Lesson 22

17.07.2023

```
public class ex1 {
    public static void main(String[] args) {
        {
            for (;;)
                System.out.println("Java");
        }
    }
}
```

```
public class ex2 {
    public static void main(String[] args) {
        try {
            foo();
        } catch (Exception ex) {
            System.out.println("exMain");
            ex.printStackTrace();
        }
    }

    public static void foo() {
        try {
            throw new IllegalArgumentException("catch");
        } finally {
            try {
                throw new RuntimeException("finally");
            } catch (IllegalArgumentException ex) {
                System.out.println("exFoo");
                ex.printStackTrace();
            }
        }
    }
}
```

```
public class ex3 {
    public static void main(String[] args) {
        int i = 10 ;
        System. out .println(i > 3 != false );
    }
}
```

```
public class ex4 {
    public static void main(String[] args) {
        byte var = 100 ;
        switch (var) {
            case 100 :
                System. out .println( "var is 100" );
                break ;
            case 200 :
                System. out .println( "var is 200" );
                break ;
            default :
                System. out .println( "In default" );
        }
    }
}
```

```
public class ex5 {
    public static void main(String[] args) {
        StringBuilder sb = new StringBuilder("TOMATO");
        System.out.println(sb.reverse()
                .replace('O', 'A'));
    }
}
```

```
ALTER TABLE table_name
ADD column_name datatype;
```

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

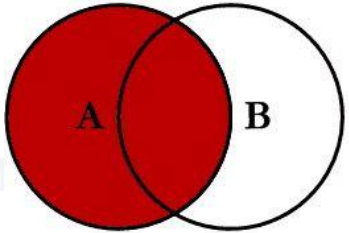
```
SELECT column1, column2, ...
FROM table_name;
```

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

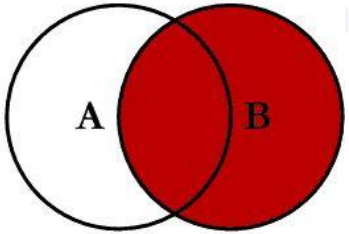
```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

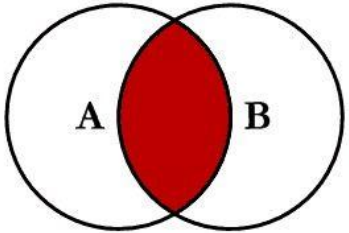
SQL JOINS



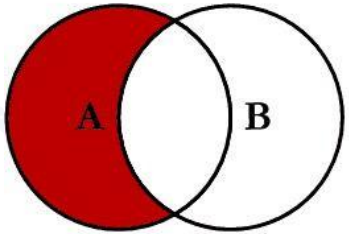

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



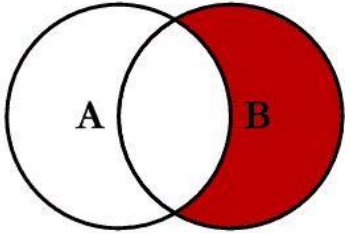

```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



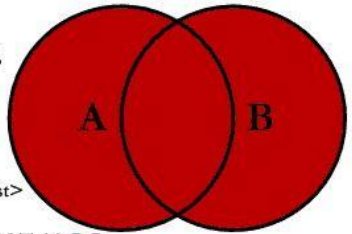

```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



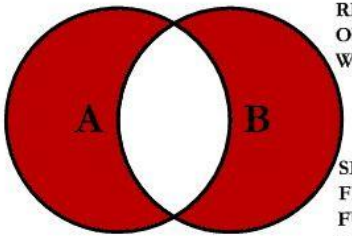

```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```




```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

© C.L. Moffatt, 2008

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

```
SELECT DeptID, AVG(Salary)
FROM Employee
GROUP BY DeptID;
```

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

Aggregate Functions

SUM(): Returns the sum or total of each group.
COUNT(): Returns the number of rows of each group.
AVG(): Returns the average and mean of each group.
MIN(): Returns the minimum value of each group.
MAX(): Returns the minimum value of each group.

Employee

EmployeeID	Ename	DeptID	Salary
1001	John	2	4000
1002	Anna	1	3500
1003	James	1	2500
1004	David	2	5000
1005	Mark	2	3000
1006	Steve	3	4500
1007	Alice	3	3500

SELECT *DeptID, AVG(Salary)*
FROM *Employee*
GROUP BY *DeptID;*

GROUP BY Employee Table using DeptID

DeptID	AVG(Salary)
1	3000.00
2	4000.00
3	4250.00

SELECT *DeptID, AVG(Salary)*
FROM *Employee*
GROUP BY *DeptID*
HAVING *AVG(Salary) > 3000;*

HAVING

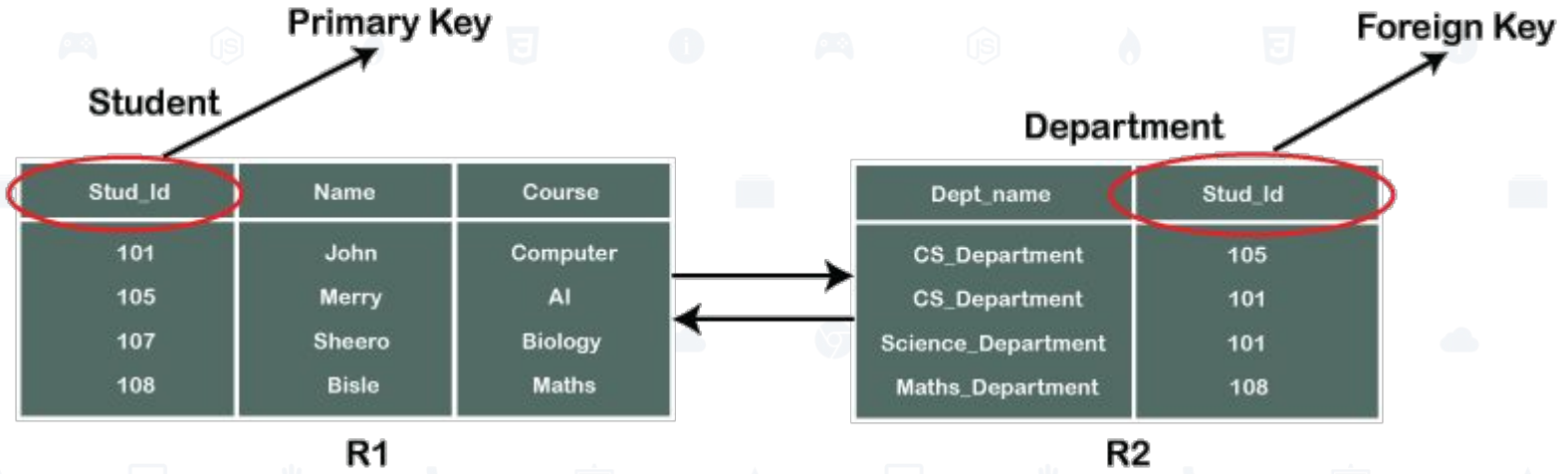
DeptID	AVG(Salary)
2	4000.00
3	4250.00

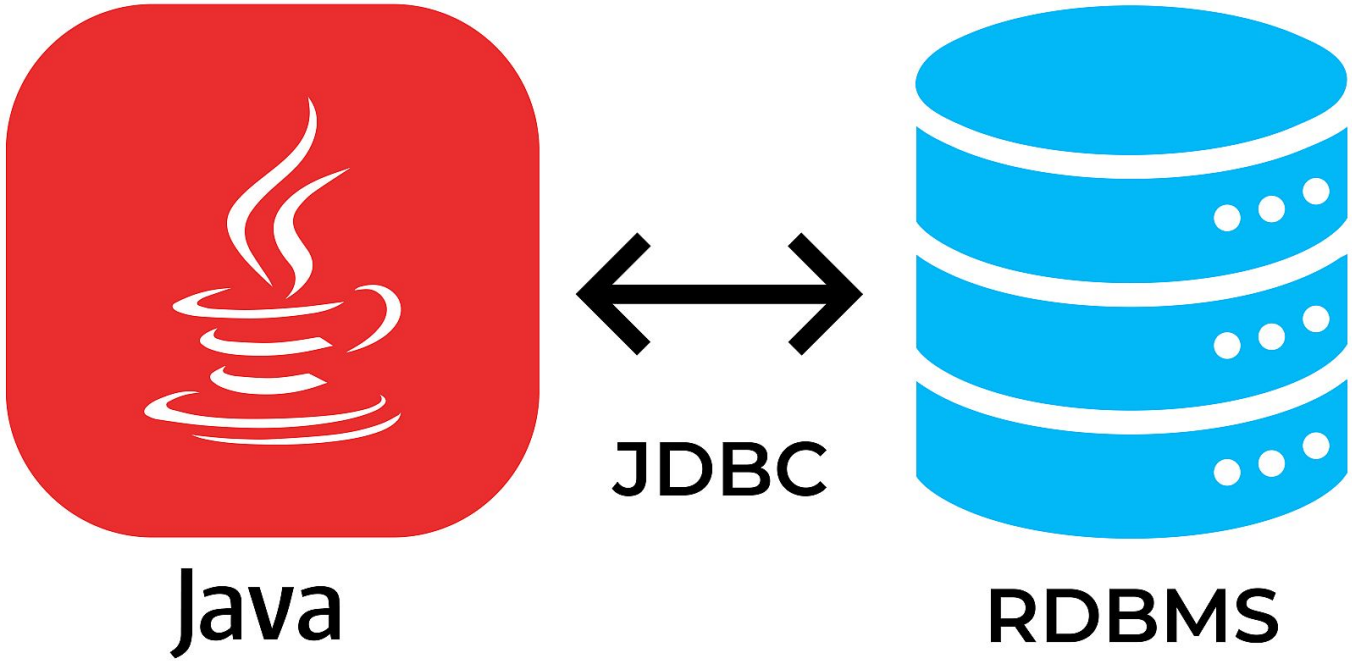
Primary key (PK)

У кожній таблиці БД може бути первинний ключ. Під первинним ключем розуміють поле або набір полів, що однозначно (унікально) ідентифікують запис. Первинний ключ має бути мінімально достатнім: він має складатися з полів, видалення яких із первинного ключа не позначиться з його унікальності.

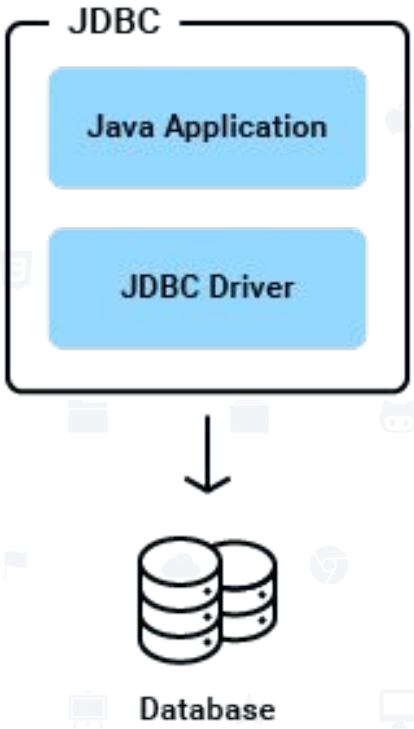
Foreign key(FK)

Забезпечує однозначний логічний зв'язок між таблицями однієї БД.



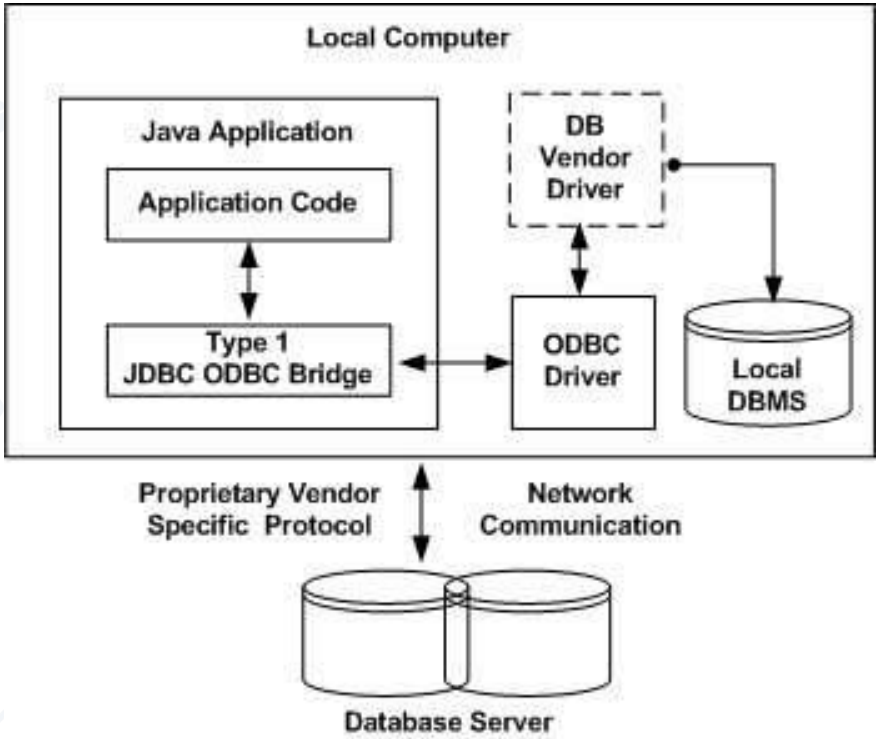

JDBC
Java
RDBMS

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.



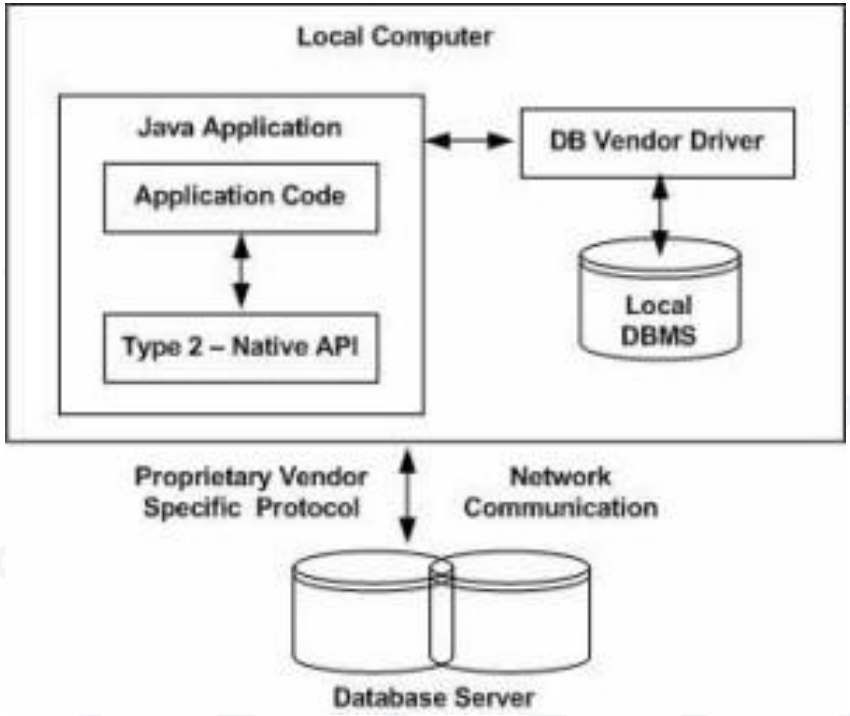
JDBC – ODBC транслятор

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



JDBC – нативний API

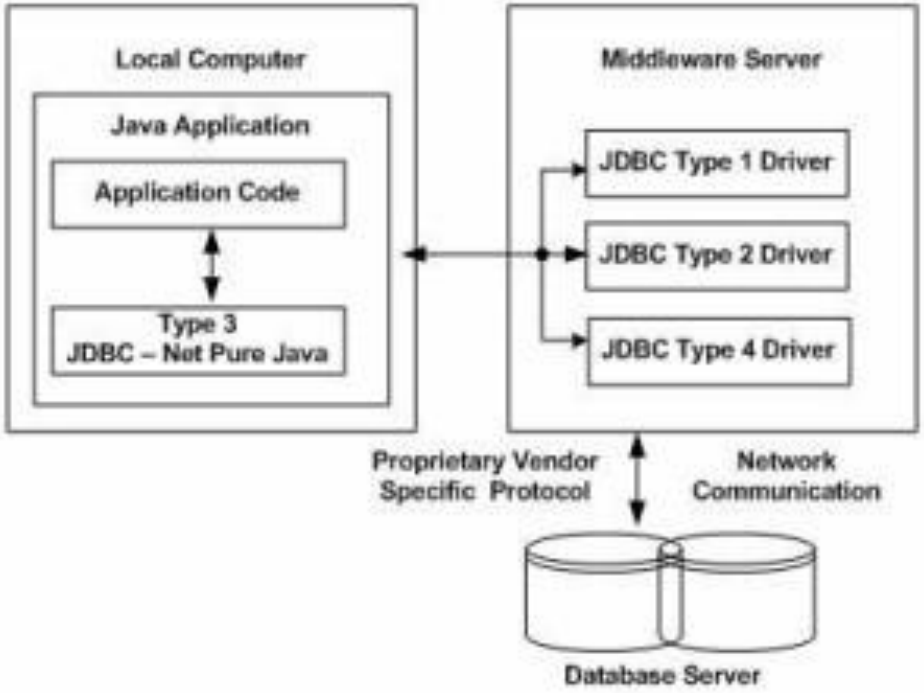
У цьому драйвері JDBC API перетворюється на унікальний кожної БД нативний C/C++ API. Його принцип роботи вкрай схожий на драйвер першого типу.



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.

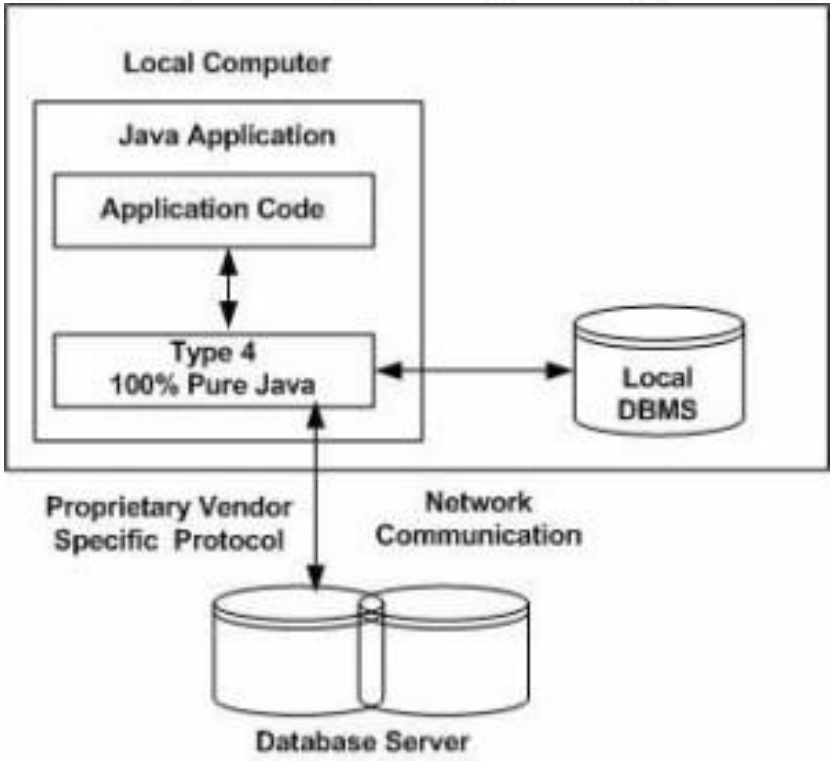
.



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.

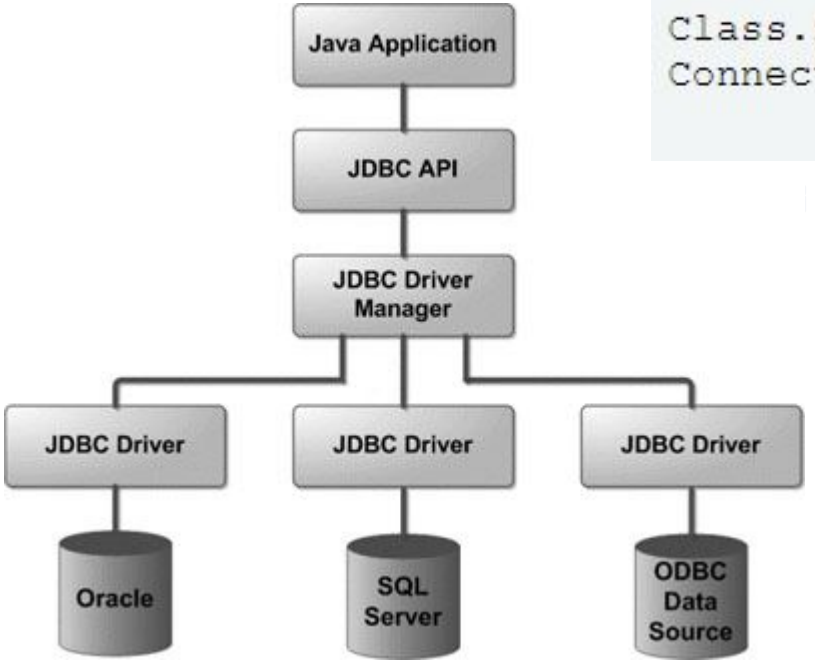
.



```
<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>
```

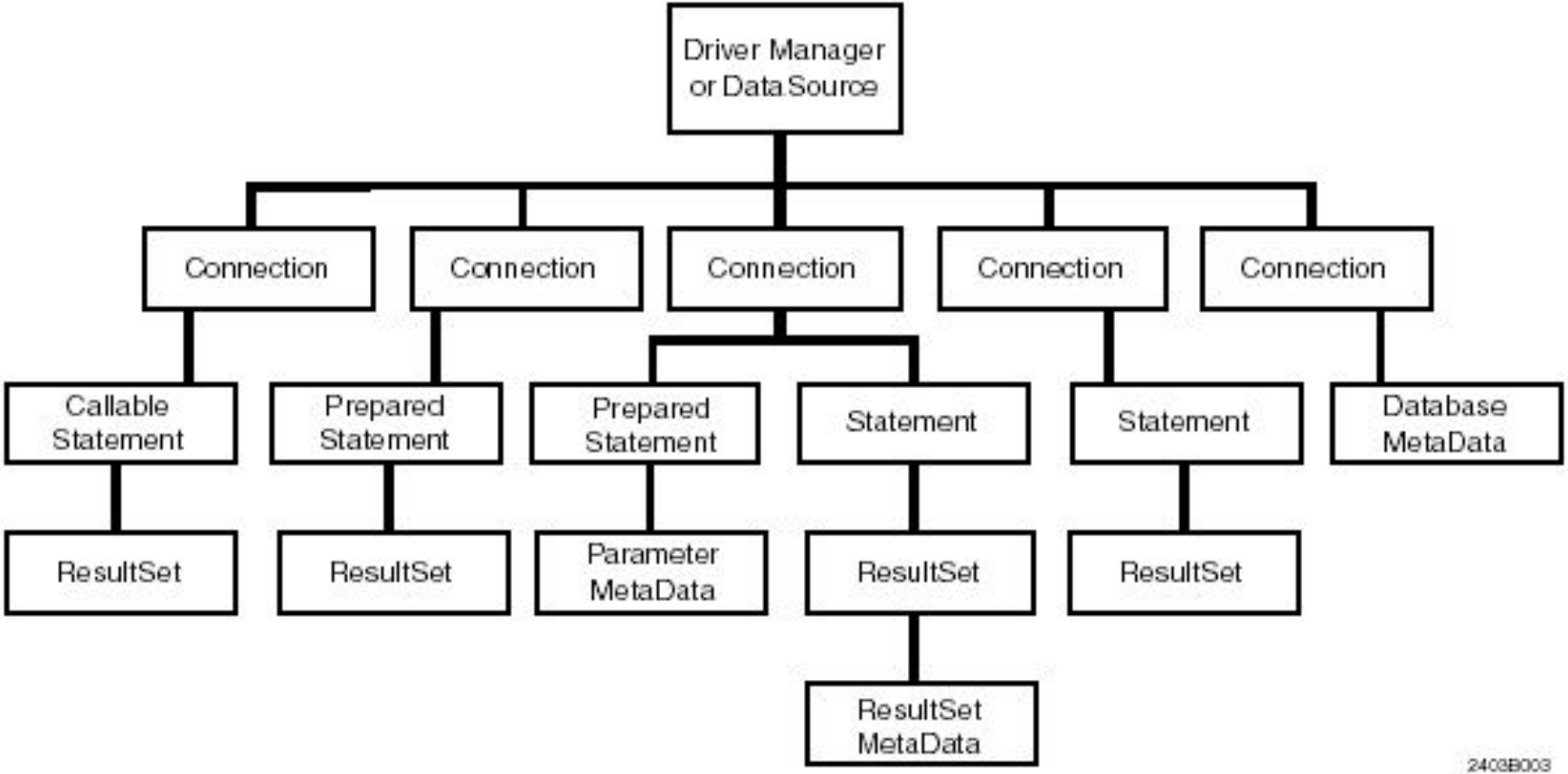


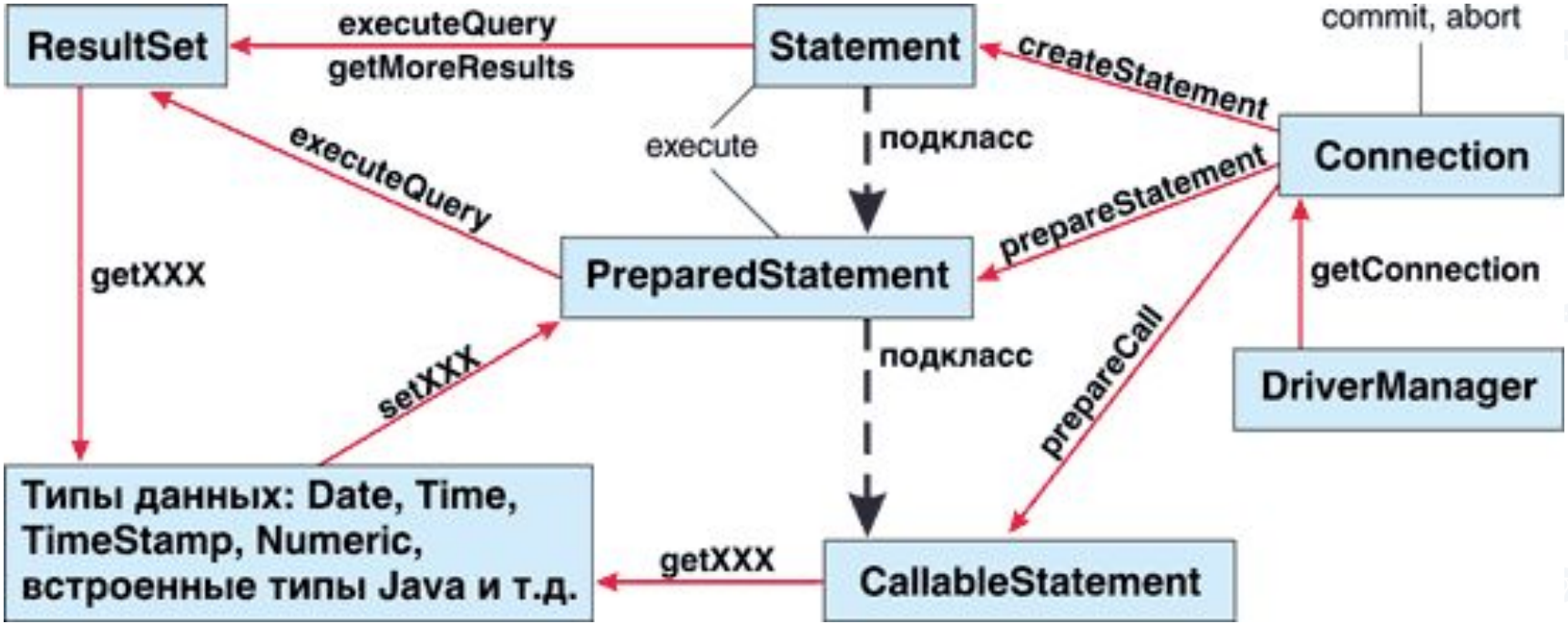

```
Class.forName(driverClass);
Connection connection = DriverManager
        .getConnection(url, user, password) ;
```

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");
```


Driver Manager or Data Source
Connection
Connection
Connection
Connection
Connection
Callable Statement
Prepared Statement
Prepared Statement
Statement
Statement
Database MetaData
ResultSet
ResultSet
Parameter MetaData
ResultSet
ResultSet
ResultSet MetaData
2403B003


ResultSet
executeQuery
getMoreResults
Statement
commit, abort
createStatement
Connection
execute
подкласс
executeQuery
prepareStatement
getXXX
PreparedStatement
getConnection
setXXX
подкласс
prepareCall
DriverManager
Типы данных: Date, Time, TimeStamp, Numeric, встроенные типы Java и т.д.
getXXX
CallableStatement

```
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.