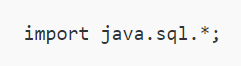
JDBC

1. Как создать Connection?

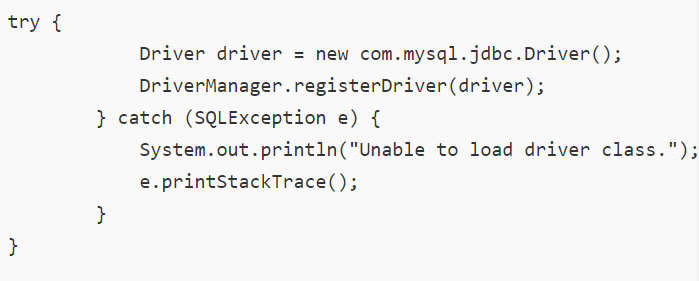
Для того, чтобы установить соединение с БД, нам необходимо выполнить следующие шаги:

* **Импортировать пакет JDBC в наш Java код.**



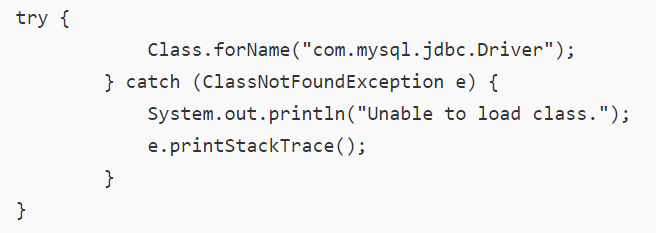
* **Зарегистрировать JDBC драйвер**.

1 способ: **DriverManager.registerDriver() (статический метод).**



2 способ: **Class.forName().** Этот способ является наиболее распространённым. Мы используем метод Class.forName() для динамической загрузки класса драйвера в память, после чего происходит его автоматическая регистрация.

Данный способ является предпочтительным, так как он обеспечивает конфигурируемую и переносимую регистрацию драйверов.



* **Передать информацию для соединения с БД (URL, имя пользователя и пароль).**

После того, как мы зарегистрировали наш драйвер, мы можем установить соединение, используется метод DriverManager.getConnection().

Этот метод имеет три перегруженных варианта:

* getConnection (String url)
* getConnection (String url, Properties props)
* getConnection (String url, String user, String password)

Как мы видим, каждый из этих методов требует URL нашей БД. Т.е. нам необходимо передать информацию о нашей БД.

Для различных БД URL различен.

* **Создать соединение с помощью метода getConnection().**

String URL = "jdbc:mysql://localhost/PROSELYTE\_TUTORIAL";

String USER = "root";

String PASSWORD = "password";

Connection = DriverManger.getConnection(URL, USER, PASSWORD);

1. Как правильно закрыть Connection?

После того, как закончим работу с нашей программой нам необходимо закрыть все соединения и освободить ресурсы. Если мы не сделаем этого вручную, это будет сделано сборщиком мусора при уничтожении объектов, которые не используются.

Но полагаться на уборщик мусора – это крайне плохая идея и пример дурного тона.

В коде это выглядит так:



1. Какие есть типы драйверов для соединения с СУБД?

JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL. Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, за который драйвер отвечает.

Различают четыре типа драйверов.Java программа работает с БД в двух частях. Первая часть это JDBC API, а вторая – драйвер, который и выполняет всю работу. Каждый тип определяет реализацию JDBC драйвера по возрастающей степени независимости от платформы, производительности и легкости администрирования. Эти четыре типа следующие:

**Тип 1: JDBC-ODBC бридж** (**JDBC-ODBC Bridge plus ODBC Driver)**– транслирует JDBC в ODBC и для взаимодействия с базой данных использует драйвер ODBC. Компания Sun включила в состав JDK один такой драйвер — мост JDBC/ODBC. Сейчас имеются более удачные реализации.

**Тип 2: Нативный API/частично Java драйвер (Native API partly Java technology-enabled driver)** – переводит вызовы JDBC в вызовы специфичные к СУБД таких как например SQL Server, Informix, Oracle или Sybase. Драйвер 2-го типа общается напрямую с сервером базы данных, следовательно он требует, чтобы какой то бинарный код был на стороне клиентской машины.

**Тип 3: Сетевой протокол/«чистый» Java драйвер (Pure Java Driver for Database Middleware)** – использует трехуровневую архитектуру, где вызовы JDBC посылаются на промежуточный т.н. сервер приложений, далее этот сервер транслирует вызовы (явно или косвенно) в вызовы специфичного к СУБД нативного интерфейса для дальнейшего обращения к базе данных. Если сервер среднего слоя написан на Java то он может использовать для трансляции JDBC драйверы 1 и 2 типов.

**Тип 4: Нативный протокол/«чистый» Java драйвер (Direct-to-Database Pure Java Driver)** – конвертирует вызовы JDBC в специфический протокол вендора СУБД, так что клиентские приложения могут напрямую обращаться с сервером базы данных. Драйверы 4-го типа полностью реализуются на Java с целью достижения платформенной независимости и устранения проблем администрирования и развертывания.



1. Чем отличается Statement от PreparedStatement? Где сохраняется запрос после первого вызова PreparedStatement? Будет ли тот же самый эффект как и от PreparedStatement, если формировать запрос просто в строке и отправлять его в Statement?

Большинство реляционных баз данных обрабатывают запрос JDBC / SQL в четыре этапа:

1. Синтаксический анализ входящего SQL-запроса
2. Компиляция SQL-запроса
3. Планирование/оптимизация пути сбора данных
4. Выполнение оптимизированного запроса / получение и возврат данных

Statement всегда будет выполнять описанные выше четыре шага для каждого SQL-запроса, отправленного в базу данных. PreparedStatement предварительно выполняет шаги (1) - (3) в процессе предыдущего выполнения. Таким образом, при создании PreparedStatement сразу же выполняется некоторая предварительная оптимизация. Результатом является уменьшение нагрузки на ядро СУБД во время выполнения.

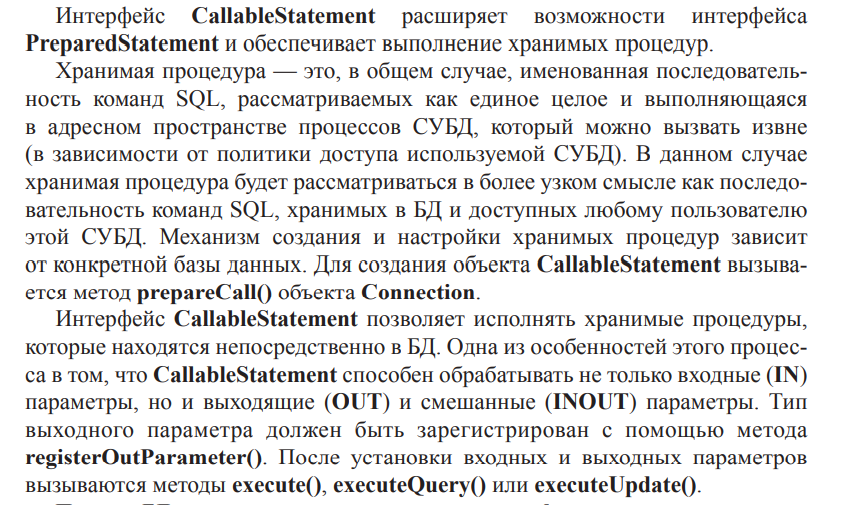
PreparedStatement **protects against SQL injection.**

**После первого вызова** PreparedStatement запрос сохраняется в объекте, что и ускоряет обмен информацией с БД при многократном выполнении однотипных запросов.

Такого же эффекта не будет, если формировать запрос просто в строке и отправлять его в Statement, по причине разной работы Statement и PreparedStatement.

1. Зачем нужен CallableStatement?

Этот интерфейс становится полезным в случае, когда мы хотим получить доступ к различным процедурам БД. Он также может принимать параметры во время работы программы.



1. Отличие executeUpdate от executeQuery

Существует несколько способов выполнять SQL-запросы в зависимости от типа этого запроса. Для этого у интерфейса Statement существует три различных метода: executeQuery(), executeUpdate(), а так же execute(). Рассмотрим их отдельно.

Самый базовый метод executeQuery() необходим для запросов, результатом которых является один единственный набор значений, таких как у запросов SELECT. Возвращает ResultSet, который не может быть null даже если у результата запроса не было найдено значений.  
Метод execute() используется, когда операторы SQL возвращают более одного набора данных, более одного счетчика обновлений или и то, и другое. Метод возвращает true, если результатом является ResultSet, как у запроса SELECT. Вернет false, если ResultSet отсутствует, например при запросах вида Insert, Update. С помощью методов getResultSet() мы можем получить ResultSet, а getUpdateCount() – количество обновленных записей.

Метод executeUpdate() используется для выполнения операторов INSERT, UPDATE или DELETE, а также для операторов DDL (Data Definition Language – язык определения данных), например, CREATE TABLE и DROP TABLE. Результатом оператора INSERT, UPDATE, или DELETE является модификация одной или более колонок в нуле или более строках таблицы. Метод executeUpdate() возвращает целое число, показывающее, сколько строк было модифицировано. Для выражений типа CREATE TABLE и DROP TABLE, которые не оперируют над строками, возвращаемое методом executeUpdate() значение всегда равно нулю.

Все методы выполнения SQL-запросов закрывают предыдущий набор результатов (result set) у данного объекта Statement. Это означает, что перед тем как выполнять следующий запрос над тем же объектом Statement, надо завершить обработку результатов предыдущего (ResultSet).

1. Как в объекте ResultSet вернуться в предыдущую строку? Всегда ли можно вернуться в предыдущую строку?

Базы данных всё таки о данных, а не о запросах. В [JDBC](https://easyjava.ru/tag/jdbc/) данные, которые возвращают запросы, представлены в виде объектов ResultSet.

По умолчанию курсор ResultSet находится перед первой строкой набора данных.

Существует целых *восемь* методов, перемещающих курсор по ResultSet:

* next() — перемещает курсор на одну строку вперёд. Возвращает true, если перемещение удалось и false, если курсор уже находится за последней строкой.
* previous() — очевидно, антоним next(). Перемещает курсорс на одну строку назад и тоже возвращает true, если перемещение удалось и false, если курсор находится перед первой строкой.
* first() и last() — перемещают курсор соответственно на первую и последнюю строку набора данных. В случае, если набор данных пуст, возвращают false. В случае успешного перемещения возвращают true.
* beforeFirst() и afterLast() — перемещают курсор на позицию *перед* первой строкой или *после* последней строки.
* relative() — перемещает курсор на указанное число строк от *текущей* позиции.
* absolute() — перемещает курсор на указанное число строк от *первой* позиции.

Стоит отметить, что не все эти методы всегда работают. ResultSet (а точнее конкретная его реализация драйвером JDBC) может не поддерживать перемещение кроме как вперёд. Такой ResultSet называется **TYPE\_FORWARD\_ONLY**. В случае, если перемещение возможно, открытый ResultSet может следить за изменениями в базе данных, произошедшими после его открытия или не следить. В первом случае это будет **TYPE\_SCROLL\_SENSITIVE** ResultSet, во втором **TYPE\_SCROLL\_INSENSITIVE**.

1. Последовательность действий необходимых для выполнения запроса к БД.

* Получаем connection
* Создаем statement
* execute()

1. Как получить сгенерированный СУБД первичный ключ без выполнения дополнительного запроса к БД?

Если в таблице используется автоматическая генерация ключей, то для их получения используется метод Statement getGeneratedKeys(), который вернет сгенерированный ключ.