

# A: Technická dokumentácia

## Systémové požiadavky

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

## Inštalácia

1. Stiahnite Maven:

<https://maven.apache.org/download.cgi>

Návod na inštaláciu Maven:

[https://www.tutorialspoint.com/maven/maven\\_environment\\_setup.htm](https://www.tutorialspoint.com/maven/maven_environment_setup.htm)

2. V termináli alebo v príkazovom riadku prejdite do adresára projektu a spustite nasledujúci príkaz na inštaláciu ovládača:

```
mvn clean install
```

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

3. Vo vašom programe otvorte súbor *pom.xml* 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 sa riadi štandardným formátom JDBC. Má nasledujúcu štruktúru:

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

Príklad správneho reťazca pripojenia:

```
"jdbc:jfsql:C:\\Users\\Zsolti\\Desktop\\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 zadáte niektorú hodnotu v nesprávnom tvare, bude sa používať predvolená hodnota daného atribútu.

## Vykonávanie príkazov

Ovládač funguje rovnako ako akýkoľvek iný JDBC ovládač. Existujú však určité obmedzenia, ako napríklad syntax, ktorú náš ovládač podporuje a ktorá je opísaná v časti Podporované SQL príkazy a ich formáty.

## Vytvorenie spojenia

Po inštalácii ovládača je možné vytvoriť spojenie podľa nasledujúcich pokynov:

```
private static final String URL =  
"jdbc:jfsql:C:\\Users\\Zsolti\\Desktop\\myDatabase";  
final Connection connection = DriverManager.getConnection(URL);
```

## Vytvorenie príkazu

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

```
final Statement statement = connection.createStatement();
```

## Vykonanie dotazov a spracovanie výsledkov

Po vytvorení instance Statement, je možné vykonávať dotazy (query). Existuje 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.

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

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

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

Rovnaké výsledky je možné dosiahnuť pomocou `PreparedStatement` následne:

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

## 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:

```
connection.close();
```

## 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 kombinácii 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. Toto je vec, ktorá v SQLite funguje inak.

- `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 kombinácii 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`, `<`, `<=`, `=`, `>=`, `>`.

Je dôležité spomenúť, že na správne fungovanie ovládača je lepšie používať buď `Statement` alebo `PreparedStatement` na vykonávanie príkazov, ale pri používaní `PreparedStatement` treba uviesť všetky hodnoty vo formáte placeholdera (otáznička).

Nesprávne použitie PreparedStatement je napríklad:

```
final PreparedStatement preparedStatement = connection.prepareStatement(
    "INSERT INTO myTable (id, name, age) VALUES (?, 'Zsolti', ?)");
preparedStatement.setInt(1, 1);
preparedStatement.setInt(2, 25);
preparedStatement.executeUpdate();
```

Správny tvar je:

```
final PreparedStatement preparedStatement = connection.prepareStatement(
    "INSERT INTO myTable (id, name, age) VALUES (?, ?, ?)");
preparedStatement.setInt(1, 1);
preparedStatement.setString(2, "Zsolti");
preparedStatement.setInt(3, 25);
preparedStatement.executeUpdate();
```

alebo cez Statement:

```
statement = connection.createStatement();
statement.executeUpdate(
    "INSERT INTO myTable (id, name, age) VALUES (1, 'Zsolti', 25);");
```

## 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**

Tento program slúži na porovnávanie jednotlivých serializačných formátov. Program používa Maven ako správcu závislostí a bol napísaný vo verzii Java 11. Na jeho spustenie si spustíte metódu `main` v triede `Main`. Celý program nájdete v zipovanom formáte s názvom *Serialization-test.zip*.

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

Tento program slúži na testovanie maximálneho počtu súčasne otvorených súborov. Program používa Maven ako správcu závislostí a bol napísaný vo verzii Java 11. Na jeho spustenie si spustíte metódu `main` v triede `Main`. Súbor `lorem.txt` sa musí nachádzať na správnom mieste, ak to nie je načítané z adresára `src/main/resources`. Celý program nájdete v zipovanom formáte s názvom *Max-files-test.zip*.

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

## F: Program na porovnávanie ovládača s SQLite

Tento program slúži na porovnanie nášho ovládača s SQLite z hľadiska výkonu. Program používa Maven ako správcu závislostí a bol napísaný vo verzii Java 11. Na jeho spustenie si spustíte metódu `main` v triede `Main`. Celý program nájdete v zipovanom formáte s názvom *Driver-comparison.zip*.



## **G: Program na generovanie SQL skriptov**

Tento program vygeneruje skripty obsahujúce SQL príkazy, ktoré boli použité pri porovnaní nášho ovládača s SQLite. Celý program nájdete v zipovanom formáte s názvom *SQL-script-generator.zip*.

## H: Demo aplikácia

Táto aplikácia je jednoduchá Spring bootová webová aplikácia, ktorá slúži ako integračný test pre náš ovládač. Program používa Maven ako správcu závislostí a bol napísaný vo verzii Java 17. Na jeho spustenie si spustíte metódu `main` v triede `DemoApplication`. Celý program nájdete v zipovanom formáte s názvom *JFSQL-demo.zip*.

# I: Plán práce

## Zimný semester

- 1. týždeň: Skúmanie JDBC API
- 2. týždeň: Skúmanie ľudsky čitateľných serializačných formátov
- 3. týždeň: Napísanie testov na porovnávanie ľudsky čitateľných formátov
- 4. týždeň: Skúmanie možných spôsobov uloženia údajov na súborovom systéme
- 5. týždeň: Práca na vlastnom ovládači
- 6. týždeň: Práca na vlastnom ovládači
- 7. týždeň: Práca na vlastnom ovládači
- 8. týždeň: Skúmanie rôznych metód na parsovanie SQL príkazov
- 9. týždeň: Práca na vlastnom parsery
- 10. týždeň: Skúmanie rôznych metód spracovania transakcií
- 11. týždeň: Práca na vlastnom ovládači
- 12. týždeň: Retrospektíva a úprava práce

## Letný semester

- 1. týždeň: Práca na vlastnom ovládači
- 2. týždeň: Skúmanie rôznych spôsobov uloženia LOBov
- 3. týždeň: Implementácia uloženia LOBov
- 4. týždeň: Úprava kódu, písanie testov
- 5. týždeň: Úprava kódu, písanie testov
- 6. týždeň: Vývoj aplikácie na porovnávanie ovládačov
- 7. týždeň: Vývoj demo aplikácie
- 8. týždeň: Práca na vlastnom ovládači
- 9. týždeň: Úprava kódu, pridávanie nových testov
- 10. týždeň: Testovanie ovládača
- 11. týždeň: Finalizácia práce
- 12. týždeň: Retrospektíva práce