**JDBC**

JDBC – стандарт для взаимодействия Java-приложения с реляционными БД, представляет собой набор классов и интерфейсов для работы с различными реляционными БД. Реализация интерфейсов JDBC API осуществляется за счет JDBC-драйвера.

JDBC-драйвер – jar-файл, который обеспечивает взаимодействие между Java-приложением и конкретной БД. Предоставляет реализацию JDBC API для конкретной реляционной БД (адаптер Java-приложения к БД – преобразует запрос на Java в формат, который БД сможет обработать и выполнить).

Стандартизированный API обеспечивает независимость Java-приложения от БД, с которой она взаимодействует, то есть можно легко переключаться между разными БД, изменив драйвер на соответствующий, при этом не меняя реализацию приложения.

**Общий план подключения и работы:**

1) Подключение драйвера к проекту – добавление его в classpath(file-> proj structure-> modules)

2) Загрузка класса этого драйвера и его регистрация – с помощью Class.forName() или DriverManager.registerDriver() – с JDBC 4 это происходит автоматически при создании подключения.

Class.*forName*("org.postgresql.Driver");  
  
Driver driver = new org.postgresql.Driver();  
DriverManager.*registerDriver*(driver);

3) Создание подключения (получение объекта Connection) с помощью DriverManager.getConnection().

4) Дальнейшая работа с классами, представляющими работу с SQL запросами (Statement, PreparedStatment, CallableStatment).

**Классы**

*DriverManager* – Класс, предназначенный для работы с драйверами и установления соединения между Java приложением и БД. Основное назначение – регистрация драйверов и установление соединения.

registerDriver() – явно можно не вызывать, т.к. метод вызывается в static блоке класса Driver при его загрузке.

getConnection() – опрашивает каждый загруженный драйвер для определения того, с которым можно установить соединение (действует простым перебором и устанавливает соединение с первым соответствием)

*Connection* – устанавливается с помощью DriverManager.getConnection(), посредством передачи в него данных для подключения. Включает в себя методы для работы с транзакциями (commit, rollback, setautocommit, savepoint), работа со специальными типами (blob, clob, array, xml), исполнение SQL запросов (statement, preparedstatment, callablestatment).

**Классы для работы с запросами**

*Statement* – Для работы SQL операторов без параметров, то есть статических SQL выражений, так как подвержены SQL инъекциям, поскольку он выполняет SQL-запрос без дополнительной проверки.

Execute() – Исполняет SQL-запрос, возвращает Boolean, поэтому предпочтительное использование – DDL операции, что не подразумевает возвращаемые данные (возвращает true, если результат запроса resultset и false в остальных случаях). При этом все таки можно получить кол-во измененных записей или ResultSet посредством вызова getUpdateCount(), getResultSet() и др. на соответствующем Statement.

executeUpdate() – отличается возвращаемым значением, возвращает число затронутых операцией записей (-1 если таких нет). Используется, когда не требуется получить множество результатов запроса.

executeQuerry() – возвращает resultset, используется, в случае, если запрос подразумевает возвращение некоторого множества. Используется в запросах типа select или запросах использующих returning.

Batch – для работы с пакетом SQL запросов – addBatch(), clearBatch()

PreparedStatment – наследник statment

CallableStatment – для вызова хранимых процедур.

SQL инъекции – внедрение вредоносного SQL запроса в основной запрос.

Примеры: or 1 = 1 или or ‘’ =’’ или после ; пишешь новый запрос

Для работы с Null результатами в resultset нужно использовать getObject и передать класс, в который кастить Class.class

Для получения ключей сгенерированных автоматически нужно выставить флаг RETURNING\_GENERATED\_KEYS при обращении к statement

Для возврата вставленных данных нужно использовать returning в sql запросе и получить его соответствующим execute и обработать полученный resultset