 | TEXT  
 | BLOB  
 ;  
  
expr  
 : columnName symbol value  
 | columnName symbol value (binaryOperator columnName symbol value)\*  
 ;  
  
binaryOperator  
 : AND  
 | OR  
 ;  
  
symbol  
 : EQ  
 | NOT\_EQ  
 | LT  
 | LT\_EQ  
 | GT  
 | GT\_EQ  
 ;  
  
value  
 : NUMERIC\_LITERAL  
 | STRING\_LITERAL  
 | QUESTION\_MARK  
 ;  
  
valuesInParentheses  
 : OPEN\_PAR value ( COL value )\* CLOSE\_PAR  
 ;  
  
databaseUrl  
 : IDENTIFIER  
 ;  
  
tableName  
 : IDENTIFIER  
 ;  
  
columnName  
 : IDENTIFIER  
 | STAR  
 ;  
  
COL : ',';  
SCOL : ';';  
DOT : '.';  
OPEN\_PAR : '(';  
CLOSE\_PAR : ')';  
STAR : '\*';  
LT : '<';  
LT\_EQ : '<=';  
GT : '>';  
GT\_EQ : '>=';  
EQ : '=';  
NOT\_EQ : '!=';  
QUESTION\_MARK : '?';  
  
AND : A N D;  
OR: O R;  
CREATE : C R E A T E;  
DELETE : D E L E T E;  
DATABASE: D A T A B A S E;  
DROP : D R O P;  
FROM : F R O M;  
INSERT : I N S E R T;  
INTO : I N T O;  
SELECT : S E L E C T;  
SET : S E T;  
TABLE : T A B L E;  
UPDATE : U P D A T E;  
USE : U S E;  
VALUES : V A L U E S;  
WHERE : W H E R E;  
INTEGER : I N T E G E R;  
REAL : R E A L;  
TEXT : T E X T;  
BLOB : B L O B;  
  
IDENTIFIER  
 : '"' ~'"'\* '"'  
 | '[' ~']'\* ']'  
 | [a-zA-Z\_] [a-zA-Z\_0-9]\*  
 ;  
  
NUMERIC\_LITERAL  
 : DIGIT+ ( '.' DIGIT\* )? ( E [-+]? DIGIT+ )?  
 | '.' DIGIT+ ( E [-+]? DIGIT+ )?  
 ;  
  
STRING\_LITERAL  
 : '\'' ( ~'\'' | '\'\'' )\* '\''  
 ;  
  
SPACES  
 : [ \u000B\t\r\n] -> channel(HIDDEN)  
 ;  
  
fragment DIGIT : [0-9];  
fragment A : [aA];  
fragment B : [bB];  
fragment C : [cC];  
fragment D : [dD];  
fragment E : [eE];  
fragment F : [fF];  
fragment G : [gG];  
fragment H : [hH];  
fragment I : [iI];  
fragment J : [jJ];  
fragment K : [kK];  
fragment L : [lL];  
fragment M : [mM];  
fragment N : [nN];  
fragment O : [oO];  
fragment P : [pP];  
fragment Q : [qQ];  
fragment R : [rR];  
fragment S : [sS];  
fragment T : [tT];  
fragment U : [uU];  
fragment V : [vV];  
fragment W : [wW];  
fragment X : [xX];  
fragment Y : [yY];  
fragment Z : [zZ];

### Podporované príkazy a ich formáty

* ALTER TABLE podporuje premenovanie tabuľky a stĺpca, pridanie nového stĺpca do tabuľky a odstránenie existujúceho stĺpca. Podporované formáty sú:

ALTER TABLE myTable RENAME TO myTableEdited;

ALTER TABLE myTable RENAME COLUMN id TO id\_edited;

ALTER TABLE myTable ADD COLUMN id INTEGER;

ALTER TABLE myTable DROP COLUMN id;

* CREATE TABLE podporuje dátové typy INTEGER, REAL, TEXT a BLOB. Podporovaný je klúčové slovo IF NOT EXISTS Tiež je možné pridávať klúčové slovo NOT NULL po jednotlivých dátových typoch:

CREATE TABLE myTable (id INTEGER NOT NULL, name TEXT, age INTEGER);  
CREATE TABLE IF NOT EXISTS myTable (id INTEGER, name TEXT, age INTEGER);

* CREATE DATABASE sa používa na vytvorenie databázy. Tento príkaz je nepovinný a nie je podmienkou na pripojenie adresára. Podporovaný formát je:

CREATE DATABASE [C:\path\to\database];

* DELETE podporuje štandardný formát, v kombinácií s binárnymi operátormi.

DELETE FROM myTable;

DELETE FROM myTable WHERE id > 1;

* DROP TABLE podporuje štandardný formát.

DROP TABLE myTable;  
DROP TABLE IF EXISTS myTable;

* DROP DATABASE slúži na vymazanie databázového adresára. Táto operácia je ireverzibilná. Podporovaný formát je:

DROP DATABASE [C:\path\to\database];

* INSERT podporuje nasledujúce formáty:

INSERT INTO myTable (c1, c2, c3) VALUES (1, 2.5, 'abc');  
INSERT INTO myTable (c1, c2, c3) VALUES (1, 2.5, 'abc'), (2, 2.3, 'def');  
INSERT INTO myTable VALUES (1, 2.5, 'def');  
INSERT INTO myTable VALUES (1, 2.5, 'abc'), (2, 2.3, 'def');

* SELECT podporuje LEFT OUTER JOIN a INNER JOIN, v kombinácií s binárnymi operátormi. V príkazoch obsahujúci JOIN, prefixy pre stĺpce, ktoré sa vyskytnú vo viacerých tabuľkách je povinné.

SELECT *\** FROM myTable;

SELECT id FROM myTable;

SELECT c1, c2, c3 FROM myTable;

SELECT c1, c2, c3 FROM myTable WHERE value1 > 1;

SELECT *\** FROM myTable JOIN myTable2 ON myTable.id = myTable2.id;

SELECT *\** FROM myTable INNER JOIN myTable2 ON myTable.id = myTable2.id;

SELECT *\** FROM myTable LEFT JOIN myTable2 ON myTable.id = myTable2.id;

SELECT *\** FROM myTable LEFT OUTER JOIN myTable2 ON myTable.id = myTable2.id;

* UPDATE podporuje nasledujúce formáty, v kombinácií s binárnymi operátormi.

UPDATE myTable SET value1 = 26;

UPDATE myTable SET value1 = 26 WHERE value1 = 1;

UPDATE myTable SET value1 = 26, value2 = 'abc' WHERE value2 = 'def';

* Podporované binárne operátory sú AND a OR, ktoré možno reťaziť, podporované symboly porovnania sú LIKE, <, <=,=,>=,>.

## C: Program na porovnávanie serializácie a deserializácie

package org.example;  
  
import com.fasterxml.jackson.databind.MappingIterator;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.fasterxml.jackson.databind.json.JsonMapper;  
import com.fasterxml.jackson.dataformat.csv.CsvMapper;  
import com.fasterxml.jackson.dataformat.csv.CsvSchema;  
import com.fasterxml.jackson.dataformat.xml.XmlMapper;  
import com.fasterxml.jackson.dataformat.yaml.YAMLFactory;  
import com.fasterxml.jackson.dataformat.yaml.YAMLGenerator;  
import com.fasterxml.jackson.dataformat.yaml.YAMLMapper;  
import org.instancio.Instancio;  
  
import java.io.File;  
import java.io.IOException;  
import java.io.RandomAccessFile;  
import java.nio.MappedByteBuffer;  
import java.nio.channels.FileChannel;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.List;  
import java.util.stream.Collectors;  
  
public class Main {  
  
 private static final String *CSV\_FILE\_PATH* = "serialization\_test/src/main/resources/output/csv\_version.csv";  
 private static final String *CSV\_FILE\_PATH2* = "serialization\_test/src/main/resources/output/csv\_version2.csv";  
 private static final String *JSON\_FILE\_PATH* = "serialization\_test/src/main/resources/output/json\_version.json";  
 private static final String *JSON\_FILE\_PATH2* = "serialization\_test/src/main/resources/output/json\_version2.json";  
 private static final String *XML\_FILE\_PATH* = "serialization\_test/src/main/resources/output/xml\_version.xml";  
 private static final String *XML\_FILE\_PATH2* = "serialization\_test/src/main/resources/output/xml\_version2.xml";  
 private static final String *YAML\_FILE\_PATH* = "serialization\_test/src/main/resources/output/yaml\_version.yaml";  
 private static final String *YAML\_FILE\_PATH2* = "serialization\_test/src/main/resources/output/yaml\_version2.yaml";  
  
 private static final int *NUMBER\_OF\_RANDOM\_OBJECTS* = 2000000;  
 private static final List<RandomClass> *RANDOM\_VALUE\_OBJECT\_LIST* = *populateListWithRandomObjects*();  
  
 private static List<RandomClass> populateListWithRandomObjects() {  
 System.*out*.println("Populating the list with random value objects");  
 return Instancio.*stream*(RandomClass.class)  
 .limit(*NUMBER\_OF\_RANDOM\_OBJECTS*)  
 .collect(Collectors.*toList*());  
 }  
  
 public static void compareByMemoryMappedFiles(final Path path1, final Path path2) throws IOException {  
 try (final RandomAccessFile randomAccessFile1 = new RandomAccessFile(path1.toFile(), "r");  
 final RandomAccessFile randomAccessFile2 = new RandomAccessFile(path2.toFile(), "r")) {  
  
 final FileChannel ch1 = randomAccessFile1.getChannel();  
 final FileChannel ch2 = randomAccessFile2.getChannel();  
 if (ch1.size() != ch2.size()) {  
 System.*out*.println("Files '" + path1.getFileName() + "' and '" + path2.getFileName() + "' were NOT identical");  
 return;  
 }  
 final long size = ch1.size();  
 final MappedByteBuffer m1 = ch1.map(FileChannel.MapMode.*READ\_ONLY*, 0L, size);  
 final MappedByteBuffer m2 = ch2.map(FileChannel.MapMode.*READ\_ONLY*, 0L, size);  
  
 if (m1.equals(m2)) {  
 System.*out*.println("Files '" + path1.getFileName() + "' and '" + path2.getFileName() + "' were identical");  
 } else {  
 System.*out*.println("Files '" + path1.getFileName() + "' and '" + path2.getFileName() + "' were NOT identical");  
 }  
 }  
 }  
  
 private static void CSV() throws IOException {  
 final CsvMapper csvMapper = new CsvMapper();  
 final CsvSchema csvSchema = csvMapper.schemaFor(RandomClass.class);  
 final long serializationStartTime = System.*nanoTime*();  
 csvMapper.writer(csvSchema.withUseHeader(true)).writeValue(new File(*CSV\_FILE\_PATH*), *RANDOM\_VALUE\_OBJECT\_LIST*);  
 final long serializationEndTime = System.*nanoTime*() - serializationStartTime;  
 System.*out*.println("CSV serialization duration: " + serializationEndTime / 1000000000 + "s");  
 final long deserializationStartTime = System.*nanoTime*();  
 final MappingIterator<RandomClass> iterator;  
 iterator = csvMapper.readerFor(RandomClass.class).with(CsvSchema.*emptySchema*().withHeader()).readValues(new File(*CSV\_FILE\_PATH*));  
 final List<RandomClass> randomClasses = iterator.readAll();  
 final long deserializationEndTime = System.*nanoTime*() - deserializationStartTime;  
 System.*out*.println("CSV deserialization duration: " + deserializationEndTime / 1000000000 + "s");  
 final long reserializationStartTime = System.*nanoTime*();  
 csvMapper.writer(csvSchema.withUseHeader(true)).writeValue(new File(*CSV\_FILE\_PATH2*), randomClasses);  
 final long reserializationEndTime = System.*nanoTime*() - reserializationStartTime;  
 System.*out*.println("CSV reserialization duration: " + reserializationEndTime / 1000000000 + "s");  
 *compareByMemoryMappedFiles*(Path.*of*(*CSV\_FILE\_PATH*), Path.*of*(*CSV\_FILE\_PATH2*));  
 }  
  
 private static void GENERIC(final String format, final ObjectMapper objectMapper, final String filePath, final String filePath2) throws IOException {  
 final long serializationStartTime = System.*nanoTime*();  
 objectMapper.writerWithDefaultPrettyPrinter().writeValue(new File(filePath), *RANDOM\_VALUE\_OBJECT\_LIST*);  
 final long serializationEndTime = System.*nanoTime*() - serializationStartTime;  
 System.*out*.println(format + " serialization duration: " + serializationEndTime / 1000000000 + "s");  
 final long deserializationStartTime = System.*nanoTime*();  
 final MappingIterator<RandomClass> iterator;  
 iterator = objectMapper.readerFor(RandomClass.class).readValues(new File(filePath));  
 final List<RandomClass> randomClasses = iterator.readAll();  
 final long deserializationEndTime = System.*nanoTime*() - deserializationStartTime;  
 System.*out*.println(format + " deserialization duration: " + deserializationEndTime / 1000000000 + "s");  
 final long reserializationStartTime = System.*nanoTime*();  
 objectMapper.writerWithDefaultPrettyPrinter().writeValue(new File(filePath2), randomClasses);  
 final long reserializationEndTime = System.*nanoTime*() - reserializationStartTime;  
 System.*out*.println(format + " reserialization duration: " + reserializationEndTime / 1000000000 + "s");  
 *compareByMemoryMappedFiles*(Path.*of*(filePath), Path.*of*(filePath2));  
 }  
  
 private static void JSON() throws IOException {  
 *GENERIC*("JSON", new JsonMapper(), *JSON\_FILE\_PATH*, *JSON\_FILE\_PATH2*);  
 }  
  
 private static void XML() throws IOException {  
 *GENERIC*("XML", new XmlMapper(), *XML\_FILE\_PATH*, *XML\_FILE\_PATH2*);  
 }  
  
 private static void YAML() throws IOException {  
 *GENERIC*("YAML", new YAMLMapper(new YAMLFactory().disable(YAMLGenerator.Feature.*WRITE\_DOC\_START\_MARKER*)), *YAML\_FILE\_PATH*, *YAML\_FILE\_PATH2*);  
 }  
  
 public static void main(final String[] args) throws IOException {  
 Files.*createDirectories*(Path.*of*("serialization\_test/src/main/resources/output/"));  
 System.*out*.println("Starting serialization processes");  
 *CSV*();  
 *JSON*();  
 *XML*();  
 *YAML*();  
 }  
}

## D: Program na testovanie maximálneho počtu súčasne otvorených súborov

package org.example;  
  
import java.io.\*;  
import java.nio.file.Files;  
import java.nio.file.Path;  
  
public class Main {  
  
 private static final int *NUMBER\_OF\_FILES* = 10000;  
 private static File[] createFiles() throws IOException {  
 File[] files = new File[*NUMBER\_OF\_FILES*];  
 String path = "os\_no\_open\_files\_test/src/main/resources/txt/";  
 Files.*createDirectories*(Path.*of*(path));  
 for (int i = 0; i < *NUMBER\_OF\_FILES*; i++) {  
 File file = new File(path + "file\_" + i + ".txt");  
 if (file.createNewFile()) {  
 files[i] = file;  
 }  
 else {  
 System.*out*.println("The file already exists.");  
 }  
 }  
 return files;  
 }  
 public static void main(String[] args) throws IOException {  
  
 String longLoremIpsum = String.*valueOf*(Files.*readString*(Path.*of*("os\_no\_open\_files\_test/src/main/resources/lorem.txt")));  
 File[] files = *createFiles*();  
 PrintStream[] fileWriters = new PrintStream[*NUMBER\_OF\_FILES*];  
  
 for (int i = 0; i < *NUMBER\_OF\_FILES*; i++) {  
 *// Not using try-with-resources on purpose* try {  
 fileWriters[i] = new PrintStream("os\_no\_open\_files\_test/src/main/resources/txt/" + files[i].getName());  
 fileWriters[i].print(longLoremIpsum);  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 *// Closing all the opened files at once* for (int i = 0; i < *NUMBER\_OF\_FILES*; i++) {  
 fileWriters[i].close();  
 }  
 }  
}