# A: Technická dokumentácia

## Systémové požiadavky

Ovládač sa spolieha na systém správy závislostí Maven. Ak chcete ovládač používať v rámci iného programu, je nevyhnutné mať nainštalovaný Maven. Ovládač bol vyvinutý v jazyku Java verzie 11 a podporuje operačné systémy Microsoft Windows aj Unix.

## Inštalácia

1. V termináli alebo príkazovom riadku prejdite do adresára projektu.

Spustite nasledujúci príkaz na inštaláciu ovládača pomocou nástroja Maven:

*mvn clean install*

Tento príkaz zostaví ovládač a nainštaluje ho do vášho lokálneho úložiska Maven.

1. Otvorte súbor pom.xml svojho projektu a do sekcie <dependencies> pridajte nasledujúcu závislosť:

<dependency>  
 <groupId>com.github.jfsql</groupId>  
 <artifactId>JFSQL-driver</artifactId>  
 <version>1.0</version>  
</dependency>

## Nastavenie pripojenia

Reťazec pripojenia pre ovládač JFSQL JDBC sa riadi štandardným formátom JDBC. Má nasledujúcu štruktúru:

jdbc:jfsql:<<absolútna cesta k adresáru>>

Príklad správneho reťazca pripojenia:

jdbc:jfsql:C:path/to/myDatabase

## Podporované konfigurácie ovládača

Podporované sú nasledovné konfigurácie:

"persistence": "json" alebo "xml"

"transaction.versioning": "default" alebo "jgit"

"schema.validation": "true" alebo "false"

"statement.caching": "true" alebo "false"

Pri hodnotách sa nerozlišujú veľké a malé písmená, ale ak používateľ zadá niektorú hodnotu v nesprávnom tvare, bude sa používať predvolená hodnota daného atribútu.

## Vykonávanie príkazov

Ovládač funguje podobne, ako hocijaký JDBC ovládač. Existujú však obmedzenia, ako syntax, ktoré je opísaný v časti Podporované príkazy a ich formáty.

# B: Podporované SQL 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 kľúčové slovo IF NOT EXISTS. Tiež je možné pridávať kľúčové 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);

* 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;

* INSERT podporuje nasledujúce formáty:

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

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

INSERT INTO myTable VALUES (default, 2.5, 'abc'), (2, 2.3, 'def');

Pri používaní default hodnoty ak ide o dátový typ INTEGER sa automaticky inkrementuje najväčšia hodnota v stĺpci.

SELECT podporuje LEFT OUTER JOIN a INNER JOIN v kombinácii s binárnymi operátormi, ako aj LIMIT, OFFSET a ORDER BY. Tabuľky je možné spájať len na základe jedného stĺpca a jednej podmienky, reťazenie podmienok nie je podporované.

SELECT *\** FROM myTable;

SELECT id FROM myTable;

SELECT id FROM myTable LIMIT 10 OFFSET 2;

SELECT id FROM myTable LIMIT 10 OFFSET 2 ORDER BY id ASC;

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: Gramatika SQL parsera

grammar JFSQL;  
  
root  
 : statement ( SCOL )? EOF  
 ;  
  
statement  
 : alterTable  
 | createTable  
 | delete  
 | dropTable  
 | insert  
 | select  
 | update  
 ;  
  
alterTable  
 : ALTER TABLE tableName ( renameTable | renameColumn | addColumn | dropColumn )  
 ;  
  
renameTable  
 : RENAME TO tableName  
 ;  
  
renameColumn  
 : RENAME COLUMN columnName TO columnName  
 ;  
  
addColumn  
 : ADD COLUMN columnDefinition  
 ;  
  
dropColumn  
 : DROP COLUMN columnName  
 ;  
  
createTable  
 : CREATE TABLE ( ifNotExists )? tableName OPEN\_PAR columnDefinition ( COL columnDefinition )\* CLOSE\_PAR  
 ;  
  
delete  
 : DELETE FROM tableName ( WHERE expr )?  
 ;  
  
dropTable  
 : DROP TABLE ( ifExists )? tableName  
 ;  
  
insert  
 : INSERT INTO tableName ( OPEN\_PAR columnName ( COL columnName )\* CLOSE\_PAR )? VALUES valuesInParentheses ( COL valuesInParentheses )\*  
 ;  
  
select  
 : SELECT columnName ( COL columnName )\* FROM tableName ( joinOperation )\* ( orderBy )? ( WHERE expr )? ( limit )?  
 ;  
  
orderBy  
 : ORDER BY columnName ( ordering )?  
 ;  
  
ordering  
 : ASC | DESC  
 ;  
  
limit  
 : LIMIT numericValue  
 | LIMIT numericValue offset  
 ;  
  
offset  
 : OFFSET numericValue  
 ;  
  
numericValue  
 : NUMERIC\_LITERAL  
 ;  
  
joinOperation  
 : innerJoin  
 | leftJoin  
 ;  
  
innerJoin  
 : ( INNER )? JOIN tableName ON tableDotColumnName EQ tableDotColumnName  
 ;  
  
leftJoin  
 : LEFT ( OUTER )? JOIN tableName ON tableDotColumnName EQ tableDotColumnName  
 ;  
  
update  
 : UPDATE tableName SET columnName EQ value ( COL columnName EQ value )\* ( WHERE expr )?  
 ;  
  
columnDefinition  
 : columnName columnType ( notNull )?  
 ;  
  
notNull  
 : NOT NULL;  
  
ifExists  
 : IF EXISTS  
 ;  
  
ifNotExists  
 : IF NOT EXISTS  
 ;  
  
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  
 | LIKE  
 ;  
  
value  
 : NUMERIC\_LITERAL  
 | STRING\_LITERAL  
 | QUESTION\_MARK  
 | 'default'  
 ;  
  
valuesInParentheses  
 : OPEN\_PAR value ( COL value )\* CLOSE\_PAR  
 ;  
  
tableName  
 : IDENTIFIER  
 ;  
  
tableDotColumnName  
 : IDENTIFIER DOT IDENTIFIER  
 ;  
  
columnName  
 : IDENTIFIER  
 | IDENTIFIER DOT IDENTIFIER  
 ;  
  
COL : ',';  
SCOL : ';';  
DOT : '.';  
OPEN\_PAR : '(';  
CLOSE\_PAR : ')';  
LT : '<';  
LT\_EQ : '<=';  
GT : '>';  
GT\_EQ : '>=';  
EQ : '=';  
NOT\_EQ : '!=';  
QUESTION\_MARK : '?';  
  
NOT : N O T;  
NULL : N U L L;  
IF : I F;  
EXISTS : E X I S T S;  
LIKE : L I K E;  
LIMIT : L I M I T;  
OFFSET : O F F S E T;  
ORDER : O R D E R;  
BY : B Y;  
ASC : A S C;  
DESC : D E S C;  
AND : A N D;  
OR: O R;  
ADD : A D D;  
ALTER : A L T E R;  
CREATE : C R E A T E;  
COLUMN : C O L U M N;  
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;  
VALUES : V A L U E S;  
WHERE : W H E R E;  
RENAME : R E N A M E;  
TO : T O;  
ON : O N;  
LEFT : L E F T;  
JOIN : J O I N;  
INNER : I N N E R;  
OUTER : O U T E R;  
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];

# D: Program na porovnávanie serializácie a deserializácie

## Trieda Main

package com.github;  
  
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 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;  
import org.instancio.Instancio;  
  
public class Main {  
  
 private static final String *CSV\_FILE\_PATH* = "src/main/resources/output/csv\_version.csv";  
 private static final String *CSV\_FILE\_PATH2* = "src/main/resources/output/csv\_version2.csv";  
 private static final String *JSON\_FILE\_PATH* = "src/main/resources/output/json\_version.json";  
 private static final String *JSON\_FILE\_PATH2* = "src/main/resources/output/json\_version2.json";  
 private static final String *XML\_FILE\_PATH* = "src/main/resources/output/xml\_version.xml";  
 private static final String *XML\_FILE\_PATH2* = "src/main/resources/output/xml\_version2.xml";  
 private static final String *YAML\_FILE\_PATH* = "src/main/resources/output/yaml\_version.yaml";  
 private static final String *YAML\_FILE\_PATH2* = "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*("src/main/resources/output/"));  
 System.*out*.println("Starting serialization processes");  
 *csv*();  
 *json*();  
 *xml*();  
 *yaml*();  
 }  
}

## Trieda RandomClass

package com.github;  
  
import java.math.BigDecimal;  
import lombok.Data;  
  
@Data  
public class RandomClass {  
  
 private String randomString;  
 private Byte randomByte;  
 private Integer randomInteger;  
 private Long randomLong;  
 private Float randomFloat;  
 private Double randomDouble;  
 private Boolean randomBoolean;  
 private Character randomCharacter;  
 private BigDecimal randomBigDecimal;  
  
}

## Súbor pom.xml

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.github</groupId>  
 <artifactId>Serialization-test</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>11</maven.compiler.source>  
 <maven.compiler.target>11</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.instancio</groupId>  
 <artifactId>instancio-core</artifactId>  
 <version>2.11.0</version>  
 <scope>compile</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.26</version>  
 <scope>provided</scope>  
 </dependency>  
 <dependency>  
 <groupId>com.fasterxml.jackson.dataformat</groupId>  
 <artifactId>jackson-dataformat-yaml</artifactId>  
 <version>2.13.4</version>  
 </dependency>  
 <dependency>  
 <groupId>com.fasterxml.jackson.dataformat</groupId>  
 <artifactId>jackson-dataformat-xml</artifactId>  
 <version>2.13.4</version>  
 </dependency>  
 <dependency>  
 <groupId>com.fasterxml.jackson.dataformat</groupId>  
 <artifactId>jackson-dataformat-csv</artifactId>  
 <version>2.13.4</version>  
 </dependency>  
 </dependencies>  
  
</project>

# E: Program na testovanie maximálneho počtu súčasne otvorených súborov

POZOR! Tento program vytvorí 1 milión súborov a pokúsi sa ich naraz načítať do pamäte, pričom do nej zapíše ASCI znaky. To môže spôsobiť, že systém zamrzne alebo prestane reagovať.

## Trieda Constants

package com.github;  
  
import lombok.experimental.UtilityClass;  
  
import java.io.File;  
import java.nio.file.Path;  
  
@UtilityClass  
public class Constants {  
  
 private static final String *BASE\_PATH* = String.*valueOf*(Path.*of*("").toAbsolutePath());  
 public static final Path *RESOURCES\_PATH* = Path.*of*(*BASE\_PATH*,"src", "main", "resources");  
 public static final Path *TXT\_OUTPUT\_FOLDER* = *RESOURCES\_PATH*.resolve("txt");  
 public static final Path *LOREM\_FILE* = *RESOURCES\_PATH*.resolve("lorem.txt");  
 public static final int *NUMBER\_OF\_FILES* = 1000000;  
  
}

V hore uvedenom reťazca sa definuje absolútna cesta k súboru lorem.txt, odkiaľ program načítava reťazec. Na spustenie programu je potrebné zmeniť cestu ak chcete súbor načítať z iného miesta. Predvolene súbor sa nachádza pod adresárom *src/main/resources/lorem.txt*

## Súbor lorem.txt

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis vel volutpat massa, vel sodales libero. Fusce sed leo in sapien eleifend accumsan a quis tellus. Aliquam erat volutpat. Aliquam risus felis, facilisis iaculis faucibus ut, convallis vel augue. Sed maximus sem ut porttitor sagittis. Aenean convallis orci at nisi consequat sagittis. Nam imperdiet risus suscipit, dictum ex non, suscipit turpis. Ut iaculis erat vitae est fringilla, vel condimentum justo varius. Ut vitae purus in ante facilisis molestie. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia curae; Integer sed nisi eu justo facilisis pretium quis sed enim. Ut ultricies, ipsum eget dictum luctus, magna arcu condimentum ligula, vitae porttitor nulla metus sit amet ex. Pellentesque vitae lobortis nulla, et efficitur augue. Sed porttitor et quam ut aliquet. Aenean vulputate leo ut odio posuere faucibus. Nullam auctor ipsum eu elit mattis dictum.

Curabitur est massa, auctor sit amet felis non, vehicula eleifend velit. Nullam tempus quam tristique sem aliquet, a tristique erat vehicula. Nulla ultricies facilisis ullamcorper. Sed vel nisl hendrerit est mollis semper. Curabitur suscipit, mi quis faucibus mollis, dolor purus pharetra augue, id tempus ex mauris a ipsum. Morbi volutpat augue neque, ut facilisis nunc condimentum tempor. In hac habitasse platea dictumst. Cras quis ex posuere, egestas nulla quis, consectetur dolor. Aenean a ipsum tempus sapien iaculis gravida. Vestibulum diam libero, commodo a malesuada a, feugiat eget lacus.

Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Quisque vulputate auctor pretium. Nunc massa arcu, consectetur vitae risus hendrerit, porta suscipit dui. Aliquam erat volutpat. Donec nec mauris luctus, tempor ante porta, viverra leo. Suspendisse euismod ex non ligula venenatis, nec laoreet turpis tincidunt. Donec luctus commodo turpis, et porttitor nibh consequat ut. Duis eu orci finibus, aliquet odio.

## Trieda Main

package com.github;  
  
import java.io.File;  
import java.io.IOException;  
import java.io.PrintStream;  
import java.nio.file.Files;  
  
public class Main {  
  
 private static File[] createFiles() throws IOException {  
 final File[] files = new File[Constants.*NUMBER\_OF\_FILES*];  
 Files.*createDirectories*(Constants.*TXT\_OUTPUT\_FOLDER*);  
 for (int i = 0; i < Constants.*NUMBER\_OF\_FILES*; i++) {  
 final File file = new File(Constants.*TXT\_OUTPUT\_FOLDER* + File.*separator* + "file\_" + i + ".txt");  
 if (file.createNewFile()) {  
 files[i] = file;  
 } else {  
 System.*out*.println("The file already exists.");  
 }  
 }  
 return files;  
 }  
  
 public static void main(final String[] args) throws IOException {  
  
 final String longLoremIpsum = String.*valueOf*(Files.*readString*(Constants.*LOREM\_FILE*));  
 final File[] files = *createFiles*();  
 final PrintStream[] fileWriters = new PrintStream[Constants.*NUMBER\_OF\_FILES*];  
  
 for (int i = 0; i < Constants.*NUMBER\_OF\_FILES*; i++) {  
 *// Not using try-with-resources on purpose* try {  
 fileWriters[i] = new PrintStream(Constants.*TXT\_OUTPUT\_FOLDER* + File.*separator* + files[i].getName());  
 fileWriters[i].print(longLoremIpsum);  
 } catch (final IOException e) {  
 e.printStackTrace();  
 }  
 }  
 *// Closing all the opened files at once* for (int i = 0; i < Constants.*NUMBER\_OF\_FILES*; i++) {  
 fileWriters[i].close();  
 }  
 }  
}

## Súbor pom.xml

*<?*xml version="1.0" encoding="UTF-8"*?>*<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.github</groupId>  
 <artifactId>Max-files-test</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>11</maven.compiler.source>  
 <maven.compiler.target>11</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.projectlombok</groupId>  
 <artifactId>lombok</artifactId>  
 <version>1.18.26</version>  
 <scope>compile</scope>  
 </dependency>  
 </dependencies>  
  
</project>

# F: Program na porovnávanie ovládača s SQLite

## Trieda Delete

package com.github.services;  
  
import com.github.Constants;  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import lombok.experimental.UtilityClass;  
  
@UtilityClass  
public class Delete {  
  
 public void delete() {  
 final File file = new File(  
 Constants.*SCRIPTS\_FOLDER* + File.*separator* + "deletes.sql");  
 for (final String connectionString : Constants.*CONNECTION\_STRINGS*) {  
 try (final Connection connection = DriverManager.*getConnection*(connectionString);  
 final Statement statement = connection.createStatement()) {  
 connection.setAutoCommit(false);  
 final long startTime = System.*nanoTime*();  
 int i = 0;  
 while (i != 10000) {  
 try (final BufferedReader reader = new BufferedReader(new FileReader(file.getAbsolutePath()))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 statement.execute(line);  
 i++;  
 }  
 } catch (final IOException e) {  
 e.printStackTrace();  
 }  
 }  
 connection.commit();  
 final long endTime = System.*nanoTime*() - startTime;  
 System.*out*.println(  
 "Executed 10 000 DELETE statements on " + connectionString + " duration: " + endTime / 1000000  
 + "ms");  
 } catch (final SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

## Trieda Insert

package com.github.services;  
  
import com.github.Constants;  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import lombok.experimental.UtilityClass;  
  
@UtilityClass  
public class Insert {  
  
 public void insert() {  
 final File file = new File(  
 Constants.*SCRIPTS\_FOLDER* + File.*separator* + "inserts.sql");  
 for (final String connectionString : Constants.*CONNECTION\_STRINGS*) {  
 try (final Connection connection = DriverManager.*getConnection*(connectionString);  
 final Statement statement = connection.createStatement()) {  
 connection.setAutoCommit(false);  
 final long startTime = System.*nanoTime*();  
 try (final BufferedReader reader = new BufferedReader(new FileReader(file.getAbsolutePath()))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 statement.execute(line);  
 }  
 } catch (final IOException e) {  
 e.printStackTrace();  
 }  
 connection.commit();  
 final long endTime = System.*nanoTime*() - startTime;  
 System.*out*.println(  
 "Executed 10 000 INSERT statements on " + connectionString + " duration: " + endTime / 1000000  
 + "ms");  
 } catch (final SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
}

## Trieda Populate

package com.github.services;  
  
import com.github.Constants;  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import lombok.experimental.UtilityClass;  
  
@UtilityClass  
public class Populate {  
  
 public void populate() {  
 final File file = new File(  
 Constants.*SCRIPTS\_FOLDER* + File.*separator* + "populate.sql");  
 for (final String connectionString : Constants.*CONNECTION\_STRINGS*) {  
 try (final Connection connection = DriverManager.*getConnection*(connectionString);  
 final Statement statement = connection.createStatement()) {  
 connection.setAutoCommit(false);  
 final long startTime = System.*nanoTime*();  
 try (final BufferedReader reader = new BufferedReader(new FileReader(file.getAbsolutePath()))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 statement.execute(line);  
 }  
 } catch (final IOException e) {  
 e.printStackTrace();  
 }  
  
 connection.commit();  
 final long endTime = System.*nanoTime*() - startTime;  
 System.*out*.println(  
 "Populated the database " + connectionString + " duration: " + endTime / 1000000 + "ms");  
 } catch (final SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
   
}

## Trieda Select

package com.github.services;  
  
import com.github.Constants;  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import java.util.concurrent.atomic.AtomicInteger;  
import lombok.experimental.UtilityClass;  
  
@UtilityClass  
public class Select {  
  
 public void select() {  
 final File file = new File(  
 Constants.*SCRIPTS\_FOLDER* + File.*separator* + "selects.sql");  
 final AtomicInteger j = new AtomicInteger();  
 for (final String connectionString : Constants.*CONNECTION\_STRINGS*) {  
 j.getAndIncrement();  
 try (final Connection connection = DriverManager.*getConnection*(connectionString);  
 final Statement statement = connection.createStatement()) {  
 final long startTime = System.*nanoTime*();  
 int i = 0;  
 while (i != 10000) {  
 try (final BufferedReader reader = new BufferedReader(new FileReader(file.getAbsolutePath()))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 statement.execute(line);  
 i++;  
 }  
 } catch (final IOException e) {  
 e.printStackTrace();  
 }  
 }  
 final long endTime = System.*nanoTime*() - startTime;  
 System.*out*.println(  
 "Executed 10 000 SELECT statements on " + connectionString + " duration: " + endTime / 1000000  
 + "ms");  
 } catch (final SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

## Trieda Update

package com.github.services;  
  
import com.github.Constants;  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.sql.Statement;  
import lombok.experimental.UtilityClass;  
  
@UtilityClass  
public class Update {  
  
 public void update() {  
 final File file = new File(  
 Constants.*SCRIPTS\_FOLDER* + File.*separator* + "updates.sql");  
 for (final String connectionString : Constants.*CONNECTION\_STRINGS*) {  
 try (final Connection connection = DriverManager.*getConnection*(connectionString);  
 final Statement statement = connection.createStatement()) {  
 connection.setAutoCommit(false);  
 final long startTime = System.*nanoTime*();  
 int i = 0;  
 while (i != 10000) {  
 try (final BufferedReader reader = new BufferedReader(new FileReader(file.getAbsolutePath()))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 statement.execute(line);  
 i++;  
 }  
 } catch (final IOException e) {  
 e.printStackTrace();  
 }  
 }  
 connection.commit();  
 final long endTime = System.*nanoTime*() - startTime;  
 System.*out*.println(  
 "Executed 10 000 UPDATE statements on " + connectionString + " duration: " + endTime / 1000000  
 + "ms");  
 } catch (final SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

## Trieda ResultSetPrinter

package com.github.util;  
  
import java.io.BufferedWriter;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.PrintWriter;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import lombok.experimental.UtilityClass;  
  
@UtilityClass  
public class ResultSetPrinter {  
  
 public static void printResultSet(final ResultSet resultSet, final String filePath)  
 throws SQLException, IOException {  
 final int numColumns = resultSet.getMetaData().getColumnCount();  
 final PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(filePath, true)));  
  
 while (resultSet.next()) {  
 for (int i = 1; i <= numColumns; i++) {  
 writer.printf("%-15s", resultSet.getString(i));  
 }  
 writer.println();  
 }  
 writer.close();  
 }  
  
}

## Trieda SqlScriptGenerator

package com.github.util;  
  
import com.github.javafaker.Faker;  
import java.io.BufferedWriter;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.util.List;  
import java.util.concurrent.TimeUnit;  
  
public class SqlScriptGenerator {  
  
 private static final List<String> *CAR\_MAKES* = List.*of*("Toyota", "Ford", "Honda", "Chevrolet", "Nissan", "Jeep",  
 "Subaru", "BMW", "Audi", "Mercedes-Benz", "Mazda", "Kia", "Hyundai", "Volkswagen", "Lexus", "Volvo", "Chrysler",  
 "Dodge", "GMC");  
 private static final List<String> *TOYOTA\_MODELS* = List.*of*("Corolla", "Camry", "Prius", "Supra", "Tacoma", "Tundra");  
 private static final List<String> *FORD\_MODELS* = List.*of*("F-150", "Mustang", "Explorer", "Expedition", "Escape",  
 "Focus");  
 private static final List<String> *HONDA\_MODELS* = List.*of*("Civic", "Accord", "Fit", "CR-V", "Odyssey", "Pilot");  
 private static final List<String> *CHEVROLET\_MODELS* = List.*of*("Silverado", "Equinox", "Malibu", "Traverse", "Camaro",  
 "Corvette");  
 private static final List<String> *NISSAN\_MODELS* = List.*of*("Altima", "Sentra", "Maxima", "Rogue", "Pathfinder",  
 "Frontier");  
 private static final List<String> *JEEP\_MODELS* = List.*of*("Grand Cherokee", "Wrangler", "Cherokee", "Compass",  
 "Renegade", "Gladiator");  
 private static final List<String> *SUBARU\_MODELS* = List.*of*("Outback", "Impreza", "Legacy", "Crosstrek", "Ascent",  
 "Forester");  
 private static final List<String> *BMW\_MODELS* = List.*of*("3 Series", "5 Series", "7 Series", "X1", "X5", "M3");  
 private static final List<String> *AUDI\_MODELS* = List.*of*("A4", "Q5", "A6", "Q7", "S5", "TT");  
 private static final List<String> *MERCEDES\_MODELS* = List.*of*("C-Class", "E-Class", "S-Class", "GLC", "GLE",  
 "AMG GT");  
 private static final List<String> *MAZDA\_MODELS* = List.*of*("Mazda3", "Mazda6", "MX-5 Miata", "CX-30", "CX-5", "CX-9");  
 private static final List<String> *KIA\_MODELS* = List.*of*("Soul", "Forte", "Optima", "Seltos", "Sportage", "Stinger");  
 private static final List<String> *HYUNDAI\_MODELS* = List.*of*("Elantra", "Sonata", "Tucson", "Santa Fe", "Kona",  
 "Venue");  
 private static final List<String> *VOLKSWAGEN\_MODELS* = List.*of*("Jetta", "Passat", "Tiguan", "Atlas", "Golf GTI",  
 "Arteon");  
 private static final List<String> *LEXUS\_MODELS* = List.*of*("ES", "IS", "GS", "NX", "RX", "LC");  
 private static final List<String> *VOLVO\_MODELS* = List.*of*("S60", "S90", "V60", "V90", "XC40", "XC90");  
 private static final List<String> *CHRYSLER\_MODELS* = List.*of*("300", "Pacifica", "Voyager", "Challenger", "Charger",  
 "Durango");  
 private static final List<String> *DODGE\_MODELS* = List.*of*("Challenger", "Charger", "Ram 1500", "Ram 2500",  
 "Ram 3500", "Journey");  
 private static final List<String> *GMC\_MODELS* = List.*of*("1500", "2500", "3500", "Sierra 1500", "Sierra 2500HD",  
 "Canyon");  
  
 private static final Faker *FAKER* = new Faker();  
  
 *// 10 000 cars, 20 dealerships, 5 000 owners, 10 000 services, 8 000 sales* private static String insertIntoCars(final int index) {  
 final String make = *CAR\_MAKES*.get(*FAKER*.random().nextInt(0, *CAR\_MAKES*.size() - 1));  
 final String model = switch (make) {  
 case "Toyota" -> *TOYOTA\_MODELS*.get(*FAKER*.random().nextInt(0, *TOYOTA\_MODELS*.size() - 1));  
 case "Ford" -> *FORD\_MODELS*.get(*FAKER*.random().nextInt(0, *FORD\_MODELS*.size() - 1));  
 case "Honda" -> *HONDA\_MODELS*.get(*FAKER*.random().nextInt(0, *HONDA\_MODELS*.size() - 1));  
 case "Chevrolet" -> *CHEVROLET\_MODELS*.get(*FAKER*.random().nextInt(0, *CHEVROLET\_MODELS*.size() - 1));  
 case "Nissan" -> *NISSAN\_MODELS*.get(*FAKER*.random().nextInt(0, *NISSAN\_MODELS*.size() - 1));  
 case "Jeep" -> *JEEP\_MODELS*.get(*FAKER*.random().nextInt(0, *JEEP\_MODELS*.size() - 1));  
 case "Subaru" -> *SUBARU\_MODELS*.get(*FAKER*.random().nextInt(0, *SUBARU\_MODELS*.size() - 1));  
 case "BMW" -> *BMW\_MODELS*.get(*FAKER*.random().nextInt(0, *BMW\_MODELS*.size() - 1));  
 case "Audi" -> *AUDI\_MODELS*.get(*FAKER*.random().nextInt(0, *AUDI\_MODELS*.size() - 1));  
 case "Mercedes-Benz" -> *MERCEDES\_MODELS*.get(*FAKER*.random().nextInt(0, *MERCEDES\_MODELS*.size() - 1));  
 case "Mazda" -> *MAZDA\_MODELS*.get(*FAKER*.random().nextInt(0, *MAZDA\_MODELS*.size() - 1));  
 case "Kia" -> *KIA\_MODELS*.get(*FAKER*.random().nextInt(0, *KIA\_MODELS*.size() - 1));  
 case "Hyundai" -> *HYUNDAI\_MODELS*.get(*FAKER*.random().nextInt(0, *HYUNDAI\_MODELS*.size() - 1));  
 case "Volkswagen" -> *VOLKSWAGEN\_MODELS*.get(*FAKER*.random().nextInt(0, *VOLKSWAGEN\_MODELS*.size() - 1));  
 case "Lexus" -> *LEXUS\_MODELS*.get(*FAKER*.random().nextInt(0, *LEXUS\_MODELS*.size() - 1));  
 case "Volvo" -> *VOLVO\_MODELS*.get(*FAKER*.random().nextInt(0, *VOLVO\_MODELS*.size() - 1));  
 case "Chrysler" -> *CHRYSLER\_MODELS*.get(*FAKER*.random().nextInt(0, *CHRYSLER\_MODELS*.size() - 1));  
 case "Dodge" -> *DODGE\_MODELS*.get(*FAKER*.random().nextInt(0, *DODGE\_MODELS*.size() - 1));  
 case "GMC" -> *GMC\_MODELS*.get(*FAKER*.random().nextInt(0, *GMC\_MODELS*.size() - 1));  
 default -> throw new IllegalStateException("Unexpected car make");  
 };  
 final int year = *FAKER*.number().numberBetween(1990, 2023);  
 final String color = *FAKER*.color().name();  
 final String transmissionType = *FAKER*.options().option("Automatic", "Manual");  
 final String fuelType = *FAKER*.options().option("Gasoline", "Diesel", "Electric");  
 final double engineSize = *FAKER*.number().randomDouble(1, (int) 1.0, (int) 5.0);  
 final int numberOfDoors = *FAKER*.number().numberBetween(2, 5);  
  
 final String sql = String.*format*(  
 "INSERT INTO Car (car\_id, make, model, year, color, transmission\_type, fuel\_type, engine\_size, number\_of\_doors) VALUES (%d, '%s', '%s', %d, '%s', '%s', '%s', %.1f, %d);",  
 index, make, model, year, color, transmissionType, fuelType, engineSize, numberOfDoors);  
  
 System.*out*.println(sql);  
 return sql;  
 }  
  
 private static void createCarScript() {  
 try (final BufferedWriter writer = new BufferedWriter(  
 new FileWriter("src/main/resources/scripts/populate/populate.sql"))) {  
 for (int i = 0; i < 10000; i++) {  
 if (i == 0) {  
 writer.write("""  
 DROP TABLE IF EXISTS Car;  
 CREATE TABLE Car (car\_id INTEGER NOT NULL, make TEXT NOT NULL, model TEXT NOT NULL, year INTEGER NOT NULL, color TEXT NOT NULL, transmission\_type TEXT NOT NULL, fuel\_type TEXT NOT NULL, engine\_size REAL NOT NULL, number\_of\_doors INTEGER NOT NULL);  
 """);  
 }  
 final String insertStatement = *insertIntoCars*(i);  
 writer.write(insertStatement + "\n");  
 }  
 } catch (final IOException e) {  
 System.*err*.println("Error writing to file: " + e.getMessage());  
 }  
 }  
  
 private static String insertIntoOwners(final int index) {  
 final String firstName = *FAKER*.name().firstName().replace("'", " ");  
 final String lastName = *FAKER*.name().lastName().replace("'", " ");  
 final String email = *FAKER*.internet().emailAddress();  
 final String phone = *FAKER*.phoneNumber().phoneNumber();  
 final String address = *FAKER*.address().fullAddress().replace("'", " ");  
 final String sql = String.*format*(  
 "INSERT INTO Owner (owner\_id, first\_name, last\_name, email, phone\_number, address) VALUES (%d, '%s', '%s', '%s', '%s', '%s');",  
 index, firstName, lastName, email, phone, address);  
 System.*out*.println(sql);  
 return sql;  
  
 }  
  
 private static void createOwnerScript() {  
 try (final BufferedWriter writer = new BufferedWriter(  
 new FileWriter("src/main/resources/scripts/populate/owner.sql"))) {  
 for (int i = 0; i < 5000; i++) {  
 if (i == 0) {  
 writer.write("""  
 DROP TABLE IF EXISTS Owner;  
 CREATE TABLE Owner (owner\_id INTEGER NOT NULL, first\_name TEXT NOT NULL, last\_name TEXT NOT NULL, email TEXT NOT NULL, phone\_number TEXT NOT NULL, address TEXT NOT NULL);  
 """);  
 }  
 final String insertStatement = *insertIntoOwners*(i);  
 writer.write(insertStatement + "\n");  
 }  
 } catch (final IOException e) {  
 System.*err*.println("Error writing to file: " + e.getMessage());  
 }  
 }  
  
 private static String insertIntoDealership(final int index) {  
 final String name = *FAKER*.company().name().replace("'", " ");  
 final String email = *FAKER*.internet().emailAddress();  
 final String phone = *FAKER*.phoneNumber().phoneNumber();  
 final String address = *FAKER*.address().fullAddress().replace("'", " ");  
 final String sql = String.*format*(  
 "INSERT INTO Dealership (dealership\_id, name, email, phone\_number, address) VALUES (%d, '%s', '%s', '%s', '%s');",  
 index, name, email, phone, address);  
 System.*out*.println(sql);  
 return sql;  
 }  
  
 private static void createDealershipScript() {  
 try (final BufferedWriter writer = new BufferedWriter(  
 new FileWriter("src/main/resources/scripts/populate/dealership.sql"))) {  
 for (int i = 0; i < 20; i++) {  
 if (i == 0) {  
 writer.write("""  
 DROP TABLE IF EXISTS Dealership;  
 CREATE TABLE Dealership (dealership\_id INTEGER NOT NULL, name TEXT NOT NULL, email TEXT NOT NULL, phone\_number TEXT NOT NULL, address TEXT NOT NULL);  
 """);  
 }  
 final String insertStatement = *insertIntoDealership*(i);  
 writer.write(insertStatement + "\n");  
 }  
 } catch (final IOException e) {  
 System.*err*.println("Error writing to file: " + e.getMessage());  
 }  
 }  
  
 private static String insertIntoSales(final int index) {  
 final int carId = *FAKER*.number().numberBetween(0, 10000);  
 final int dealershipId = *FAKER*.number().numberBetween(0, 20);  
 final int ownerId = *FAKER*.number().numberBetween(0, 5000);  
 final String salesDate = *FAKER*.date().between(Faker.*instance*().date().past(365 \* 2, TimeUnit.*DAYS*),  
 Faker.*instance*().date().future(365 \* 2, TimeUnit.*DAYS*)).toString();  
 final double salePrice = *FAKER*.number().randomDouble(2, 5000, 50000);  
 final String sql = String.*format*(  
 "INSERT INTO Sales (sale\_id, car\_id, owner\_id, dealership\_id, sale\_date, sale\_price) VALUES (%d, %d, %d, %d, '%s', %.2f);",  
 index, carId, dealershipId, ownerId, salesDate, salePrice);  
 System.*out*.println(sql);  
 return sql;  
 }  
  
 private static void createSalesScript() {  
 try (final BufferedWriter writer = new BufferedWriter(  
 new FileWriter("src/main/resources/scripts/populate/sales.sql"))) {  
 for (int i = 0; i < 8000; i++) {  
 if (i == 0) {  
 writer.write("""  
 DROP TABLE IF EXISTS Sales;  
 CREATE TABLE Sales (sale\_id INTEGER NOT NULL, car\_id INTEGER NOT NULL, owner\_id INTEGER NOT NULL, dealership\_id INTEGER NOT NULL, sale\_date TEXT NOT NULL, sale\_price REAL NOT NULL);  
 """);  
 }  
 final String insertStatement = *insertIntoSales*(i);  
 writer.write(insertStatement + "\n");  
 }  
 } catch (final IOException e) {  
 System.*err*.println("Error writing to file: " + e.getMessage());  
 }  
 }  
  
 private static String insertIntoService(final int index) {  
 final int carId = Faker.*instance*().number().numberBetween(1, 10000);  
 final int dealershipId = Faker.*instance*().number().numberBetween(0, 20);  
 final String serviceDate = Faker.*instance*().date().past(3, TimeUnit.*DAYS*).toString();  
 final String serviceType = Faker.*instance*().options().option("Oil Change", "Tire Rotation", "Brake Inspection");  
 final String serviceDescription = Faker.*instance*().lorem().sentence(10);  
 final double serviceCost = Faker.*instance*().number().randomDouble(2, 50, 500);  
  
 final String sql = String.*format*(  
 "INSERT INTO Service (service\_id, car\_id, dealership\_id, service\_date, service\_type, service\_description, service\_cost) "  
 +  
 "VALUES (%d, %d, %d, '%s', '%s', '%s', %.2f);",  
 index, carId, dealershipId, serviceDate, serviceType, serviceDescription, serviceCost);  
 System.*out*.println(sql);  
 return sql;  
  
 }  
  
 private static void createServiceScript() {  
 try (final BufferedWriter writer = new BufferedWriter(  
 new FileWriter("src/main/resources/scripts/populate/service.sql"))) {  
 for (int i = 0; i < 10000; i++) {  
 if (i == 0) {  
 writer.write("""  
 DROP TABLE IF EXISTS Service;  
 CREATE TABLE Service (service\_id INTEGER NOT NULL, car\_id INTEGER NOT NULL, dealership\_id INTEGER NOT NULL, service\_date TEXT NOT NULL, service\_type TEXT NOT NULL, service\_description TEXT NOT NULL, service\_cost REAL NOT NULL);  
 """);  
 }  
 final String insertStatement = *insertIntoService*(i);  
 writer.write(insertStatement + "\n");  
 }  
 } catch (final IOException e) {  
 System.*err*.println("Error writing to file: " + e.getMessage());  
 }  
 }  
  
 public static void main(final String[] args) throws IOException {  
  
 Files.*createDirectories*(Path.*of*("src/main/resources/scripts/populate/"));  
  
 *createCarScript*();  
  
 *createDealershipScript*();  
  
 *createOwnerScript*();  
  
 *createSalesScript*();  
  
 *createServiceScript*();  
 }  
}

## Trieda Constants

package com.github;  
  
import java.io.File;  
import java.nio.file.Path;  
import lombok.experimental.UtilityClass;  
  
@UtilityClass  
public class Constants {  
  
 private static final String *pwd* = String.*valueOf*(Path.*of*("").toAbsolutePath());  
 public static final Path *RESOURCES\_PATH* = Path.*of*(*pwd*, "src", "main", "resources");  
 public static final String *SQLITE\_CONNECTION\_STRING* =  
 "jdbc:sqlite:" + *RESOURCES\_PATH* + File.*separator* + "databases" + File.*separator* + "myDatabase.db";  
 public static final String *JFSQL\_CONNECTION\_STRING* =  
 "jdbc:jfsql:" + *RESOURCES\_PATH* + File.*separator* + "databases" + File.*separator* + "myDatabase";  
 public static final String[] *CONNECTION\_STRINGS* = {*JFSQL\_CONNECTION\_STRING*, *SQLITE\_CONNECTION\_STRING*};  
 public static final String *SCRIPTS\_FOLDER* = String.*valueOf*(*RESOURCES\_PATH*.resolve("scripts"));  
  
}

## Trieda DemoApplication

package com.github;  
  
import com.github.services.Delete;  
import com.github.services.Insert;  
import com.github.services.Populate;  
import com.github.services.Select;  
import com.github.services.Update;  
import java.util.Scanner;  
  
public class DemoApplication {  
  
 public static void main(final String[] args) {  
 Populate.*populate*();  
 final Scanner scanner = new Scanner(System.*in*);  
 int option;  
  
 do {  
 System.*out*.println("Please select an operation:");  
 System.*out*.println("1. INSERT");  
 System.*out*.println("2. SELECT");  
 System.*out*.println("3. UPDATE");  
 System.*out*.println("4. DELETE");  
 System.*out*.println("0. Exit");  
 System.*out*.print("Enter your choice: ");  
  
 option = scanner.nextInt();  
  
 switch (option) {  
 case 1 -> {  
 System.*out*.println("INSERT operation selected");  
 Insert.*insert*();  
 }  
 case 2 -> {  
 System.*out*.println("SELECT operation selected");  
 Select.*select*();  
 }  
 case 3 -> {  
 System.*out*.println("UPDATE operation selected");  
 Update.*update*();  
 }  
 case 4 -> {  
 System.*out*.println("DELETE operation selected");  
 Delete.*delete*();  
 }  
 case 0 -> System.*out*.println("Exiting program...");  
 default -> System.*out*.println("Invalid option! Please try again.");  
 }  
  
 System.*out*.println();  
 } while (option != 0);  
  
 scanner.close();  
 }  
}

## Skripty

Kvôli veľkosti tieto súbory sme nevložili do príloh, avšak sa nachádzajú v adresári program pod *src/main/resources/scripts*

# G: Demo aplikácia