База данных — это совокупность структурно организованных данных, относящихся к некоторой предметной области, при этом вместе с данными хранится информация о характеристиках этих данных и об их взаимных связях. Система управления базами данных — это приложение, которое предназначено для работы с базами данных. Для работы с реляционными данными создан язык SQL (Structured Query Language). Это декларативный язык, он не говорит, как можно получить результат. Он указывает, какой нужен результат, а дальше СУБД сама решает, как его получить оптимальным способом.

На значения в таблицах могут накладываться ограничения. Это могут быть ограничения по типу данных — каждый столбец имеет свой тип, и значения в этом столбце должны соответствовать типу. Может ограничиваться диапазон допустимых значений. Также можно разрешить только уникальные, либо только не пустые значения в столбце. Есть специальные виды ограничений. Это, например, первичный ключ — один или множество столбцов, значения которого однозначно идентифицируют строку таблицы. Значения первичного ключа должны быть уникальны и не могут иметь значение NULL. Внешний ключ задается, если значения столбца в таблице могут принимать только значения, которые содержатся в некотором столбце другой таблицы.

JOIN

Таблицы можно соединять с помощью оператора JOIN. Есть несколько типов соединений, самым распространенным их них является внутреннее соединение — INNER JOIN. При выполнении соединения двух таблиц в результирующий таблице набор столбцов представляет из себя объединение столбцов первой и второй таблиц. Значения формируются следующим образом: каждой строке первой таблицы ставится в соответствие каждая строка второй таблицы, и проверяется условие соединения (ON). Если оно истинно для данной строки, то данная строка попадает в итоговую таблицу. Если соединение производится по условию равенства значений столбцов (это наиболее частый вариант), и сравниваемые столбцы имеют одинаковое название, то можно использовать сокращенный синтаксис с использованием USING с именем столбца вместо ON с условием.

SELECT

Для получения данных из таблиц используется SQL команда выборки SELECT. С ее помощью можно выбрать все значения из таблицы, ограничить выборку только нужными столбцами, ограничить выборку условием, которому должны удовлетворять выводимые строки. Можно произвести выборку из соединения таблиц, также можно отсортировать выводимые данные по некоторому столбцу, выбрать только неповторяющиеся данные, а также применить агрегирующие функции, например, посчитать количество строк в таблице или сумму значений.

```
SELECT * FROM students;
SELECT name, group FROM students;
SELECT * FROM students where group = 'P3110';
SELECT name, faculty FROM students INNER JOIN groups
USING(group) where faculty = 'СУиР';
SELECT * FROM students ORDER BY name;
SELECT DISTINCT type FROM courses;
SELECT COUNT(*) FROM students;
```

DDL

Создавать, изменять и удалять таблицы можно с помощью подмножества языка SQL — языка определения данных (DDL). К таким командам относятся — CREATE, предназначенная для создания таблиц, и DROP, предназначенная для удаления таблиц. При создании таблицы необходимо задать имена и типы данных всех столбцов, также при создании задаются ограничения.

```
DDL — Data definition language
CREATE TABLE persons (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    birthday DATE
);
DROP TABLE persons;
```

DML

Для работы с данными, когда таблицы уже созданы предназначен DML — язык манипулирования данными. К этим командам относятся:

- INSERT — позволяет добавить в таблицы данные
- UPDATE — позволяет изменить значения данных в строках, удовлетворяющих некоторому условию.
- DELETE — позволяет удалить строки по заданному условию.

```
DML — Data manipulation language
INSERT INTO students VALUES (300000, 'Джо', 'P3110');
UPDATE courses SET  semester = 2 WHERE course_id = 1;
DELETE FROM students WHERE student_id > 299999;
```

Основы JDBC

При взаимодействии с базами данных, желательно обращаться к различным базам унифицированным способом. Есть 2 варианта решения проблемы:

1. Для каждой базы пишется своя собственная библиотека, набор функций или методов, позволяющих любому приложению пользоваться основными функциями базы данных. Основной недостаток такого способа — необходимость переписывать приложения при изменении базы данных на другую (а иногда и при обновлении базы данных до новой версии).

2. Разработать единый интерфейс взаимодействия, а для каждой конкретной базы написать свой драйвер.

В настоящий момент используются 2 основных реализации варианта единого интерфейса работы с базами данных: ODBC и JDBC. При этом приложению предоставляется единый интерфейс взаимодействия, работа осуществляется с помощью менеджера драйверов, который подключает драйвер для определенной СУБД.

JDBC - Java Database Connectivity, описывает 2 основных интерфейса:

JDBC API — это интерфейс для доступа к данным со стороны пользователя базы, его обычно используют разработчики приложений, работающих с базой данных.

JDBC Driver API — это интерфейс драйверов, его используют разработчики самих драйверов, обычно это производители конкретных СУБД.

Все основные классы JDBC API находятся в 2 пакетах:

java.sql - базовая библиотека, которая содержит основной набор интерфейсов и классов, позволяющих выполнять все основные функции взаимодействия с базами данных

javax.sql - расширенная библиотека, содержащая классы и интерфейсы, позволяющие обеспечить дополнительные возможности работы с базами данных.

Процесс взаимодействия

Вызываем метод getConnection у класса DriverManager, которому передаем параметры соединения с базой данных. Этот метод вернет объект типа Connection, представляющий собой абстракцию соединения. У соединения вызывается метод createStatement, который создает запрос - объект типа Statement. Исполнение запроса производится с помощью метода executeQuery. Он возвращает результат - объект типа ResultSet. Результат представляет из себя таблицу и реализует шаблон Iterator, то есть позволяет перебирать строки результата с помощью метода next(). После окончания работы необходимо закрыть все объекты: ResultSet, Statement и Connection. Для этого можно применить блок try с ресурсами.

```
Connection conn =
   DriverManager.getConnection( … );

Statement stat = conn.createStatement();

ResultSet res = stat.executeQuery("SELECT … ");

while (res.next()) { … }

res.close();
stat.close();
conn.close();
```

Класс DriverManager

Для организации работы с базой данных используется класс DriverManager, который содержит список драйверов. В прошлых версиях JDBC драйверы надо было загружать явно, сейчас это делается автоматически. Загрузить драйвер явно можно, вызвав метод Class.forName() и передав ему имя класса драйвера. Также можно перечислить нужные классы драйверов в системном свойстве jdbc.drivers. Для автоматической загрузки драйверов применяется механизм сервис-провайдеров, для этого в jar-архиве с драйверами в файле java.sql.Driver указывается список классов, реализующих интерфейс Driver, которые должны быть загружены для работы с базой данных.

Управляет списком драйверов

Загрузка драйвера

- Class.forName()
- jdbc.drivers=

Неявная загрузка с помощью ServiceLoader

- META-INF/services/java.sql.Driver

DriverManager содержит метод getConnection(). Ему передаются параметры соединения, один из вариантов - передать URL, который имеет вид jdbc:protocol:name, компоненты разделяются точкой с запятой. Протокол обычно зависит от типа базы данных, а поле name содержит название конкретной базы данных. Есть еще второй формат метода, где указывается объект Properties, в котором задаются имя пользователя и пароль для установления соединения. Значения можно прочитать из файла. Также можно указать имя пользователя и пароль прямо в аргументах метода, но так делать не рекомендуется. Метод getConnection() возвращает объект типа Connection.

Метод getConnection()

- Возвращает Connection
- getConnection(String url)
  - URL = jdbc:*protocol*:*name*
- getConnection(String url, Properties info)
  - Properties info = new Properties();
  - info.load(new FileInputStream("db.cfg");

  файл db.cfg

```
user = s999999
password = sss999
```

- getConnection(Stiring url, String username, String passwd)

Интерфейс Connection

Connection — это абстракция соединения с базой данных, или сессии. Интерфейс Connection имеет методы для получения объекта запроса - createStatement, prepareStatement и prepareCall. Кроме этого, Connection можно использовать для получения метаданных о базе данных, с которой установлено соединение.

Абстракция соединения (сессия)

- методы:
  - Statement createStatement()
  - PreparedStatement prepareStatement(String sql)
  - CallableStatement prepareCall(String sql)
  - DatabaseMetaData getMetaData()

Statement

Рассмотрим подробнее интерфейс Statement и его потомков. Объект интерфейса Statement сам запрос не содержит. Запрос передается в качестве параметра при вызове метода executeQuery(). При вызове данного метода нужно быть внимательным при формировании запроса, если он составляется из данных, предоставленных пользователем. Никогда нельзя доверять полученным от пользователя данным, нужно контролировать их тип и содержимое, чтобы не допустить применение SQL-инъекций.

Statement

- Статический SQL-запрос
- Statement st = connection.createStatement();

  st.executeQuery("SELECT * FROM table WHERE id = 15");

PreparedStatement

Объект интерфейса PreparedStatement представляет подготовленный запрос с параметрами. Он применяется для выполнения идентичных запросов, отличающиеся одним или несколькими параметрами, например, ID. В таких случаях запрос подготавливается заранее, а на месте параметров указываются знаки вопроса. Затем можно задать значения параметров с помощью одного из методов setInt, setString, или подобных (выбор метода зависит от типа параметра). Запрос выполняется при вызове метода executeQuery(). Далее можно установить новые значения параметров и опять выполнить запрос. Этот способ имеет несколько преимуществ. Во-первых, запросы исполняются быстрее, так как при подготовке запроса СУБД формирует оптимальный план исполнения запроса, остается только подставить параметры и выполнить его. Во-вторых, такие запросы защищены от SQL-инъекций, так как все специальные символы экранируются автоматически.

PreparedStatement (extends Statement)

- Динамический запрос с параметрами
- ps = prepareStatement("SELECT * FROM table WHERE id = ?");
- ps.setInt(1, 15); // 1 — номер параметра, 15 — значение
- SELECT * FROM table WHERE id = 15

CallableStatement

Интерфейс CallableStatement расширяет PreparedStatement, и еще умеет вызывать хранимые процедуры, то есть процедуры, которые хранятся в базе данных. В языке SQL они вызываются с помощью команды CALL. Для этого в объекте типа CallableStatement, задаются входные параметры, и регистрируются выходные параметры. При регистрации выходного параметра его номеру ставится в соответствие тип параметра. После выполнения запроса можно получить значения выходных параметров.

CallableStatement (extends PreparedStatement)

- Вызов хранимой процедуры
- SQL: CREATE PROCEDURE
- cs = prepareCall("CALL getResult (?)");
- cs.setInt(1, 15);
- cs.registerOutParameter(1, Types.INTEGER);
- ...
- int result = cs.getInt(1);
  CALL getResult(15);

ResultSet

Для объектов Statement запрос указывается в параметрах методов исполнения запроса. Для объектов PreparedStatement и CallableStatement параметр не нужен, так как запрос уже задан. Метод execeuteQuery выполняет запрос (обычно это запрос типа SELECT) и возвращает в качестве результата ResultSet, из которого потом можно будет получить данные.

Запросы типа INSERT, UPDATE или DELETE, выполняются методом executeUpdate(), который возвращает количество изменившихся в результате запроса строк. Кроме команд DML, можно также выполнять команды DDL, при этом метод executeUpdate будет возвращать 0, так как команды DDL не работают со строками.

Универсальный метод execute может использоваться для любого типа запросов. Он возвращает результат типа boolean. Значение true обозначает, что после выполнения запроса есть результат в виде таблицы, то есть ResultSet, который можно получить с помощью метода getResultSet(). А значение false обозначает, что в результате выполнения запроса имеются модифицированные строки, количество которых можно узнать с помощью метода getUpdateCount().

ResultSet executeQuery(String sql)

- для исполнения команды SELECT
- Возвращает ResultSet

int executeUpdate(String sql)

- для выполнения запросов INSERT, UPDATE, DELETE
- возвращает количество измененных строк
- Для команд DDL возвращает 0

boolean execute(String sql)

- для выполнения любых запросов
- true, если результат — ResultSet : ResultSet getResultSet()
- false, если результат — updateCount : int getUpdateCount()

- Получение данных из ResultSet

```
while (rs.next()) {
    String name = rs.getString(1);
    int id = rs.getInt("id");
}
```

Переходим к рассмотрению интерфейса ResultSet, объект которого возвращают методы executeQuery и getResultSet. Получить результат из ResultSet можно как из обычного итератора, в цикле while вызывается метод next для установки курсора на очередную строку. С помощью методов getString(), getInt() и т. д. можно получать значения столбцов для текущей строки.

При создании запроса можно указать, какие характеристики будет иметь выдаваемый запросом ResultSet. По умолчанию заданы FORWARD_ONLY и CONCUR_READ_ONLY, при этом ResultSet позволяет перемещать курсор только вперед и работает только на чтение. Если задать любой из типов SCROLL, то перемещать курсор можно будет в обоих направлениях. При этом в INSENSITIVE не видны изменения данных, произошедшие после получения результата, а в SENSITIVE – видны. Характеристика CONCUR_UPDATABLE позволяет вносить в ResultSet изменения, передающиеся в базу данных. Еще 2 характеристики управляют состоянием курсора после команды COMMIT.

- ResultSet
  - Connection.createStatement(sql, type, concurrency, holdability)
  - ResultSetType
    - **TYPE_FORWARD_ONLY**
    - TYPE_SCROLL_INSENSITIVE
    - TYPE_SCROLL_SENSITIVE
  - ResultSetConcurrency,
    - **CONCUR_READ_ONLY**
    - CONCUR_UPDATABLE
  - ResultSetHoldability
    - HOLD_CURSORS_OVER_COMMIT
    - CLOSE_CURSORS_AT_COMMIT

Другие методы нужны для получения данных — это методы get с разными типами данных SQL. Все они принимают в качестве параметра либо порядковый номер элемента в строке, либо имя столбца. Второй способ предпочтительнее, так как при изменении запроса не придется менять код для получения результата.

Получение данных
- `getString(int)`
- `getString(String)`
- `getInt(int)`
- `getInt(String)`
- `getBoolean`
- `getLong`
- `getDouble`
- `GetArray (SQL Array)`
- `getDate`
- `getTimestamp`
- `getReader`

Методы обновления данных позволяют изменить данные сначала вызываются методы updateInt, updateString и т. д., указывающие, какие столбцы в строке должны быть обновлены. Этим методам передается либо порядковый номер, либо имя столбца, и новое значение. Затем вызывается метод updateRow для обновления данных в базе.

Для добавления новой строки нужно переместить курсор на специальную строку методом moveToInsertRow, затем установить нужные значения и потом обновить таблицу базы данных с помощью метода insertRow.

С помощью метода getMetaData у объекта ResultSet можно получить метаданные результата, например, количество, имена и типы данных столбцов, имя таблицы, из которой получен столбец, и т. д. С помощью метода getMetaData у объекта Connection можно получить метаданные о самой базе данных, например, какие каталоги есть в данной базе, какие схемы в ней определены, какие таблицы хранятся в базе и т. д.

ResultSetMetaData ResultSet.getMetaData()

- `getTableName()`
- `getColumnCount()`
- `getColumnName(int n)`
- `getColumnType(int n)`

DatabaseMetaData Connection.getMetaData()

- `getCatalogs()`
- `getTables()`
- `getSchemas()`

Обновление строк

- `updateInt(String, int)`
- `updateInt(int, int)`
- `updateString(String, String)`
- `updateString(int, String)`
- `updateRow()`

Добавление строк

- `moveToInsertRow()`
- `updateInt(String, int)`
- `insertRow()`

Транзакции

Connection

- setAutoCommit(true/false)
- commit()
- rollback()
- setSavepoint()

Statement

- addBatch(String sql)
- clearBatch()
- executeBatch()

Некоторые запросы должны выполняться вместе, то есть за одну транзакцию. Например, мы управляем средствами на банковских счетах. Допустим, что выполняется перевод с одного счета на другой. При этом важно обеспечить атомарность перевода. Операция снятия денег с одного счета и операция зачисления этих денег на другой счет зависимы, и должны либо вместе выполниться, либо вместе не выполниться. Если транзакция началась, то либо она успешно завершится целиком, либо она будет отменена с возвратом базы данных в состояние до начала транзакции. Транзакции в базе обычно реализуются с помощью механизма фиксаций (commit) и откатов (rollback). При старте транзакция, реальное состояние данных в базе не изменяется до момента выполнения команды COMMIT, фиксирующей состояние базы данных. До выполнения команды COMMIT есть возможность отменить транзакцию командой ROLLBACK.

При обычной работе с JDBC состояние базы фиксируется после каждого запроса (Autocommit). Для перевода в режим транзакций, нужно вызвать метод setAutoCommit с параметром false. Метод commit() фиксирует состояние базы данных, метод rollback() позволяет откатить все изменения, сделанные после последней фиксации. Метод setSavePoint() позволяет создать точку сохранения. Запросы можно выполнять пакетами. Для этого можно вызвать метод addBatch() который добавляет запрос в пакет. После формирования пакета можно выполнить все запросы за один раз методом executeBatch(). Метод clearBatch() очищает пакет.

RowSet

В расширенный пакет входит интерфейс RowSet. Он расширяет ResultSet. Кроме работы с результатом запроса RowSet может осуществить соединение с базой данных и выполнить сам запрос. В этом интерфейсе есть методы setURL, setUsername и setPassword для задания параметров соединения. Метод setCommand() позволяет задать запрос, метод execute выполняет запрос и получает результат. RowSet может поддерживать постоянное соединение с базой данных, а может соединяться с базой только в случае необходимости. В пакете javax.sql имеется класс RowSetProvider, который возвращает фабрику, способную создавать объекты разных типов, реализующих интерфейс RowSet.

javax.rowset.*

Единый интерфейс для всех операций

RowSet extends ResultSet

- setUrl(),
- setUsername(),
- setPassword(),
- setCommand("Select * from ...");

- execute()
- next()
- getXXX()

```
RowSetFactory factory = RowSetProvider.newFactory();
factory.createJdbcRowSet();
```