CHAPTER 03: Persistence and Databases

By: Muleta Taye T.

E-mail: muleta.taye@aastu.edu.et

April 25, 2023

CH03: Topics to be covered

- Database ?
  - is an $\underbrace{\textit{organized}}$ collection of data!

    $\downarrow$

    To facilitate easy access and manipulation $\mapsto$ DBMS

    $\downarrow$

    - provides mechanisms for storing, organizing, retrieving and modifying
  - Popular Database System
    - Recap Relational databases!
      - Microsoft SQL Server,
      - Oracle,
      - Sybase,
      - Informix,
      - PostgreSQL
      - MySQL.

      How java Programs Communicate ?

CH03: Persistence and Databases

- Java provides $\Rightarrow$ JDBC API
  - stands for Java Database connectivity
  - allows Java programs to access database management systems.
  - java.sql package contains **classes** and **interfaces** for JDBC API.
  - JDBC API uses JDBC drivers to connect with the database.
    1. Type 1 driver
       - aka JDBC-ODBC bridge driver

         $\downarrow$

         converts JDBC method calls into the ODBC function calls
    2. Type 2 driver
       - aka Native-API driver

         $\downarrow$

         converts JDBC method calls into the native calls of the database API.
    3. Type 3 driver
       - aka Network Protocol driver

         $\downarrow$

         converts JDBC method calls directly or indirectly into the vendor-specific database protocol.
    4. Type 4 driver
       - aka Thin driver

         $\downarrow$

         converts JDBC method calls converts JDBC calls directly into the vendor-specific database protocol.

CH03: Persistence and Databases

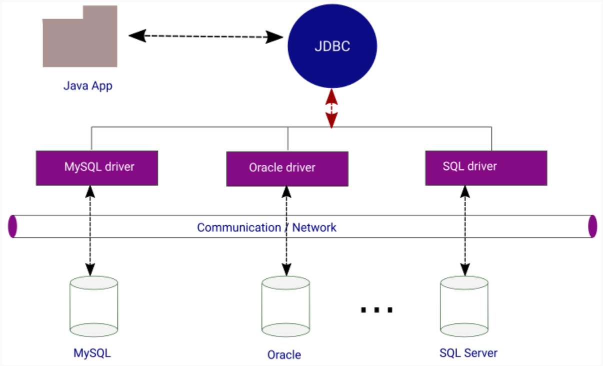

Figure 1: Java JDBC Architecture!

- Two Packages
  - java.sql: an API to access and process the data stored in a database
  - javax.sql: an API for the server side for accessing and processing the data from the databases

CH03: Persistence and Databases

- What does it the packages API provide!
  1. The DriverManager facility
     - Where a connection would be made via.
     - DriverManager class $\mapsto$ makes a connection with a driver.
     - SQLPermission class $\mapsto$ provides permission when code running within a Security Manager.
     - Driver interface $\mapsto$ provides the API for registering and connecting drivers based on JDBC technology.
     - DriverPropertyInfo class $\mapsto$ provides properties for a JDBC driver; not used by the general user

CH03: Persistence and Databases

- What does it the packages API provide!
  1. The DriverManager facility
     - Where a connection would be made via.
     - DriverManager class $\mapsto$ makes a connection with a driver.
     - SQLPermission class $\mapsto$ provides permission when code running within a Security Manager.
     - Driver interface $\mapsto$ provides the API for registering and connecting drivers based on JDBC technology.
     - DriverPropertyInfo class $\mapsto$ provides properties for a JDBC driver; not used by the general user
  2. Sending SQL statements to a database
     - Statement $\mapsto$ used to send basic SQL statements
     - PreparedStatement $\mapsto$ used to send prepared statements or basic SQL statements
     - CallableStatement $\mapsto$ used to call database stored procedures (derived from PreparedStatement)
     - Connection interface $\mapsto$ provides methods for creating statements and managing connections and their properties
     - Savepoint $\mapsto$ provides savepoints in a transaction

CH03: Persistence and Databases

- java.sql package provision ....

  3 Retrieving and updating the results of a query
    - provides ResultSet interface

  4 Standard mappings for SQL types to classes and interfaces
    - Array interface $\mapsto$ mapping for SQL ARRAY
    - Blob interface $\mapsto$ mapping for SQL BLOB
    - Clob interface $\mapsto$ mapping for SQL CLOB
    - Date class $\mapsto$ mapping for SQL DATE
    - NClob interface $\mapsto$ mapping for SQL NCLOB

      $\downarrow$

      Many more ...

  5 Custom mapping an SQL user-defined type (UDT) to a class
    - SQLData interface $\mapsto$ specifies the mapping of a UDT to an instance of this class
    - SQLInput interface $\mapsto$ provides methods for reading UDT attributes from a stream
    - SQLOutput interface $\mapsto$ provides methods for writing UDT attributes back to a stream

CH03: Persistence and Databases

- java.sql package provision ....

  6 Exceptions

  - SQLException
    - thrown by most methods when there is a problem accessing data and by some methods for other reasons
  - SQLWarning
    - thrown to indicate a warning
  - DataTruncation $\mapsto$
    - thrown to indicate that data may have been truncated
  - BatchUpdateException
    - thrown to indicate that not all commands in a batch update executed successfully

CH03: Persistence and Databases

- java.sql package provision ....

  6 Exceptions

  - SQLException
    - thrown by most methods when there is a problem accessing data and by some methods for other reasons
  - SQLWarning
    - thrown to indicate a warning
  - DataTruncation $\mapsto$
    - thrown to indicate that data may have been truncated
  - BatchUpdateException
    - thrown to indicate that not all commands in a batch update executed successfully

- using JDBC to connect and manage a database has 5 steps.
  - Register the Driver class
  - Create connection
  - Create statement
  - Execute queries
  - Close connection

CH03: Persistence and Databases

- Register the Driver class
  - is the process by which the RDBMS driver's class file is loaded into the memory
  - can be done in 2 ways
    - Approach-1 $\mapsto$ Class.forName() method
      - dynamically load the driver's class file into memory.
      - preferable
        ∟ it allows you to make the driver registration configurable and portable.

```
import java.sql.*
try {
   Class.forName("driver name");
   // oracle.jdbc.driver.OracleDriver
   // com.mysql.jdbc.Driver
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

    - Approach-2: DriverManager.registerDriver() method

```
try {
      Driver driverObjectName = new drivername; //oracle.jdbc.driver.OracleDriver()
      DriverManager.registerDriver( driverObjectName );
}
catch(ClassNotFoundException ex) {
    System.out.println("Error: unable to load driver class!");
    System.exit(1);
}
```

CH03: Persistence and Databases

- Register the Driver class ...
  - Database URL Formulation
    - is an address that points to your database.

      $\downarrow$

      achieved $\mapsto$ DriverManager.getConnection()
    - Syntax:

      ```
      DriverManager.getConnection(String url);
      ```
    - URL format
      - jdbc:mysql://hostname/databaseName
      - jdbc:oracle:thin:@hostname:port Number:databaseName
- Creating Connection
  - DriverManager.getConnection() method to create a connection object.
    - 3 overloaded getConnection() methods
      - getConnection(String url)
        - The database URL includes the username and password

          ```
          import java.sql.*;
          String url = jdbc:oracle:driver:username/password@database;
          Connection conn = DriverManager.getConnection(url);
          ```

CH03: Persistence and Databases

- 3 overloaded getConnection() methods ....
  - getConnection(String url, String Properties)
    - A Properties object holds a set of keyword-value pairs.

      $\downarrow$

      "It is used to pass driver properties to the driver during a call to the getConnection() method".
    - Syntax:

```
import java.util.*;
String url = "database url";
Properties propertiesObject = new Properties( );
propertiesObject.put( "key", "value" );
// propertiesObject.put( "user", "username" );
// propertiesObject.put( "password", "password" );

Connection conn = DriverManager.getConnection(url, propertiesObject);
```

  - getConnection(String url, String username, String password)
    - Takes three different parameters
    - Syntax

```
import java.sql.*;
String url = "driver url";
String username = "username";
String password = "password";
Connection conn = DriverManager.getConnection(url, username, password);
```

CH03: Persistence and Databases

- Connection Interface
  - is a session between a Java application and a specific database.
  - SQL statements are **executed** and **results are returned** within **the context of a connection**.
  - Connection object $\mapsto$ provide info. about tables.

CH03: Persistence and Databases

- Connection Interface
  - is a session between a Java application and a specific database.
  - SQL statements are **executed** and **results are returned** within **the context of a connection**.
  - Connection object $\mapsto$ provide info. about tables.

Methods	Description
public Statement createStatement()	creates a statement object that can be used to execute SQL queries
public Statement createStatement(int resultSetType,int resultSetConcurrency)	Creates a Statement object that will generate ResultSet objects with the given type and concurrency.
public void setAutoCommit(boolean status)	is used to set the commit s status. By default it is true.
public void commit()	saves the changes made since the previous commit/rollback permanent
public void rollback()	Drops all changes made since the previous commit/rollback.
public void close()	closes the connection and releases JDBC resources immediately.

$\uparrow$

Connection Interface Commonly used Methods

- How to do ? $\mapsto$ Steps ...
  - First create a connection object using DriverManager.getConnction() API.
  - use Connection.createStatement() method to create Statement object:

```
import java.sql.*;
// Creating a Connection
try(
    Connection connection = DriverManager.getConnection("url", "username", "password");
       #Step 2:Create a statement using connection object
    Statement statement = connection.createStatement();
  ){
       #Step 3: Execute the query or update query
       statement.execute(Query);
 }catch (SQLException e) {

 }
```

  - (3) methods for the transaction management
    - setAutoCommit()
    - commit()
    - rollback()

CH03: Persistence and Databases

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class InsertPStatementExample {
     private static final String INSERT_DATA = "INSERT INTO sectionAB" + "(id, name, email) VALUES " + " (?, ?, ?);";

       public void insertRecord() throws SQLException {
         System.out.println(INSERT_DATA);
         // Step 1: Establishing a Connection
         try (Connection connection = DriverManager
            .getConnection("jdbc:mysql://localhost:3306/advancedDB, "jdbcExample", "root"); {
                    connection.setAutoCommit(false);
                     try (PreparedStatement insertUserData = connection.prepareStatement(INSERT_DATA) {
                           insertUserData.setInt(1,001);
                           insertUserData.setString(2, "Adavanced Programming");
                           insertUserData.setString(3, adnaved@aastu.edu.et");

                           insertUserData.executeUpdate();

                           // you can commit the changes
                           connection.commit();
                           System.out.println("Transaction is commited successfully.");
                     }catch (SQLException e) {
                             printSQLException(e);
                             if (connection != null) {
                             try {
                                // STEP 3 - Roll back transaction
                                System.out.println("Transaction is being rolled back.");
                                connection.rollback();
                             } catch (Exception ex) {
                                 ex.printStackTrace();
                    }
                }
            }
```

CH03: Persistence and Databases

- Create statement
  - Interaction with DB starts!
  - JDBC defines three interfaces
    1. Statement
       - For general-purpose access to your database
       - The Statement interface cannot accept parameters.
       - First need to create one using the Connection object's createStatement() method
       - Syntax:

```
import java.sql.*;
Statement stmtObj = null;
try {
     Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
     stmtObj = conn.createStatement();
       // Your pgm logic
}catch (SQLException e) {
     // Your Exception logic
}finally {
stmtObj.close();
       // code that execute regardless of anything
}
```

$\rightarrow$ use Statement Object to execute

- boolean execute (String SQL) $\mapsto$ true if a ResultSet object can be retrieved;
- int executeUpdate (String SQL) $\mapsto$ Returns the number of rows affected by the execution of the SQL statement

CH03: Persistence and Databases

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class CreateStatementExample {

    private static final String createTableSQL =
      "create table sectionAB(\r\n" + "id  int(3) primary key, \r\n" +
         "name varchar(20),\r\n" + "email varchar(20),\r\n");";
       public void createTable() throws SQLException {
         System.out.println(createTableSQL);
         // Step 1: Establishing a Connection
         try (Connection connection = DriverManager
            .getConnection("jdbc:mysql://localhost:3306/mysql_database", "jdbcExample", "pass");

             // Step 2:Create a statement using connection object
             Statement statement = connection.createStatement();) {

             // Step 3: Execute the query or update query
             statement.execute(createTableSQL);
         } catch (SQLException e) {
             // print SQL exception information
             printSQLException(e);
         } finally {
         statement.close();
         }
          public static void main(String[] args) throws SQLException {
                 CreateStatementExample createTableExample = new CreateStatementExample();
                 createTableExample.createTable();
          }
     }
```

CH03: Persistence and Databases

- JDBC defines three interfaces ...
    - Statement ...
        - ResultSet executeQuery (String SQL) $\mapsto$ Executes the given SQL statement, which returns a single ResultSet object.
    - PreparedStatement
        - is a subinterface of Statement.
        - It is used to execute a parameterized query.
        - Basically used for sending SQL statements to the database

Methods	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used to create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

- CallableStatement

CH03: Persistence and Databases

- JDBC defines three interfaces ...
    - Statement ...
        - ResultSet executeQuery (String SQL) $\mapsto$ Executes the given SQL statement, which returns a single ResultSet object.
    - PreparedStatement
        - is a subinterface of Statement.
        - It is used to execute a parameterized query.
        - Basically used for sending SQL statements to the database

Methods	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used to create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

- CallableStatement
    - Check it ...

CH03: Persistence and Databases

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class InsertPStatementExample {
     private static final String INSERT_DATA = "INSERT INTO sectionAB" +
        "(id, name, email) VALUES " + " (?, ?, ?);";

       public void insertRecord() throws SQLException {
         System.out.println(INSERT_DATA);
         // Step 1: Establishing a Connection
         try (Connection connection = DriverManager
            .getConnection("jdbc:mysql://localhost:3306/mysql_database, "jdbcExample", "root");

            // Step 2:Create a statement using connection object
            PreparedStatement preparedStatement = connection.prepareStatement(INSERT_DATA)) {
            preparedStatement.setInt(1, 1);
            preparedStatement.setString(2, "Abebe");
            preparedStatement.setString(3, "abebe.kebede.aastu.edu.et");

            System.out.println(preparedStatement);
            // Step 3: Execute the query or update query
            preparedStatement.executeUpdate();
        } catch (SQLException e) {
            // print SQL exception information
            printSQLException(e);
        }
    }
    public static void main(String[] args) throws SQLException {
        InsertPStatementExample insertDataObject = new InsertPStatementExample();
        insertDataObject.insertRecord();
    }
```

CH03: Persistence and Databases

- ResultSet Interface
  - interface provides methods for retrieving and manipulating the results of executed queries,
  - ResultSet object maintains a cursor that points to the current row in the result set.
  - has three core X-stics:
    - Type: the type of a ResultSet object
    - concurrency: ResultSet constants (read-only or updatable)
  - Cursor
    - is a temporary work area created in the system memory when a SQL statement is executed.
    - contains information on a select statement and the rows of data accessed by it.
    - is movable based on the $\underbrace{properties of the ResultSet}$

      $\downarrow$

      set when Statement that generates the ResultSet is created.

      $\swarrow$ JDBC provides connection methods

      createStatement(int RSType, int RSConcurrency);
      prepareStatement(String SQL, int RSType, int RSConcurrency);
      prepareCall(String sql, int RSType, int RSConcurrency);

CH03: Persistence and Databases

- Type of ResultSet

Type	Description
ResultSet.TYPE_FORWARD_ONLY	The cursor can only move forward in the result set.
ResultSet.TYPE_SCROLL_INSENSITIVE	The cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.
ResultSet.TYPE_SCROLL_SENSITIVE	The cursor can scroll forward and backward, and the result set is sensitive to changes made by others to the database that occur after the result set was created.

- possible RSConcurrency

Type	Description
ResultSet.CONCUR_READ_ONLY	Default!. Creates a read-only result set.
ResultSet.CONCUR_UPDATABLE	Creates an updateable result set.

- Syntax:

```
try {
   Statement stmt = connnection.createStatement(ResultSet.TYPE_FORWARD_ONLY, ResultSet.CONCUR_READ_ONLY);
   PreparedStatement stmt = connnection.PreparedStatement(String SQL, ResultSet.TYPE_FORWARD_ONLY,
    ResultSet.CONCUR_READ_ONLY);
}
catch(Exception ex) { ....}
finally { .... }
```

CH03: Persistence and Databases

- ResultSet Interface ...
    - ResultSet interface provides that involve moving the cursor
        - public void beforeFirst() throws SQLException
          - To moves the cursor just before the first row.
        - public void afterLast() throws SQLException
          - To moves the cursor just after the last row.
        - public boolean first() throws SQLException
          Moves the cursor to the first row.
        - public void last() throws SQLException • Moves the cursor to the last row.
        - public boolean absolute(int row) throws SQLException
          - Moves the cursor to the specified row.
        - public boolean relative(int row) throws SQLException
          - Moves the cursor the given number of rows forward or backward, from where it is currently pointing.
        - public boolean previous() throws SQLException
          - Moves the cursor to the previous row. This method returns false if the previous row is off the result set

CH03: Persistence and Databases

- DataBase CRUD Operations
  - Before executing sql:
    - Import the packages $\mapsto$ import java.sql.*;
    - Open a connection $\mapsto$ DriverManager.getConnection() method
    - Open a connection
    - Clean up the environment
  - Create query
    - Creating database

```
import java.sql.*;
try(Connection jdbcConnect = DriverManager.getConnection(DB_URL, USER, PASS);
          Statement execute = jdbcConnect.createStatement();
      ) {
         String query = "CREATE DATABASE DATABASE_NAME ";
         execute.executeUpdate(query);
         System.out.println("Database created successfully...");
      } catch (SQLException e) {
         e.printStackTrace();
      }
```

    - Creating Table

$$\downarrow$$

Recall Tables!

CH03: Persistence and Databases

- Creating Table ...

```
import java.sql.*;

// inside main method
static final String DB_URL = "jdbc:mysql://hostname/DATABASE_NAME";
static final String USERNAME = "username";
static final String PASSWORD = "password";

try(Connection jdbcConnect = DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
    Statement stmt = jdbcConnect.createStatement();
) {
          String query = "CREATE TABLE TABLE_NAME " +
                   "(ATT_1 INTEGER not NULL, " +
                   " ATT_2 VARCHAR(255), " +
                   " ATT_3 VARCHAR(255), " +
                   " ATT_4 INTEGER, " +
                   " PRIMARY KEY ( ATT_1 ))";

         stmt.executeUpdate(query);
         System.out.println("Created table in given database...");
      } catch (SQLException e) {
         e.printStackTrace();
      }
```

- Inserting Data to Table $\mapsto$ Same except the Query

```
import java.sql.*;
...
...
String query = "INSERT INTO table_name VALUES ()";
Statement.executeUpdate(query);
```

- Reading Data ...

```
import java.sql.*;

// Inside main method
static final String DB_URL = "jdbc:mysql://hostname/DATABASE_NAME";
static final String USERNAME = "username";
static final String PASSWORD = "password";
static final String QUERY = "SELECT attributes FROM table_name";
static final String QUERY_with_CONDITION = "SELECT attributes FROM table_name WHERE condition";
try(Connection jdbcConnect = DriverManager.getConnection(DB_URL, USER, PASS);
          Statement stmt = jdbcConnect.createStatement();
          ResultSet rsData = stmt.executeQuery(QUERY);
      ) {
         while(rsData.next()){
            //Display values
            System.out.print("ATT_1: " + rsData.getInt("ATT_1"));
            System.out.print(", ATT_2: " + rsData.getString("ATT_2"));
         }
         rsData.close();
      } catch (SQLException e) {
         e.printStackTrace();
      }
```

- Updating Existing Data ...

```
import java.sql.*;
...
try(Connection jdbcConnect = DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
          Statement execute = jdbcConnect.createStatement();
      ) {
         String updateQuery = "UPDATE table_name " + "SET value_to_set WHERE condition";
         execute.executeUpdate(updateQuery);

      } catch (SQLException e) {
         e.printStackTrace();
      }
```

- Removing Existing Data ...

```
import java.sql.*;
...
try(Connection jdbcConnect = DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
          Statement execute = jdbcConnect.createStatement();
      ) {
         String deleteQuery = "DELETE FROM table_name " + "WHERE condition";
         execute.executeUpdate(deleteQuery);

      } catch (SQLException e) {
         e.printStackTrace();
      }
```

CH03: Persistence and Databases

- Removing Tables / Database ...

```
import java.sql.*;
...
try(Connection jdbcConnect = DriverManager.getConnection(DB_URL, USERNAME, PASSWORD);
          Statement execute = jdbcConnect.createStatement();
      ) {
        String dropDBsql = "DROP DATABASE database_name";
        String dropTableSql = "DROP TABLE table_name";
        execute.executeUpdate(dropDBsql);
        execute.executeUpdate(dropTablesql);

      } catch (SQLException e) {
         e.printStackTrace();
      }
```

CH03: Persistence and Databases

- Query Result and Metadata.
  - Recall ResultSet Interface $\mapsto$ for handling QuerySet.
  - Metadata
    - Def^n $\mapsto$ data about data!
    - JDBC provides $\mapsto$ DatabaseMetaData Interface
    - DatabaseMetaData $\mapsto$ has methods
      - database product name
      - database product version,
      - DBMS driver name,
      - name of a total number of tables,
      - A name of a total number of views.
      - more ...

CH03: Persistence and Databases

- Query Result and Metadata.
  - Recall ResultSet Interface $\mapsto$ for handling QuerySet.
  - Metadata
    - Def^n $\mapsto$ data about data!
    - JDBC provides $\mapsto$ DatabaseMetaData Interface
    - DatabaseMetaData $\mapsto$ has methods
      - database product name
      - database product version,
      - DBMS driver name,
      - name of a total number of tables,
      - A name of a total number of views.
      - more ...

Method	Description
public String getDriverName() throws SQLException	it returns the name of the JDBC driver.
public String getDriverVersion() throws SQLException	it returns the version number of the JDBC driver.
public String getUserName() throws SQLException	it returns the username of the database.
public String getDatabaseProductName() throws SQLException	it returns the product name of the database.
public String getDatabaseProductVersion() throws SQLException	it returns the product version of the database..

CH03: Persistence and Databases

```
import java.sql.*; // assume we are using mysql dbms
public class DatabaseMetaDataExample {
    public static void main(String[] args) {
        databaseMetaData();
    }
    private static void databaseMetaData() {
        // Step 1: Establishing a Connection
        try (Connection connection = DriverManager
            .getConnection("jdbc:mysql://localhost:3306/mysql_database", "username", "password")) {
            DatabaseMetaData dbmd = connection.getMetaData();
            System.out.println("Driver Name: " + dbmd.getDriverName());
            System.out.println("Driver Version: " + dbmd.getDriverVersion());
            System.out.println("UserName: " + dbmd.getUserName());
            System.out.println("Database Product Name: " + dbmd.getDatabaseProductName());
            System.out.println("Database Product Version: " + dbmd.getDatabaseProductVersion());

        } catch (SQLException e) {
            printSQLException(e);
        }
    }
    public static void printSQLException(SQLException ex) {
        for (Throwable e: ex) {
            if (e instanceof SQLException) {
                e.printStackTrace(System.err);
                System.err.println("SQLState: " + ((SQLException) e).getSQLState());
                System.err.println("Error Code: " + ((SQLException) e).getErrorCode());
                System.err.println("Message: " + e.getMessage());
                Throwable t = ex.getCause();
                while (t != null) {
                    System.out.println("Cause: " + t);
                    t = t.getCause();
                }
            }
        }
    }
```

Done!

Question!