ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

Конструирование программного обеспечения (2021 - 2022)

Лекция 25

Java DataBase Connectivity (JDBC)

к.т.н. Гринкруг Е.М. (email: egrinkrug@hse.ru)

Java DataBase Connectivity

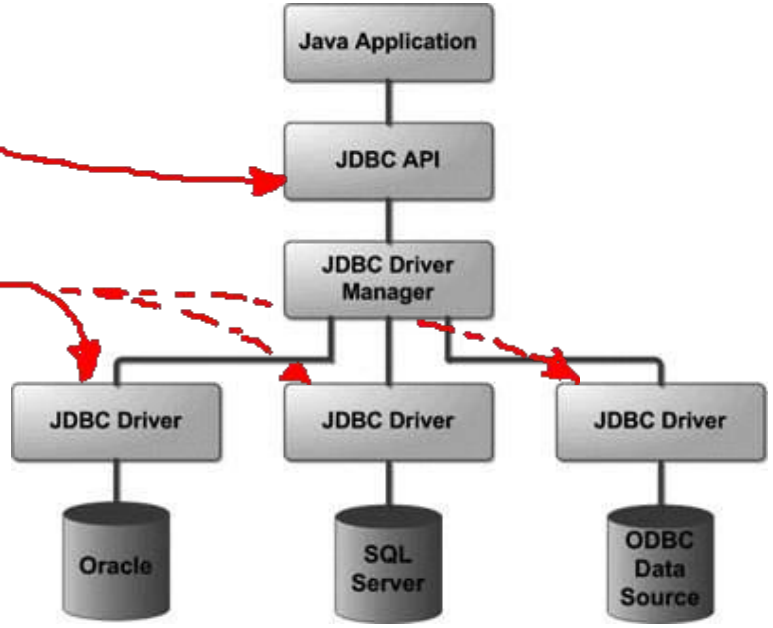
- Это – интерфейс для организации доступа Java-приложений к реляционным базам данных с помощью SQL (*что это?*)
- Со времени появления в 1996 г. по настоящее время интерфейс развивался и стал одним из наиболее используемых API в Java-библиотеках (эти средства – ровесники Java-платформы и главное поле деятельности Oracle).
- К моменту разработки JDBC имелось много разных DB и средств (протоколов) работы с ними; необходимо было достичь максимально-возможного обобщения и совместимости (при работе с «чистой» Java);
- Архитектура строилась под влиянием Microsoft ODBC (**O**pen **D**ata **B**ase **C**onnectivity) и с учетом SQL стандарта :
  - ***Программы, написанные с использованием JDBC API, общаются с driver manager'ом***
  - ***Driver manager общается с драйвером конкретной DB.***

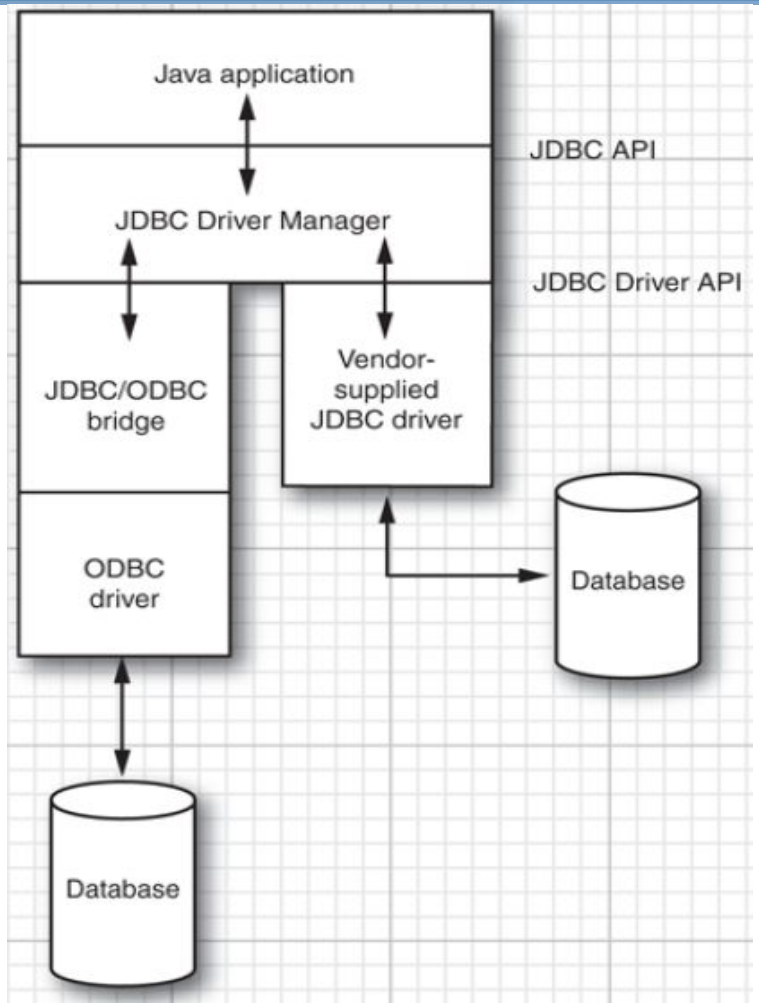
JDBC Connection

- Прежде чем использовать базы данных в Java-программах, полезно понять, что такое JDBC, и почему это понадобилось вообще...
  - **JDBC** – **J**ava **D**ata**B**ase **C**onnectivity – стандартная спецификация API для передачи данных от frontend'а к backend'у. Этот API состоит из Java классов и интерфейсов и служит средством взаимодействия между Java-программами и базами данных.
  - **JDBC** – это развитие **ODBC** (**O**pen **D**ata**B**ase **C**onnectivity), платформо-зависимого аналога - интерфейса, имевшегося для других языков программирования.
- Шаги, выполняемые при соединении java-программы с базой данных:
  - Импортирование базы данных;
  - Загрузка драйвера (методом forName());
  - Регистрация драйвера (используя Driver Manager);
  - Установка соединения (Connection);
  - Создание оператора (Statement);
  - Выполнение запроса (Query);
  - Закрытие соединения.

Архитектура JDBC

- Архитектура JDBC состоит их двух уровней:
  - **JDBC API**: соединяет приложение с JDBC Driver Manager'ом.
  - **JDBC Driver API**: поддерживает соединение JDBC Driver Manager'а с конкретным Driver'ом.
- JDBC использует Driver Manager и специфические Driver'ы для поддержки взаимодействия с различными СУБД.
- Возможно одновременное соединение с различными СУБД с их Driver'ами.





Прикладные программисты работают только с JDBC API

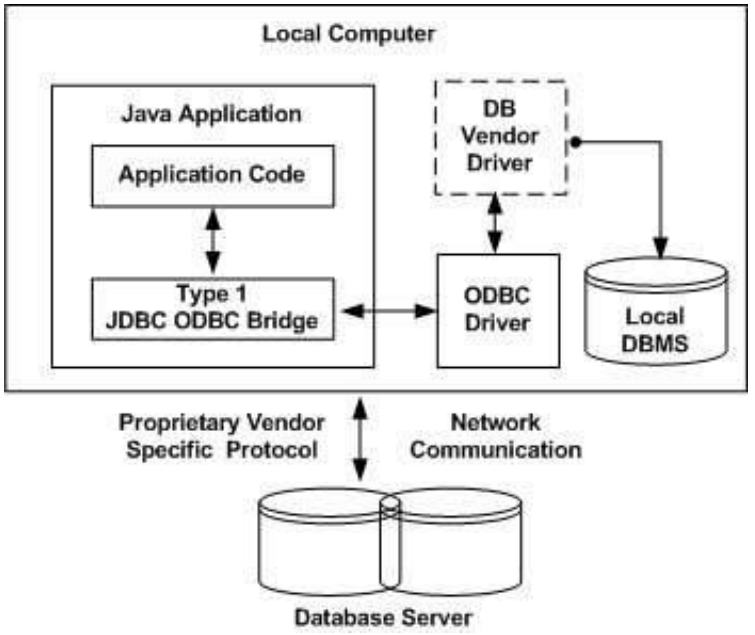
- При этом разработчики DB могут удобно встраивать свои продукты в такую архитектуру
- Это позволяет единообразно использовать разные протоколы общения с разными DBMS разных производителей.
- Эта идеология была использована в Microsoft ODBC (**O**pen **D**ata**B**ase **C**onnectivity) - стандарте (для общения C-программ с разными DBMS – платформо-зависимым образом).

Типы JDBC драйверов

- JDBC-драйверы реализуют интерфейсы JDBC API при работе с соответствующим DB-сервером.
- Пакет **java.sql** из JDK содержит классы и интерфейсы, которые определяют общие поведения, а фактические реализации предоставляются конкретными драйверами, которые реализуют общий интерфейс **java.sql.Driver**.
- Реализации драйверов для разных СУБД и для разных аппаратных и программных платформ весьма многочисленны и разнообразны.
- Исторически принято разделять все возможные реализации на четыре категории, которые рассматриваются ниже…

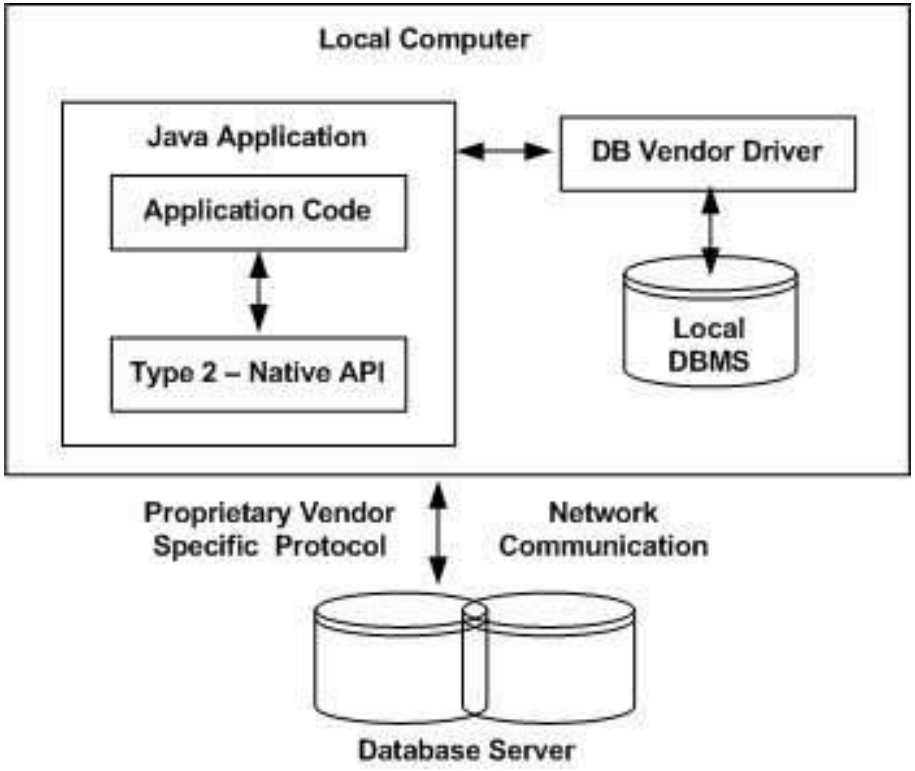
Type 1: JDBC-ODBC Bridge Driver

- JDDC Bridge используется для доступа к ODBC-драйверам, установленным на каждой клиентской машине.
- При этом требуется эти ODBC-драйверы конфигурировать.
- Этот мост был полезен давно, когда все СУБД поддерживали только ODBC …
- Такой подход уже давно не рекомендуется и не используется (и не поддерживается в новых JDK, хотя во времена JDK1.2 был популярен)...



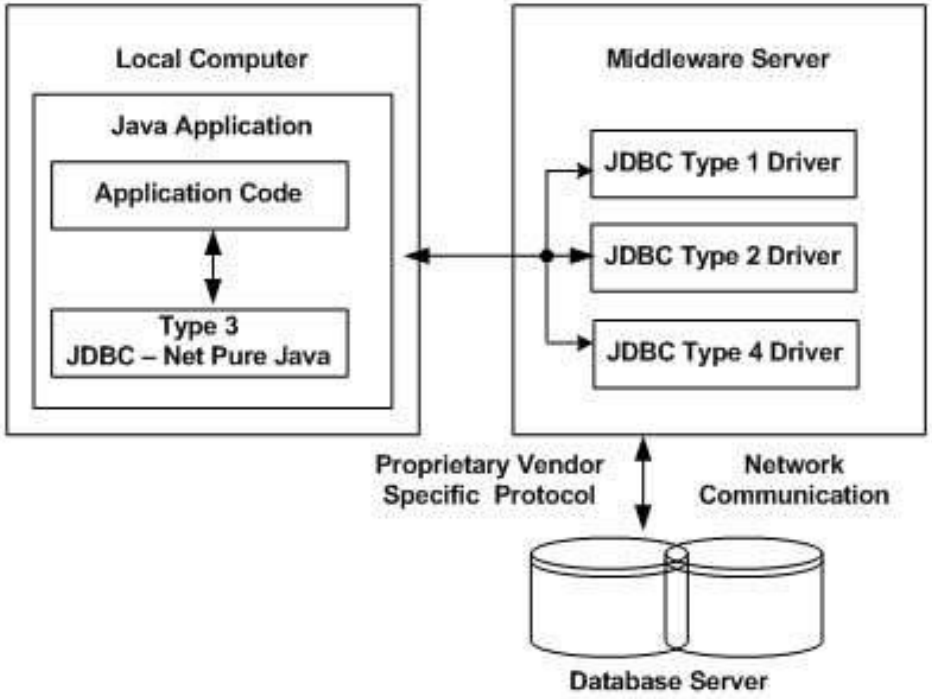
Type 2: JDBC-Native API

- JDBC-вызовы преобразуются в native C/C++ вызовы, которые уникальны для данной СУБД.
- Такие драйверы поставляются разработчиками СУБД и устанавливаются у каждого клиента (аналогично ODBC).
- При замене СУБД их надо тоже заменять (но они эффективнее моста).



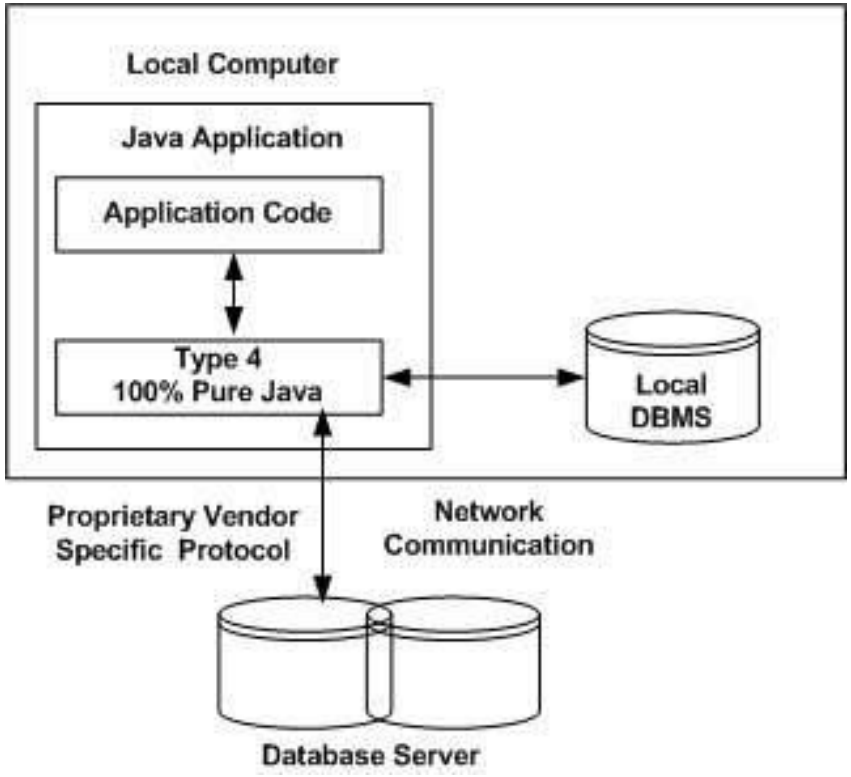
Type 3: JDBC-Net pure Java

- Используется 3-х уровневый подход организации доступа к базам данных;
- JDBC-клиенты взаимодействуют с промежуточным сервером по сети, а тот обеспечивает общение с СУБД в нужном ей формате;
- Это – гибкое решение: оно не требует установки кода на клиенте, один драйвер может дать доступ к нескольким БД…



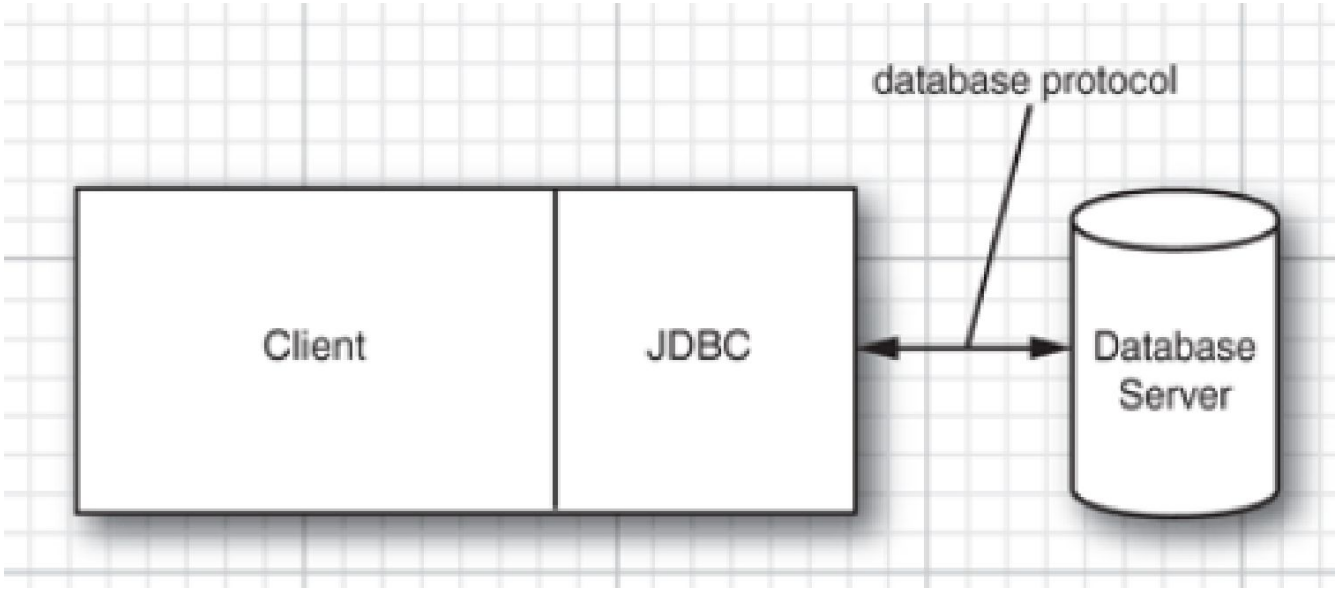
Type 4: 100% Pure Java

- Чисто Java-implemented драйвер взаимодействует по сети с СУБД, предоставляя наибольшую производительность;
- Поставляется производителями СУБД, ничего специфического не надо устанавливать ни на клиенте, ни на сервере, может загружаться динамически;
- Производители СУБД обычно предоставляют такие драйверы из-за специфики своих протоколов



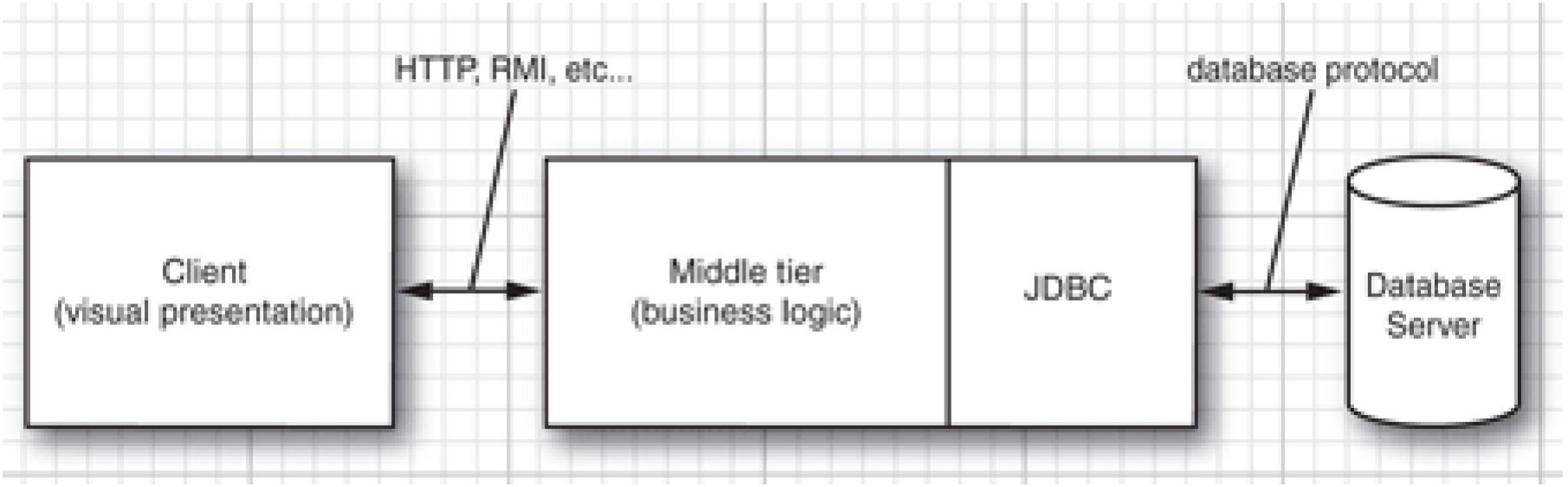
Типичные применения JDBC

- Традиционная модель «клиент-сервер» с богатым интерфейсом на клиенте (rich client GUI) и DB на сервере; драйвер устанавливается на клиенте



- Теперь чаще используют трех- (или N-) уровневую модель (N > 2), где клиент не общается с DB «непосредственно». Он общается с промежуточным уровнем (middleware) на сервере, а тот уже общается с базой.
- Преимущества 3-уровневой модели:
  - Отделение визуального представления (на клиенте) от «бизнес-логики» (в middleware) и собственно данных (в DataBase);
  - Можно одновременно работать многочисленным и разным клиентам (Java-приложениям, web-формам...).
- Взаимодействие клиента с middleware-сервером может происходить по HTTP (если клиент – это web-браузер), или как-то еще;
- JDBC управляет взаимодействием middleware и back-end DB;
- Возможны многочисленные вариации такой многоуровневой модели.

Трехуровневое приложение



Компоненты JDBC API

JDBC API предоставляет следующие основные интерфейсы и классы:

- **DriverManager:** управляет списком DB-драйверов, сопоставляя запросы на соединение от приложения с подходящим DB-драйвером с использованием под-протокола взаимодействия.
- **Driver:** этот интерфейс управляет взаимодействием с DB-сервером. Непосредственно взаимодействовать с объектом-драйвером надо очень редко; обычно им управляет DriverManager.
- **Connection:** интерфейс со всеми методами общения с database, который представляет контекст взаимодействия.
- **Statement:** объект для выдачи SQL-операторов в database.
- **ResultSet:** хранит данные, полученные в результате выполнения SQL-запроса с помощью объектов Statement и работает как их итератор.
- **SQLException:** работает со всевозможными ошибками работы с database.

The Structured Query Language (SQL)

- JDBC позволяет общаться с DB посредством SQL – командного языка для практически всех современных реляционных баз данных;
- Локальные («настольные») DB обычно имеют свой GUI и могут обойтись без SQL, но доступ к серверным DB возможен только по SQL;
  - **Пример «локальной» системы – Power Builder tool…**

- Можно считать, что **JDBC** – это не более чем **API для взаимодействия с базами данных с помощью SQL-операторов**.

- База данных – набор именованных таблиц со строками и столбцами;
  - **Каждый столбец имеет свое имя;**
  - **Каждая строка содержит набор взаимосвязанных данных (иногда называемых *записями*);**

- *Что вам известно про SQL? Кто уже пробовал его использовать?*
- *Кто такой Эдгар Франк Кодд? Чем он знаменит?*

Реляционные базы данных

- База данных – совокупность средств хранения/извлечения информации (не путать с конкретной базой данных, например - телефонных номеров, купленной на Горбушке)…
- Реляционная база данных – база данных, где информация представлена в виде таблиц, состоящих из строк и столбцов;
  - Таблица называется отношением (relation) в том смысле, что она является коллекцией объектов одного типа (строк таблицы, элементы которых как-то соотнесены вместе друг с другом).
- Данные в таблице могут быть собраны в соответствии с общими ключами или концепциями;
- Извлечение данные из таблиц - основное свойство реляционной БД.
- Система управления базами данных (СУБД) – или DBMS (Data Base Management System, - *англ.*) – определяет, как данные хранятся, поддерживаются и извлекаются.
- Важный частный случай – Relational DBMS, т.е. RDBMS...

Правила целостности (Integrity Rules)

- Реляционные базы соблюдают правила, обеспечивающие сохранность и доступность данных в них
  - **Все строки в реляционной таблице должны быть разными;**
  - **Значениями столбцов не могут быть повторяющиеся группы или массивы;**
  - **Используется концепция «пустого» (null) значения, которое нельзя сравнивать;**
- Если все строки различны, можно использовать один или более столбцов для идентификации конкретной строки;
- Такой столбец (или группа столбцов) называется «первичным ключом» (primary key);
  - **Никакой столбец из первичного ключа не может содержать значение null** (это требование называется *entity integrity rule*).

Пример – Employees Table

- Какой столбец (столбцы?) можно использовать как primary key?
  - Почему ? Почему не другие ?

(пример взят из https://docs.oracle.com/javase/tutorial/jdbc/overview/database.html)

Employee_Number	First_name	Last_Name	Date_of_Birth	Car_Number
10001	John	Washington	28-Aug-43	5
10083	Arvid	Sharma	24-Nov-54	null
10120	Jonas	Ginsberg	01-Jan-69	null
10005	Florence	Wojokowski	04-Jul-71	12
10099	Sean	Washington	21-Sep-66	null
10035	Elizabeth	Yamaguchi	24-Dec-59	null

Оператор SELECT

- SQL – язык, созданный для работы с реляционными базами даных; есть набор стандартных SQL-команд, выполняемых всеми RDBMS;
- Для извлечения информации из таблицы используется оператор (команда, запрос) SELECT, где указывается:
  - Один или более заголовков столбцов;
  - Одна или более таблиц, откуда извлекать информацию;
  - Некоторый критерий для извлечения;
- RDBMS возвращает строки для тех столбцов, которые удовлетворяют условию извлечения:

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number IS NOT NULL
```

FIRST_NAME	LAST_NAME
John	Washington
Florence	Wojokowski

- Следующий код выдаст ***result set***, состоящий из всей таблицы:

```
SELECT *
FROM Employees
```

(* - означает SELECT all columns; без ограничений, т.к. нет **WHERE**)

- Оператор **WHERE** определяет критерий выбора; следующий код выберет только тех, у кого фамилия начинается с символов ‘Washington’

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Last_Name LIKE 'Washington%'
```

(‘%’ - ноль или более дополнительных символов; ‘_’ – один любой символ)

- Можно сравнивать числа (и могут быть весьма сложные where-операторы):

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number > 10005
```

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Employee_Number < 10100 and Car_Number IS NULL
```

Join

- Отличительной чертой RDBMS является возможность извлечения данных из нескольких таблиц с помощью join'ов («соединенных запросов»);
- Пусть есть еще одна таблица – Cars (см. пример ниже);
- Должен быть столбец, общий для двух таблиц, чтобы соотнести их друг с другом. Такой столбец, являющийся первичный ключом в одной таблице, называется внешним ключом для другой таблицы;
  - Так, **Car_number** – это первичный ключ в Cars и внешний ключ в Employees.
- Внешний ключ может быть либо null, либо указывать на существующее первичное значение в таблице, на которую он указывает. Первичный ключ не может быть null.

Car_Number	License_Plate	Mileage	Year
5	ABC123	5000	1996
12	DEF123	7500	1999

Пример Join'а

```
SELECT DISTINCT Employees.First_Name, Employees.Last_Name,
    Cars.License_Plate, Cars.Mileage, Cars.IssueDate
FROM Employees, Cars
WHERE Employees.Car_Number = Cars.Car_Number
```

FIRST_NAME	LAST_NAME	LICENSE_PLATE	MILEAGE	YEAR
John	Washington	ABC123	5000	1996
Florence	Wojokowski	DEF123	7500	1999

Замечание: приведенные здесь примеры взяты из учебника Oracle и содержат некоторые неточности (ошибки), которые исправлены в примерах семинара (в Derby **YEAR** является ключевым словом и не может быть именем столбца, и т.п.). ***Это изложение не может рассматриваться как учебное пособие по SQL.***

Команды SQL

- Команда SQL делятся на две основные категории:
  - Data Manipulation Language (**DML**);
  - Data Definition Language (**DDL**)
- Наиболее употребляемые **DML**-команды:
  - **SELECT**
  - **INSERT**
  - **DELETE**
  - **UPDATE**
- Наиболее употребляемые **DDL**-команды:
  - **CREATE TABLE**
  - **DROP TABLE**
  - **ALTER TABLE**

Result Sets и Cursors

- Строки, удовлетворяющие условию запроса, называются результирующим множеством – *result set*;
- В результате выполнения запроса их может быть ноль, одна или более;
- Пользователь может обращаться только к одной строке в результирующем множестве, что обеспечивает *курсор* – аналог указателя в файле, содержащем строки из result set; курсор указывает, к какой строке осуществляется обращение и позволяет итерировать эти строки;

Транзакции

- Разные пользователи могут общаться с базой одновременно;
- Требуется синхронизация для сохранения целостности данных;
- Транзакция – это набор из одного или более SQL-операторов, выполняющих логическую единицу работы.
- Транзакция заканчивается либо действием commit, либо действием rollback – в зависимости от того, нарушается ли целостность данных;
- Оператор **COMMIT** делает изменения, выполненные операторами транзакции, постоянными в базе данных;
- Оператор **ROLLBACK** отменяет все изменения от транзакции;
- Механизм блокировки (Lock) запрещает двум транзакциям одновременно манипулировать одними данными...

Stored Procedures (Хранимые процедуры)

- Это – группа SQL-операторов, которую можно выполнить, вызвав по имени (как функцию или метод);
- Традиционно их писали на языке общения с DBMS;
- Современные продукты позволяют писать stored procedures на Java (и перемещать их байткоды между разными DBMS);
- Они хранятся в DBMS, и единожды написанные/сохраненные могут использоваться там многократно;

Метаданные

- Базы данных хранят пользовательские данные + информацию о самих себе;
- DBMS имеют системные таблицы, в которых перечисляются все таблицы в базе, их имена, первичные и внешние ключи, хранимые процедуры и т.д. и т.п.
- JDBC предоставляет интерфейс DatabaseMetaData, который позволяет получать всю необходимую служебную информацию.

Конфигурация JDBC

- Было и есть много DBMS – хороших и разных:
  - **IBM DB2,**
  - **Microsoft SQL Server,**
  - **MySQL,**
  - **Oracle,**
  - **Informix,**
  - **PostgreSQL (http://jdbc.postgresql.org),**
  - **и т.д.**
- Мы рассмотрим работу с **Apache Derby** – это RDBMS, полностью реализованная на Java (в open source сообществе Apache)
- Ранее это называлось Java DB и входило в состав JDK (по JDK8 включительно)

Apache Derby

- Apache Derby ранее поставлялась вместе с JDK (до его последних сборок JDK8), но теперь не входит в JDK (с JDK9 и далее). Теперь ее надо брать с сайта: http://db.apache.org/derby/derby_downloads.html
- С Apache Derby можно работать:
  - **в серверном режиме (надо стартовать отдельный сетевой сервер, который будет выполнять запросы к базе данных от разных клиентов), или**
  - **во встроенном режиме (работает в той же JVM, что и само приложение, которое только и может иметь доступ к базе; клиенты могут работать через него).**
- Для наших целей можно взять bin distribution – см. на сайте (ссылка выше).
  - **Обладатели Windows / MAC'ов / Linux'ов берут соответствующий дистрибутив;**
  - **Derby – это java-программа, что сразу «уравнивает» весь зоопарк платформ!**
- Радость – в том, что установка Derby (как Java-программы) проста: надо распаковать архив – и все... (на моей машине – см. C:\distrib\ApacheDerby)

Для начала работы с Derby:

- Заготовьте директорию, где будут ваши базы данных: например,
  …\DERBY (мы это проделаем в Idea на практических занятиях...)
- В командных файлах используется переменная **DERBY_HOME**, указывающая на корень установки Derby на компьютере, например:
  set DERBY_HOME=C:\distrib\ApacheDerby\db-derby-10.15.2.0-bin
- Заготовьте следующие файлы в некоторой директории (для работы с конкретной базой):
  - **<someName>Start.bat**
    java -jar %DERBY_HOME%\lib\derbyrun.jar server start
  - **<someName>Shutdown.bat**
    java -jar %DERBY_HOME%\lib\derbyrun.jar server shutdown
  - **ij.properties**
    ij.driver=org.apache.derby.jdbc.ClientDriver
    ij.protocol=jdbc:derby://localhost:1527/
    ij.database=MYDATABASE;create=true
  - **<someName>IJ_Dialog.bat**
    java -jar %DERBY_HOME%\lib\derbyrun.jar ij -p ij.properties
    - База данных создается относительно текущей рабочей директории; если мы хотим нашу базу данных иметь в подпапке DERBY, то **working dir** должна быть DERBY (cd …\DERBY)

- Для таких дел удобно использовать plugin в Idea для запуска командных файлов (в командной строке) или Far Manager…
- Запустить <…>Start.bat
  - Ответ: Apache Derby network server started and ready to accept connections on port…)
- Запустить <…>IJ_Dialog.bat (в другом консольном окне)
  - создана база и начат диалог...
- Поиграть с полученной базой данных MYDATABASE в диалоге **ij**:
  - **не набирать “ вместо ‘ и не забывать ; в конце**

```
CREATE TABLE Greetings (Message CHAR(20));
INSERT INTO Greetings VALUES (‘Hello, World!');
SELECT * FROM Greetings;
DROP TABLE Greetings;
EXIT;
```

- Запустить <…>Shutdown.bat (из другого окна).

Регистрация класса драйвера

- Derby автоматически должен это делать
  - если в derby.jar есть запись META-INF/services/java.sql.Driver
- Есть два способа зарегистрировать драйвер у Driver Manager'а:
  - Принудительно загрузить класс драйвера и его статический инициализатор обеспечит регистрацию. Например:
    - **Class.forName("org.postgresql.Driver");** // force loading of driver class
  - Задать property **jdbc.drivers** при запуске своего приложения. Например:

    java -Djdbc.drivers=org.apache.derby.jdbc.ClientDriver ProgramName
    - Можно также задать это свойство программно:

    System.setProperty("jdbc.drivers","org.apache.derby.jdbc.ClientDriver");

Выполнение операторов SQL

- См. Хорстмана т.2, глава 5 про JDBC и примеры к этой главе...
- В нашем курсе мы не ставим целью освоение SQL; для этого будут другие, специально предназначенные курсы...
- Но способ работы с SQL в Java с помощью JDBC должен быть виден и понятен...
- Компания Oracle – главная мировая компания–производитель СУБД, серверов и приложений баз данных – не случайно приобрела Sun Microsystems за $ 7.4 миллиарда (в 2010 г.), которая сама приобрела шведскую компанию MySQL AB за $ 1 миллиард (в 2008 г.).

Источники

- Учебник: Хорстманн т.2 глава 5 (11 издание) + Примеры кода
- Материалы семинара

- https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html
- https://db.apache.org/derby/
- http://db.apache.org/derby/docs/10.15/ref/
- http://db.apache.org/derby/docs/10.15/devguide/
- http://www.vogella.com/tutorials/ApacheDerby/article.html
- http://www.ibm.com/developerworks/ru/library/os-ad-trifecta1/index.html
- https://ru.bmstu.wiki/Apache_Derby
- https://coderlessons.com/tutorials/bazy-dannykh/izuchite-apache-derby/apache-derby-kratkoe-rukovodstvo

- **Для MySQL см. сайт https://www.mysql.com/ and do google all that stuff around …**

Appendix: Как проверить существование таблицы

- Рассмотрим, как можно проверить существование таблицы в базе данных с помощью JDBC и стандартного SQL
- JDBC предоставляет стандартные средства чтения и записи данных в базе данных. Помимо данных, хранящихся в таблицах, мы можем читать данные, описывающие собственно базу данных. Для этого мы можем использовать объект DatabaseMetaData, который можно взять из JDBC connection:

  DatabaseMetaData databaseMetaData = connection.getMetaData();
- Этот объект предоставляет много полезных методов. В частности, мы можем напечатать все имеющиеся таблицы:

```
ResultSet resultSet = databaseMetaData.getTables(
  null, null, null, new String[] {"TABLE"});
```

```
while (resultSet.next()) {
    String name = resultSet.getString("TABLE_NAME");
    String schema = resultSet.getString("TABLE_SCHEM");
    System.out.println(name + " on schema " + schema);
}
```

- Так как мы не предоставили три первых параметра, мы получаем все таблицы во всех каталогах и схемах. Мы могли бы сузить наш запрос для, например, одной схемы:

```
ResultSet resultSet = databaseMetaData.getTables(
    null, "PUBLIC", null, new String[] {"TABLE"});
```

Проверка существования таблицы

- Если мы хотим проверить существование таблицы с помощью объекта DatabaseMetaData, нам не надо итерировать по ResultSet, *нам надо просто убедиться, что он не пустой*.
- Создадим таблицу EMPLOYEE:

```
connection.createStatement().executeUpdate("create table EMPLOYEE
(id int primary key auto_increment, name VARCHAR(255))");
```

- Теперь мы может сделать так:

```
boolean tableExists(Connection connection, String tableName) throws SQLException {
    DatabaseMetaData meta = connection.getMetaData();
    ResultSet resultSet = meta.getTables(null, null, tableName, new String[] {"TABLE"});
    return resultSet.next();
}
```

- Замечание. Хотя SQL не является case-sensitive, реализация метода getTables() – является case-sensitive. Даже если мы определим таблицу, назвав ее маленькими буквами, она будет храниться с заглавными буквами. Поэтому, метод getTables() будет работать с именами таблиц, заданными заглавными буквами, и нам надо использовать имя “EMPLOYEE”, а не “employee”.

Проверка существования таблицы с помощью SQL

- Хотя использовать DatabaseMetaData удобно, мы можем захотеть использовать в тех же целях «чистый» SQL. Для этого нам надо посмотреть на таблицу **tables**, расположенную в схеме **information_schema**. Это – часть стандарта SQL-g2, реализованного в большинстве СУБД (с важными исключениями: Oracle, Derby…).
- Мы выполняем следующее:

  SELECT count(*) FROM information_schema.tables //не везде…

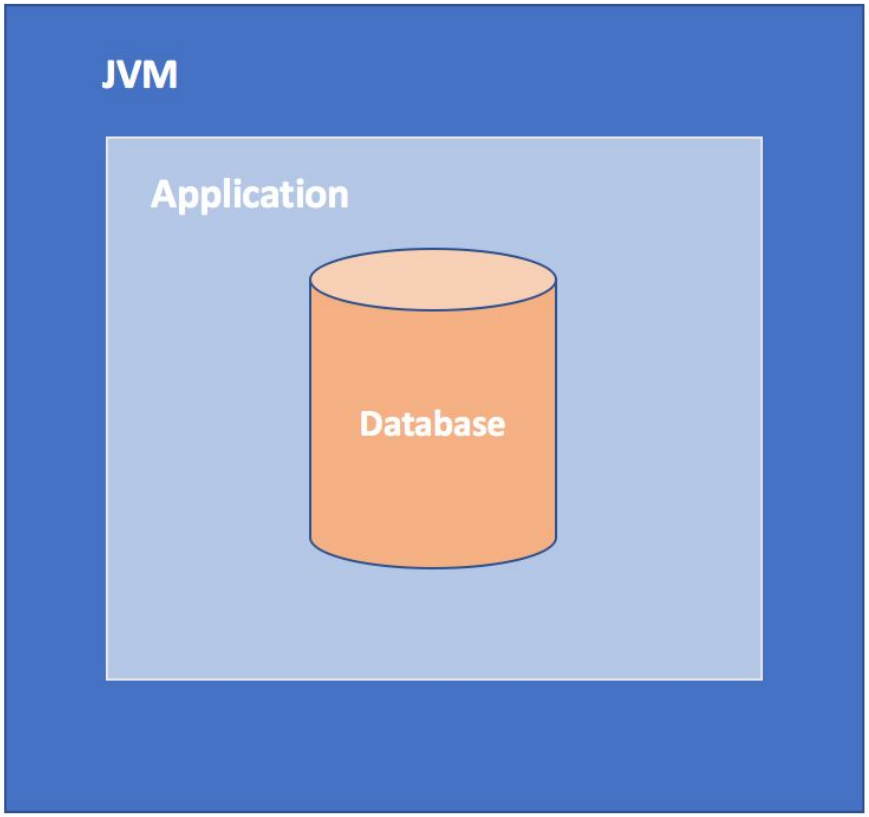
  WHERE table_name = 'EMPLOYEE’

  LIMIT 1; // не везде…
- Это можно сделать в JDBC созданием простого prepared statement’a и проверкой результирующего счетчика на равенство нулю.
- Можно и просто попытаться воспользоваться таблицей («на пробу»)…

Appendix: разные Java embedded СУБД

- Встроенная база данных – это база данных, которая является составной частью приложения.
- Встроенные базы данных могут быть очень полезными на этапах разработки и тестирования, так как они:
  - Легкие в реализации (lightweight);
  - Быстрые;
  - Упрощают тестирование;
  - Просто конфигурируются.
- Есть несколько альтернатив для выбора (со своими плюсами и минусами). Все они поддерживают JDBC API (со встроенными и клиент-серверными конфигурациями) :
  - HSQLDB (HyperSQL Database); http://hsqldb.org/
  - H2; http://www.h2database.com/html/main.html
  - Apache Derby.

Встроенная конфигурация



Источники

- https://www.baeldung.com/jdbc-check-table-exists
- https://dzone.com/articles/3-java-embedded-databases?edition=728528
- http://hsqldb.org/
- http://www.h2database.com/html/main.html
- https://db.apache.org/derby/