



JDBC

OBJECT-ORIENTED PROJECT

AHMED ASHRAF ABDEL AZIZ BASHA (2).

Amr Khaled (28).

Mohamed Alaa (42).

Mustafa Mohamed(50).

Description

- ✎ Java Database Connectivity (JDBC) provides Java developers with a standard API that is used to access databases, regardless of the driver and database product.
- ✎ JDBC presents a uniform interface to databases - change vendors and your applications only need to change their driver.

Features

- ◆ In this program, you can manage databases and tables using a user-friendly Gui.
- ◆ User is supported with log showing his activity consequences.
- ◆ The program also supports Adding Batches.
- ◆ You can select any column with any condition.
- ◆ You can also update your existent data or remove them.
- ◆ You can insert any additional data under any condition.

Javafx is used to Implement the Interface.

- # Diagrams

```

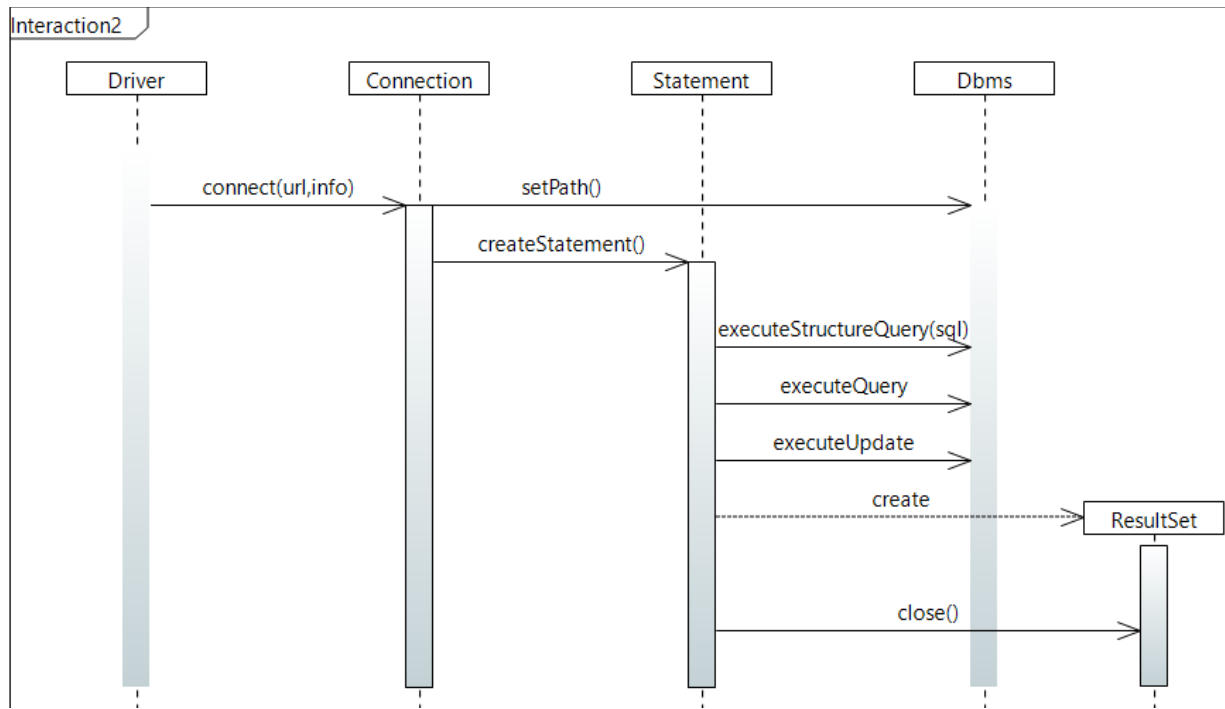
classDiagram
    class DriverImp {
        <<Java Class>>
        info: Properties
        logger: Logger
        DriverImp()
        connect(String, Properties): Connection
        acceptsURL(String): boolean
        getPropertyInfo(String, Properties): DriverPropertyInfo[]
        getMajorVersion(): int
        getMinorVersion(): int
        jdbcCompliant(): boolean
        getParentLogger(): Logger
    }
    class Connection {
        <<Java Interface>>
        +Connection
    }
    class ConnectionImp {
        <<Java Class>>
        logger: Logger
        database: Database
        info: Properties
        isConnected(): boolean
    }
    class Statement {
        <<Java Interface>>
        +Statement
    }
    class StatementImp {
        <<Java Class>>
        logger: Logger
        database: Database
        batch: List<String>
        currentResultSet: ResultSet
        isClosed(): boolean
        timeout: int
    }
    class ResultSetBuilder {
        <<Java Class>>
        db: Database
        query: String
        ResultSetBuilder()
        setStatement(Statement): void
        setDb(Database): void
        setQuery(String): void
        getQuery(String): void
        build(): ResultSet
    }
    class ResultSetImp {
        <<Java Class>>
        logger: Logger
        set: Object[][]
        cursor: int
        closeStatus: boolean
        colLabelIndex: HashMap<String, Integer>
    }
    class ResultSetMetaDatum {
        <<Java Interface>>
        +ResultSetMetaDatum
    }
    class ResultSetMetaDatumImp {
        <<Java Class>>
        logger: Logger
        tableName: String
        colName: String[]
        colType: String[]
    }
    class Log {
        <<Java Class>>
        logger: Logger
        fh: FileHandler
        Log()
        getLogeer(): Logger
    }

    DriverImp ..> ConnectionImp
    ConnectionImp ..|> Connection
    StatementImp ..|> Statement
    StatementImp ..> Statement : -statement 0..1
    StatementImp ..> ResultSetBuilder : -statement 0..1
    StatementImp ..> ResultSetImp
    ResultSetBuilder ..> ResultSetImp
    ResultSetImp ..> ResultSetMetaDatum : -metaData 0..1
    ResultSetMetaDatumImp ..|> ResultSetMetaDatum
    Log ..> ConnectionImp
    Log ..> StatementImp
    Log ..> ResultSetImp
    Log ..> Log : -SingleLog 0..1
  
```

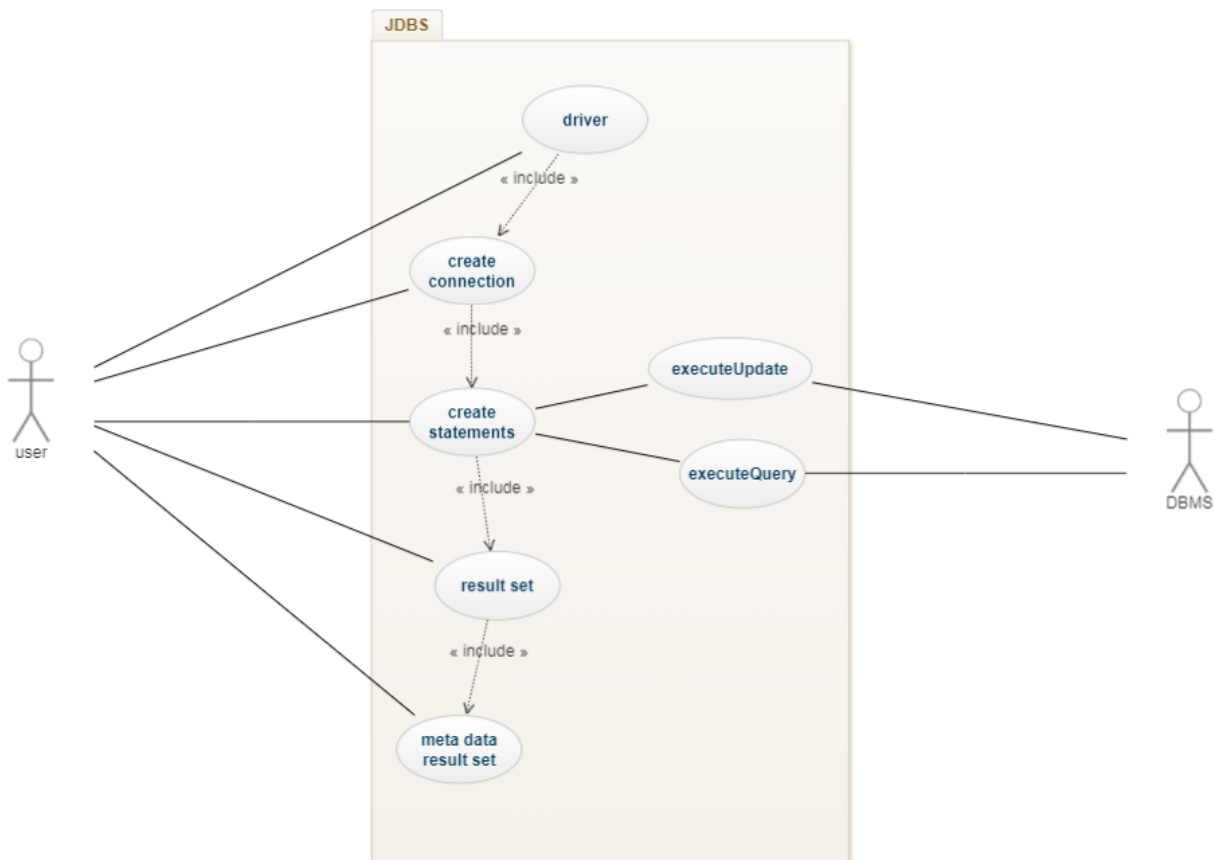
The diagram illustrates the JDBC API structure with the following components and relationships:

- DriverImp** (Java Class): Implements `Driver`. It manages database connections and logging. It has properties `info: Properties` and `logger: Logger`. Methods include `DriverImp()`, `connect(String, Properties): Connection`, `acceptsURL(String): boolean`, `getPropertyInfo(String, Properties): DriverPropertyInfo[]`, `getMajorVersion(): int`, `getMinorVersion(): int`, `jdbcCompliant(): boolean`, and `getParentLogger(): Logger`.
- Connection** (Java Interface): Represents a database connection. It has a method `Connection`.
- ConnectionImp** (Java Class): Implements `Connection`. It has properties `logger: Logger`, `database: Database`, `info: Properties`, and `isClosed(): boolean`.
- Statement** (Java Interface): Represents a database statement. It has a method `Statement`.
- StatementImp** (Java Class): Implements `Statement`. It has properties `logger: Logger`, `database: Database`, `batch: List<String>`, `currentResultSet: ResultSet`, `isClosed(): boolean`, and `timeout: int`. It has associations with `Statement` (labeled `-statement 0..1`) and `ResultSetBuilder` (labeled `-statement 0..1`).
- ResultSetBuilder** (Java Class): Builds `ResultSet` objects. It has properties `db: Database` and `query: String`. Methods include `ResultSetBuilder()`, `setStatement(Statement): void`, `setDb(Database): void`, `setQuery(String): void`, `getQuery(String): void`, and `build(): ResultSet`.
- ResultSetImp** (Java Class): Implements `ResultSet`. It has properties `logger: Logger`, `set: Object[][]`, `cursor: int`, `closeStatus: boolean`, and `colLabelIndex: HashMap<String, Integer>`. It has an association with `ResultSetMetaDatum` (labeled `-metaData 0..1`).
- ResultSetMetaDatum** (Java Interface): Represents metadata for a `ResultSet`. It has a method `ResultSetMetaDatum`.
- ResultSetMetaDatumImp** (Java Class): Implements `ResultSetMetaDatum`. It has properties `logger: Logger`, `tableName: String`, `colName: String[]`, and `colType: String[]`.
- Log** (Java Class): Manages logging. It has properties `logger: Logger` and `fh: FileHandler`. Methods include `Log()` and `getLogeer(): Logger`. It has associations with `ConnectionImp`, `StatementImp`, `ResultSetImp`, and itself (labeled `-SingleLog 0..1`).

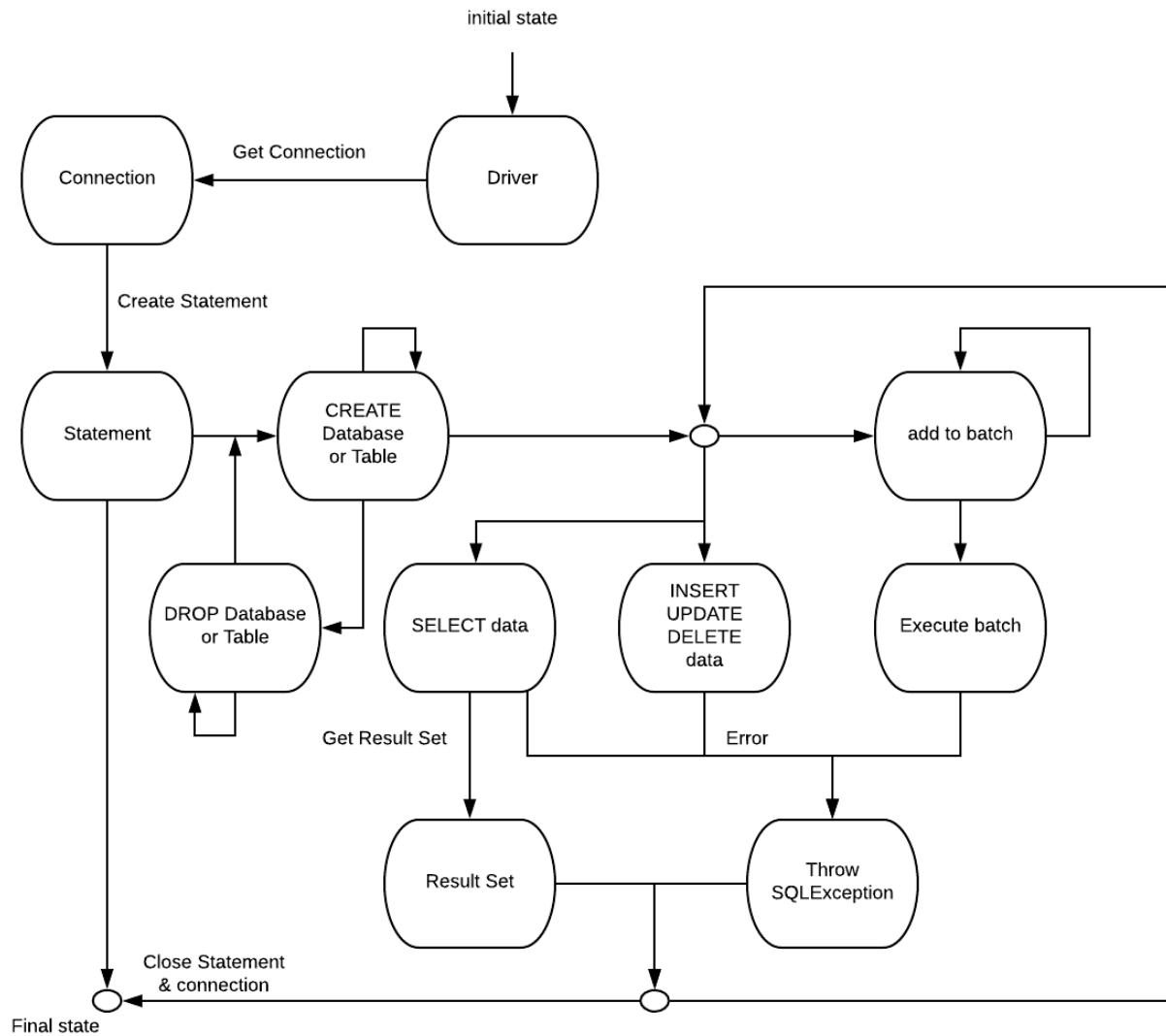
Sequence Diagram :



Use Case :



Class Diagram



division of labor among group members

1)Ahmed Ashraf, Amr Khaled :
ResultSet,ResultSetMetaData,
Timer, UI.

3)Mohamed Alaa, Mustafa mohamed :
Driver, Connection, Statement,
Logger, Diagrams .

Team Collaborated in editing
and updating every single
function in the project as it's
shown in bitbucket commits.

User Manual



JDBC

console

Invalid Query!!
cdcsd
Invalid Query!!
sdvsdfv
couldn't excute the query due to error in file system
CREATE TABLE table_name12(column_name1 varchar, column_

create database x

Logging

Dec 11, 2018 11:52:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.D
INFO: connecting
Dec 11, 2018 11:52:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.D
INFO: checking the URL
Dec 11, 2018 11:52:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.C
INFO: Creating statement

JDBC

• Select any database from your current driver, then start working.

add batch

ADD

JDBC

console

INSERT INTO table_name12(column_name1, COLUMN_NAME

INSERT INTO table_name12(column_NAME1, COLUMN_name

Invalid Query!!
Invalid Query!!
cdcsd
Invalid Query!!
sdvsdfv
couldn't excute the query due to error in file system
CREATE TABLE table_name12(column_name1 varchar, colum

Logging

Dec 11, 2018 11:52:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.D
INFO: connecting
Dec 11, 2018 11:52:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.D
INFO: checking the URL
Dec 11, 2018 11:52:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.C
INFO: Creating statement

JDBC

• Select any database from your current driver, then start working.

add batch

ADD

SELECT * From table_name12

JDBC

console

```
INSERT INTO table_name12(column_name1, COLUMN_NAME
INSERT INTO table_name12(column_NAME1, COLUMN_name
Invalid Query!!
Invalid Query!!
cdcsd
Invalid Query!!
sdvsdfv
couldn't excute the query due to error in file system
CREATE TABLE table_name12(column_name1 varchar, colum
```

Logging

```
INFO: Creating statement
Dec 11, 2018 11:54:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.S
INFO: adding query to batch : SELECT * From table_name12
Dec 11, 2018 11:54:59 PM eg.edu.alexu.csd.oop.jdbc.cs50.S
SEVERE: NOT INSERT or UPDATE Query : SELECT * From ta
```

JDBC

- Select any database from your current driver, then start working.

add batch

ADD

```
SELECT * From table_name12
```