



# Lesson 29

28.10.2024



5 java questions

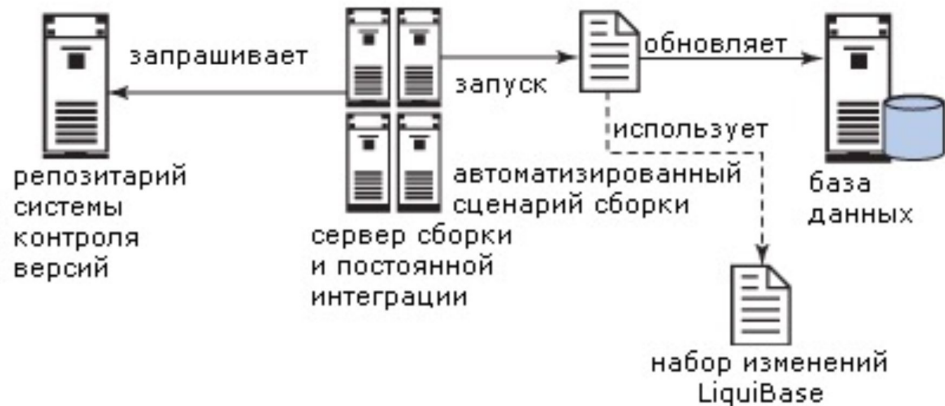
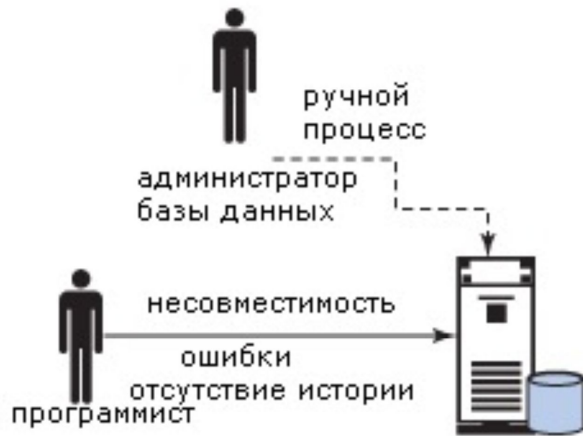


1. Відмінності String/StringBuilder/StringBuffer
2. Interface vs Abstract Class
3. override vs overload
4. Регіони пам'яті в JVM
5. java.util.collection.

## Міграція баз даних

Розробка програмного забезпечення призводить до таких проблем:

- ручне внесення змін в БД;
- різні версії БД у різних учасників команди розробників;
- непослідовні підходи до внесення змін (в базу даних або дані);
- неефективні механізми ручного управління змінами при переходах між версіями баз даних.





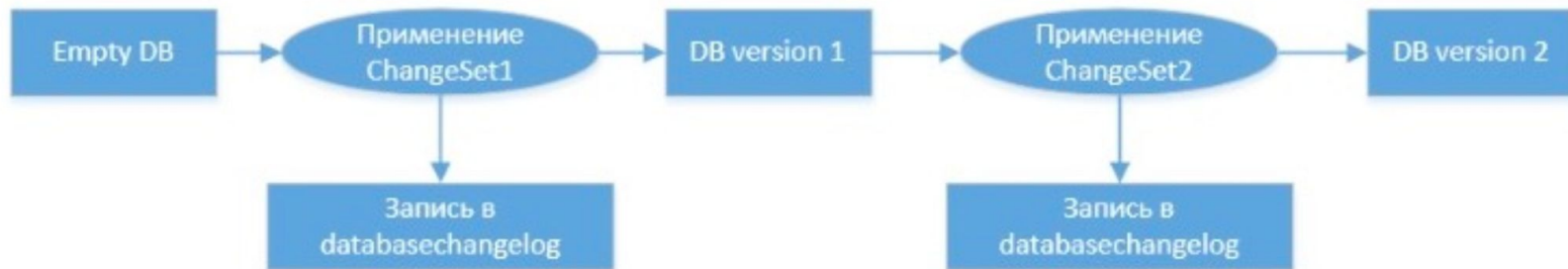
Liquibase




Flyway

## Управління змінами в базі даних за допомогою LiquiBase ([www.liquibase.org](http://www.liquibase.org))

**Liquibase** — це незалежна від бази даних бібліотека для відстеження, керування та застосування змін схеми бази даних. Для того, щоб внести зміни в БД, створюється файл міграції (\*changeset\*), який включається в головний файл (\*changeLg\*), який контролює версії та керує всіма змінами. У якості опису структури та змін бази даних використовуються формати XML, YAML, JSN і SQL.



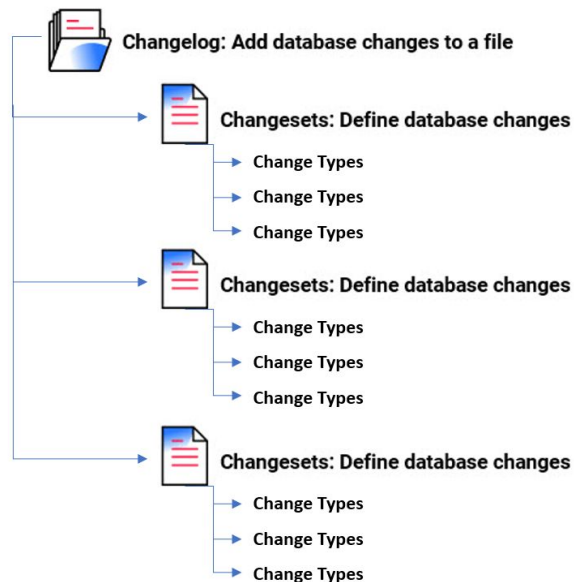


## Основні функціональні можливості:

- Оновлення бази даних до поточної версії
- Відкат останніх змін у базі даних / на визначену дату / час / тега
- SQL для оновлення бази даних і відкатів можна зберегти для ручного перегляду
- «Контексти» для включення / виключення набору змін для виконання
- Отчет о различиях баз данных
- Генерація змін в базі даних
- Можливість створення журналу змін для створення існуючої бази даних
- Створення документації по змінам бази даних
- Перевірка СУБД, перевірка користувача та попередні умови перевірки SQL
- Можливість розбивати журнал змін на кілька файлів для посилення управління
- Виконується через командну строку, Apache Ant, Apache Maven, контейнер сервлетів або Spring Framework

## Щоб розпочати роботу з LiquiBase, потрібно виконати чотири кроки:

- Створити файл із журналом змін у базі даних (**change log**).
- визначити набір змін (**change set**) у цьому файлі.
- Призначити набір змін до бази даних через командну строку або сценарій збірки.
- Перевірити зміни в базі даних.



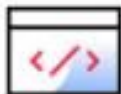




**Changesets: Define database changes**



**Changelog: Add database changes to a file**



**Update: Run the command to deploy database changes**

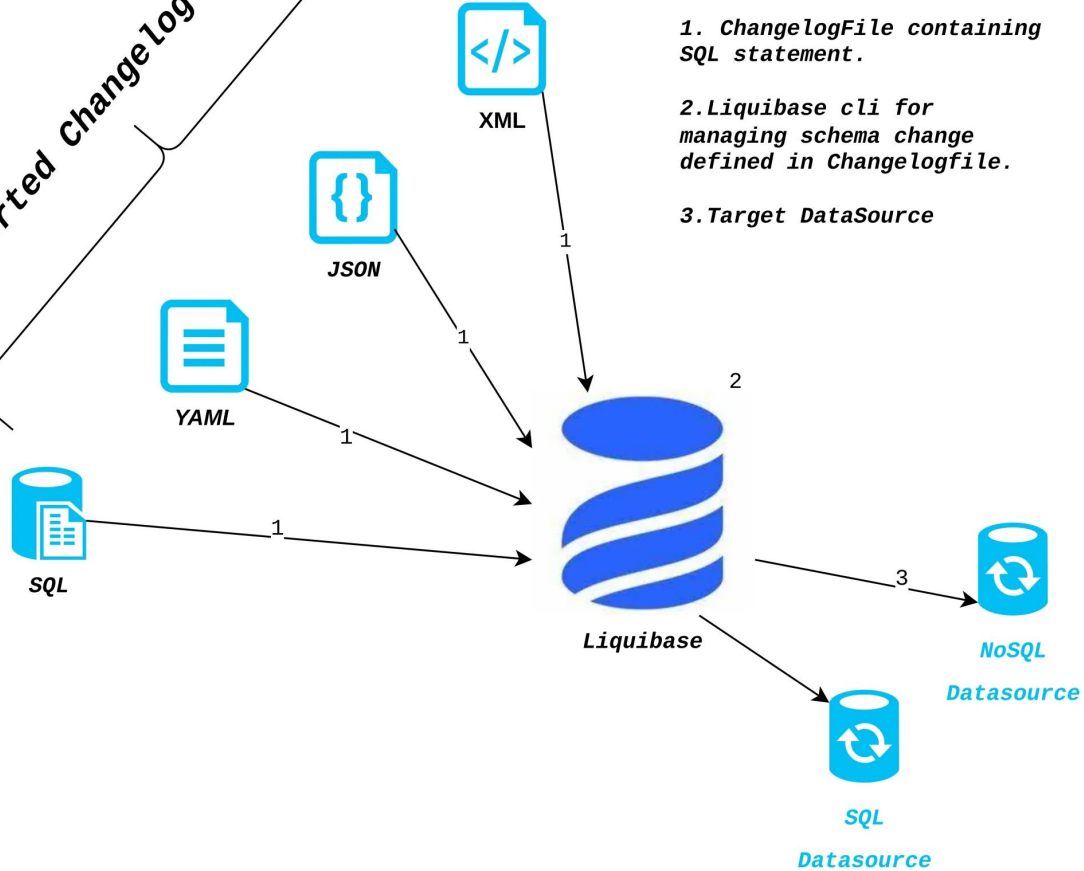


**DATABASECHANGELOG and DATABASECHANGELOGLOCK:  
Track and version database changes**

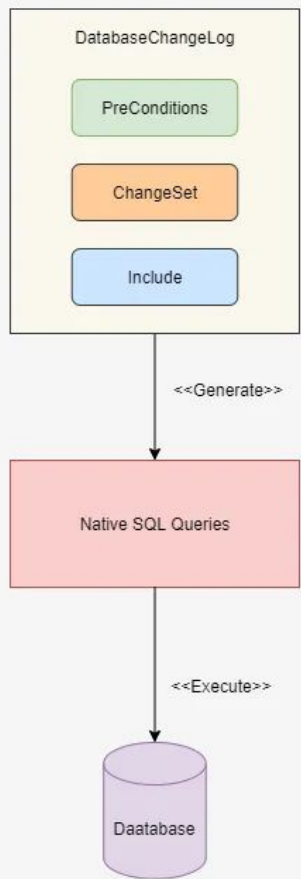


**Manage your database changes with other Liquibase commands**

# Supported Changelog format

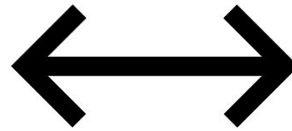


# How Liquibase Works

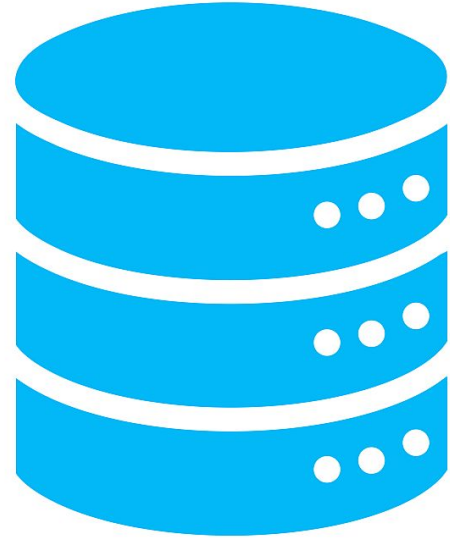




**Java**

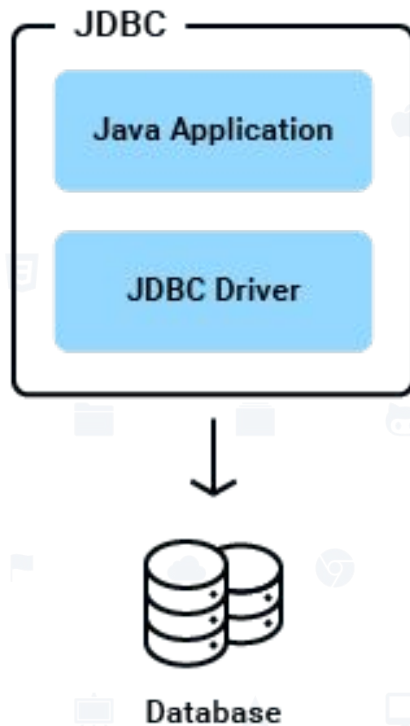


**JDBC**



**RDBMS**

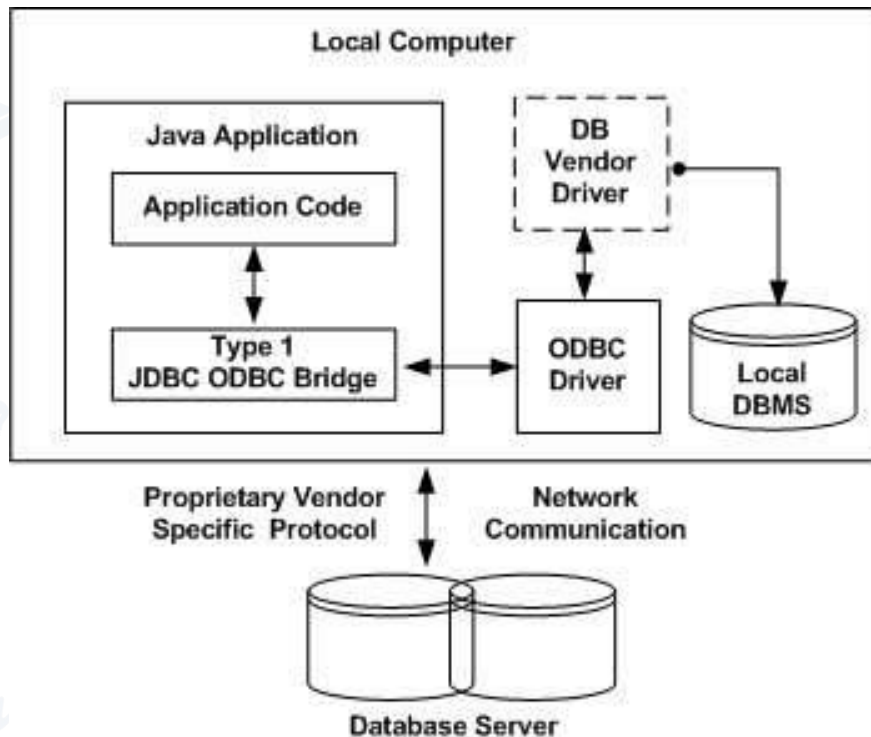
JDBC (Java DataBase Connectivity - з'єднання з базами даних на Java) призначений для взаємодії Java-додатки з різними системами управління базами даних (СУБД). Весь рух в JDBC засновано на драйверах, які вказуються спеціально описаним URL.





## JDBC – ODBC транслятор

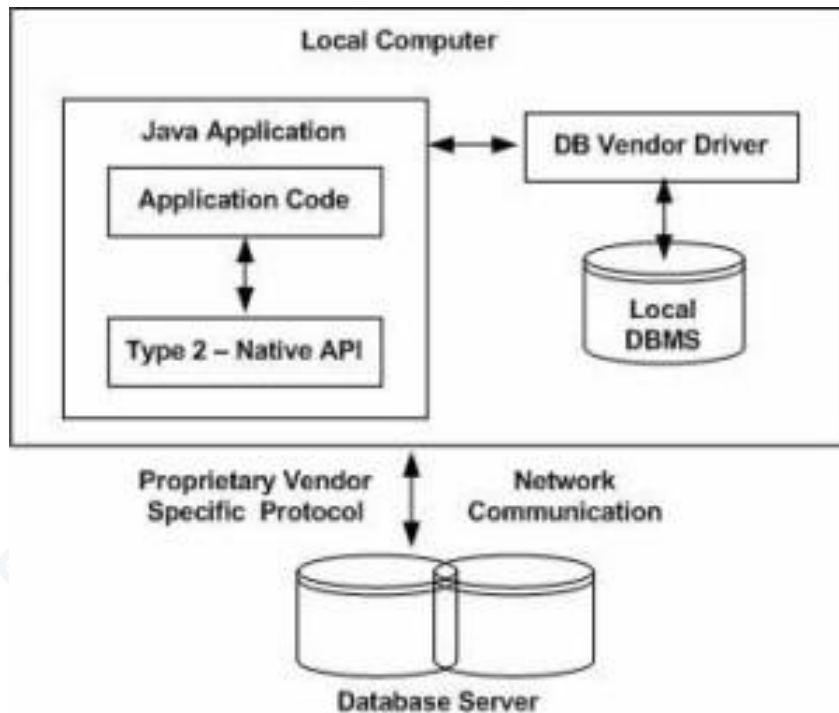
Цей тип драйвера трансліює JDBC у встановлений на кожній машині клієнтську машину ODBC. Використання ODBC вимагає конфігурації DSN, який є цільовою базою даних.





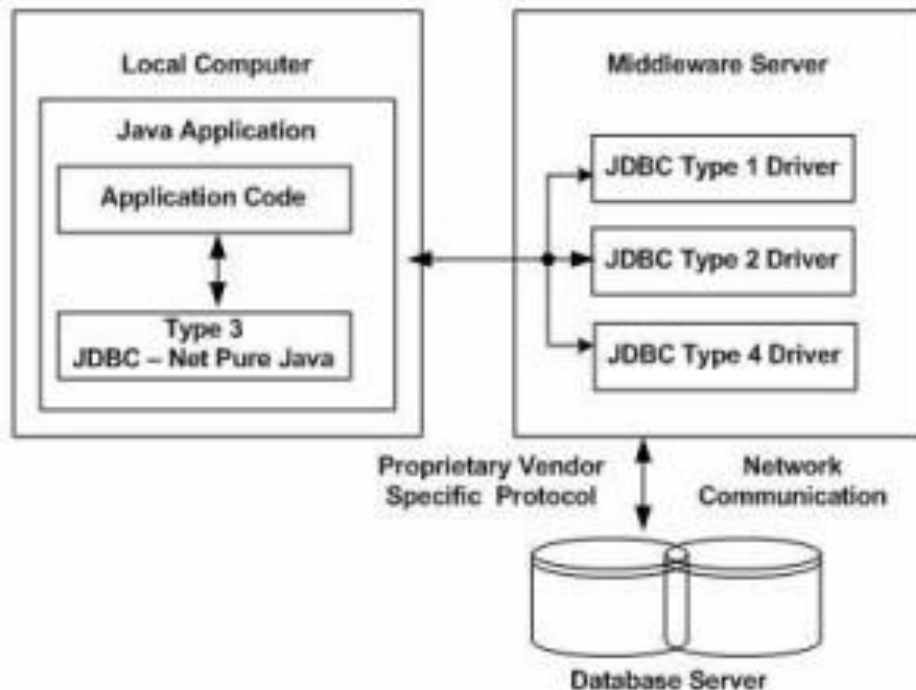
## JDBC – нативний API

У цьому драйвері JDBC API перетворюється на унікальний кожної БД нативний C/C++ API. Його принцип роботи вкрай схожий на драйвер першого типу.



## JDBC драйвер на основі бібліотеки Java

Цей тип драйверів використовує триланковий підхід для отримання доступу до БД. Для зв'язку з проміжним сервером програми використовується стандартний мережевий сокет. Інформація, отримана від цього сокету, транслюється проміжним сервером у формат, який необхідний для конкретної БД і направляється в сервер БД.

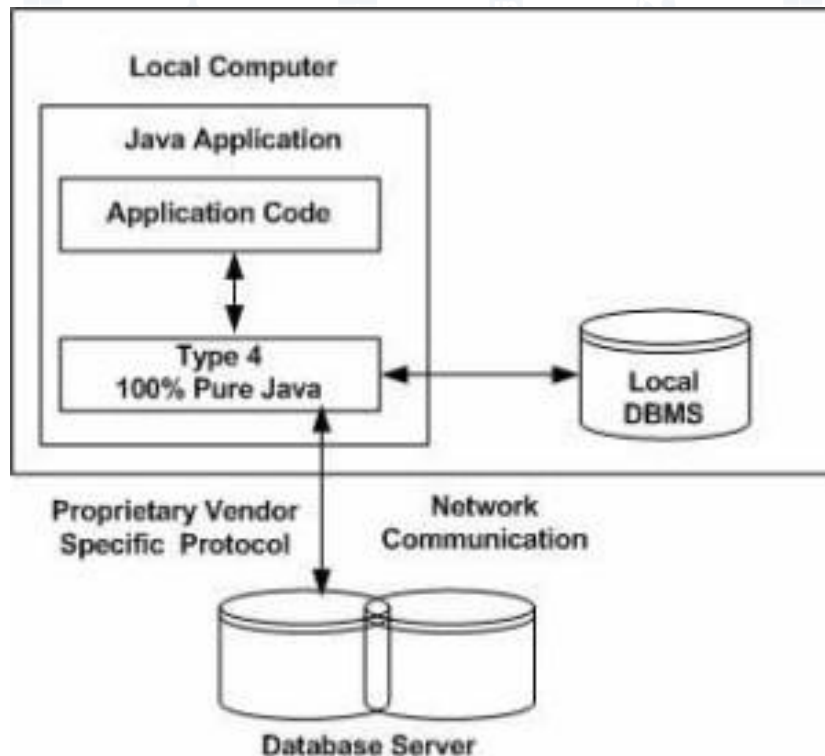







## Чиста Java

Цей тип драйверів розроблений повністю з використанням мови програмування Java та працює з БД через сокетне з'єднання. Головна його перевага - найбільша продуктивність і, як правило, надається розробником БД.

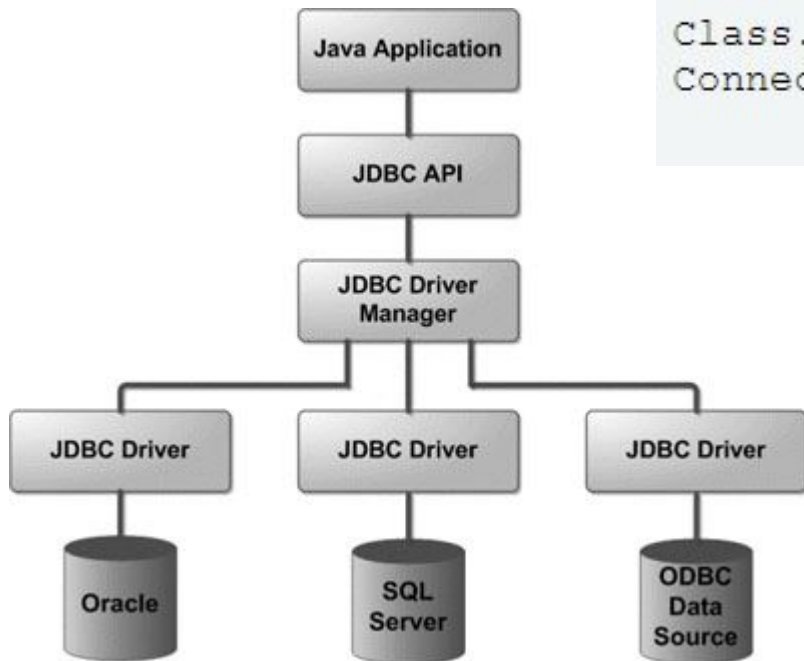




```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <version>8.0.33</version>  
</dependency>
```

```
<dependency>  
  <groupId>org.postgresql</groupId>  
  <artifactId>postgresql</artifactId>  
  <version>42.6.0</version>  
</dependency>
```

```
<dependency>  
  <groupId>com.oracle.database.jdbc</groupId>  
  <artifactId>ojdbc8</artifactId>  
  <version>23.2.0.0</version>  
</dependency>
```

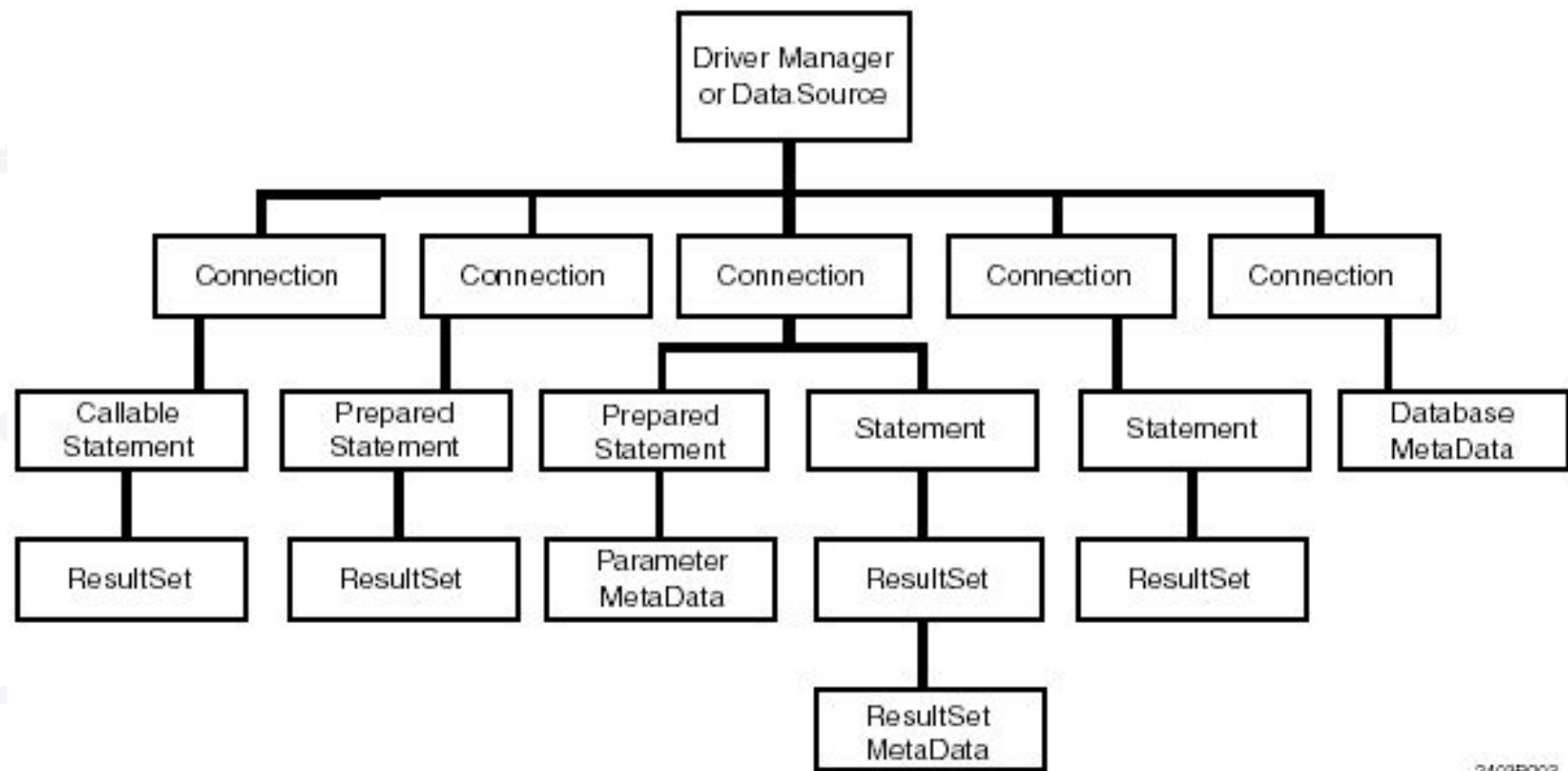


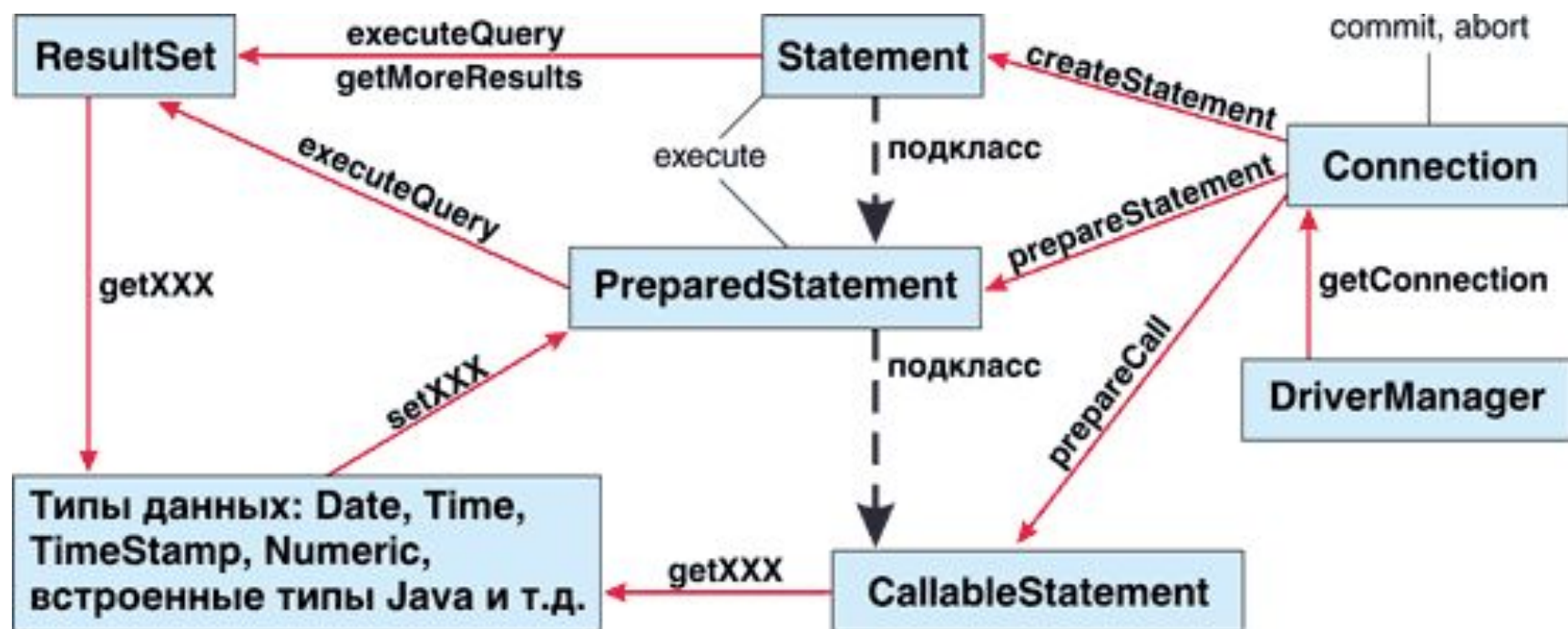
```
Class.forName(driverClass);
Connection connection = DriverManager
    .getConnection(url, user, password) ;
```

Де **driverClass** - це рядок із повним ім'ям класу JDBC драйвера, наприклад `org.h2.Driver` для H2 Database або `com.mysql.jdbc.Driver` для MySQL.

**DriverManager** - це синглтон, який містить інформацію про всі зареєстровані драйвери. Метод `getConnection` на основі параметра URL знаходить `java.sql.Driver` відповідної бази даних і викликає метод `connect`.

```
Class.forName("com.mysql.jdbc.Driver");
Connection connect = DriverManager
    .getConnection( url: "jdbc:mysql://localhost:3306/student?"
        + "user=root&password=root" );
```





```
Statement statement = connect.createStatement();  
statement.executeQuery( sql: "select * from city");
```

```
PreparedStatement preparedStatement = connection.prepareStatement( sql: "insert into city(city) values (?)");
```

```
List<String> cityList = Arrays.asList("London", "Paris", "Madrid", "Berlin");
```

```
cityList.forEach(city -> {  
    try {  
        preparedStatement.setString( parameterIndex: 1, city);  
        preparedStatement.executeUpdate();  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
});
```



## DatabaseMetaData

С помощью Connection можно получить очень полезную сущность DatabaseMetaData. Она позволяет получить метаинформацию о схеме базы данных, а именно какие в базе данных есть объекты - таблицы, колонки, индексы, триггеры, процедуры и так далее.