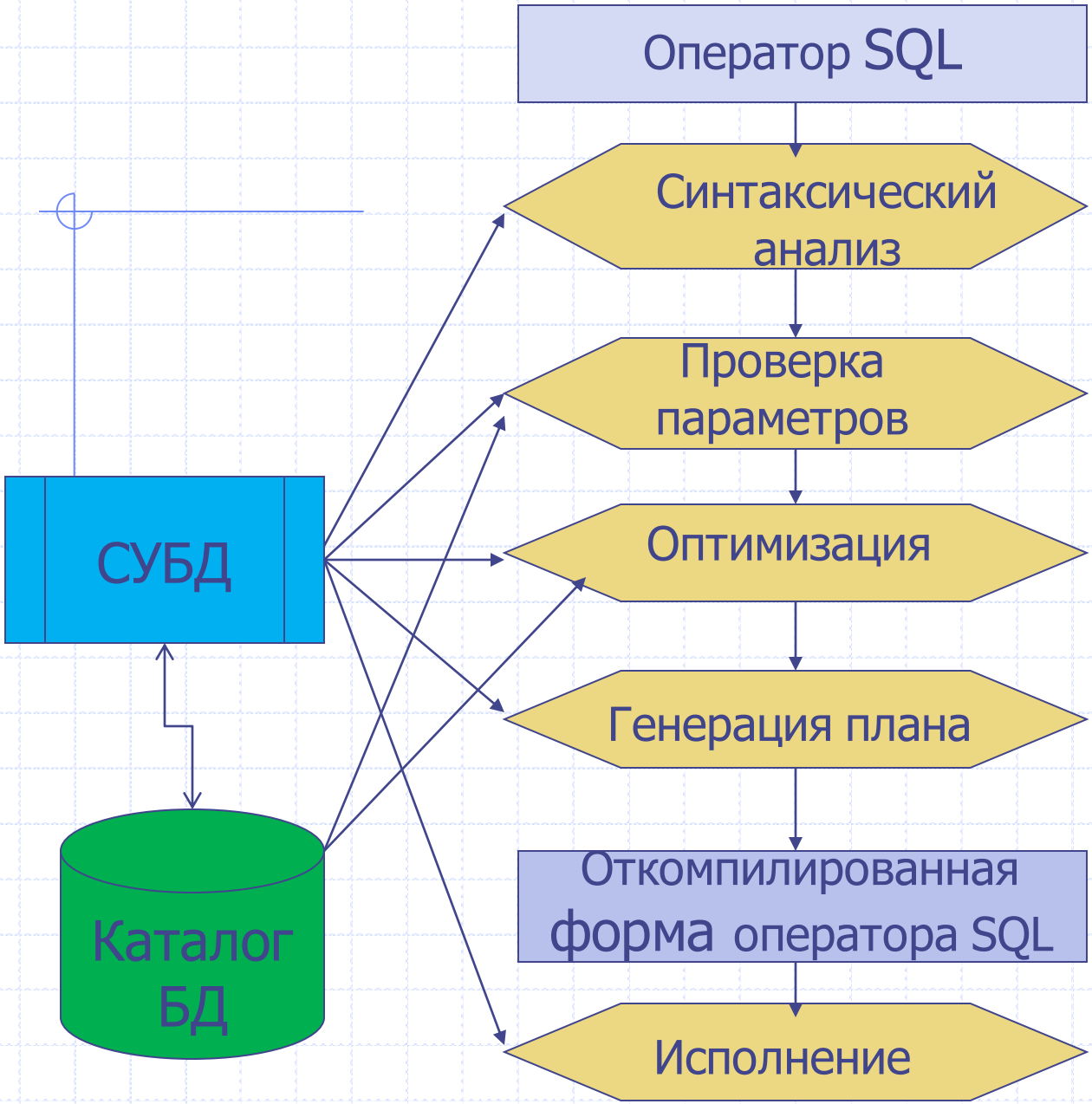
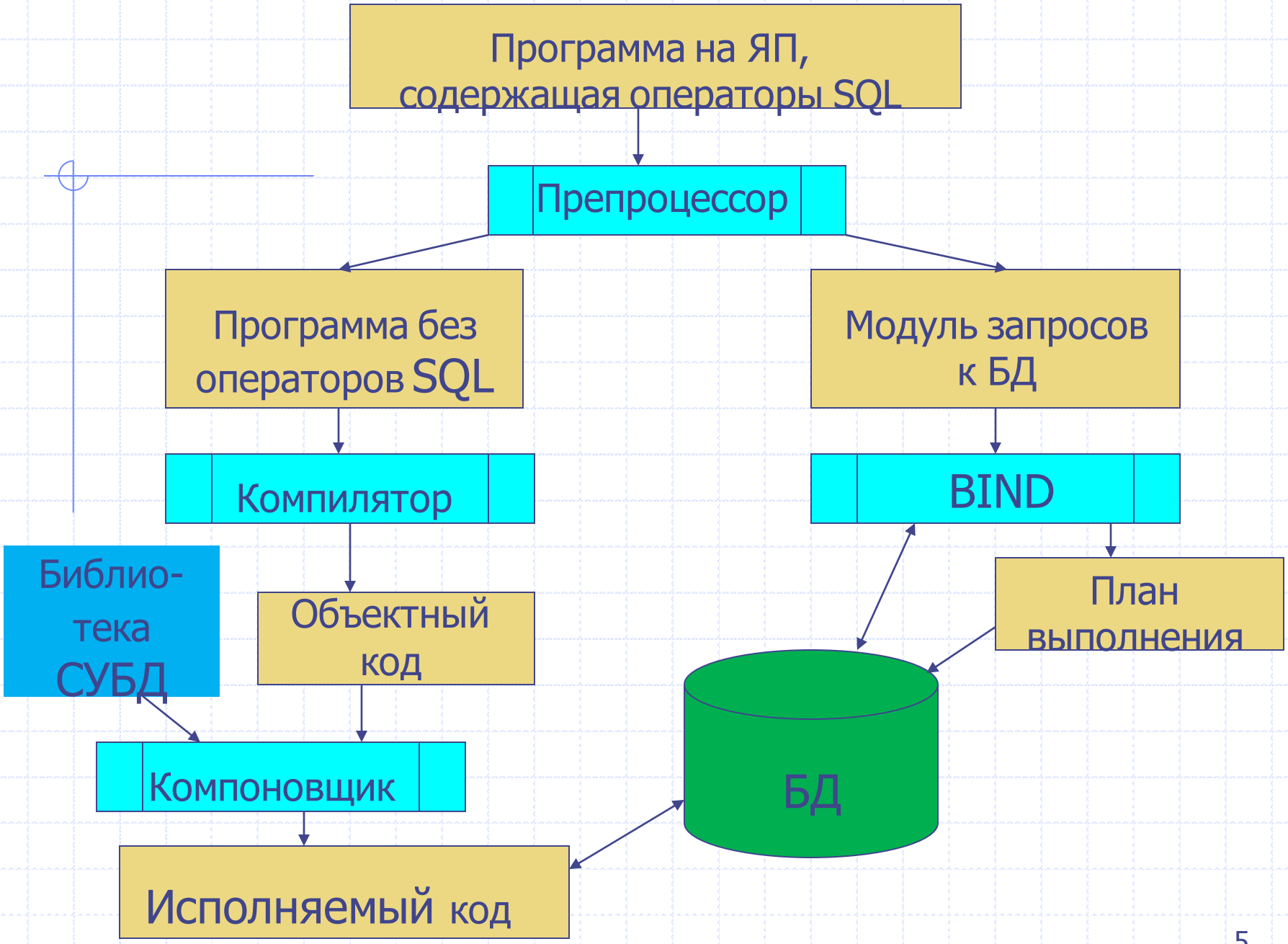
Программирование информационных систем

Выполнение операторов SQL


Оператор SQL
Синтаксический анализ
Проверка параметров
Оптимизация
Генерация плана
Откомпилированная форма оператора SQL
Исполнение
СУБД
Каталог БД

Встроенный SQL


Программа на ЯП, содержащая операторы SQL
Препроцессор
Программа без операторов SQL
Модуль запросов к БД
Компилятор
BIND
Библио-тека СУБД
Объектный код
План выполнения
БД
Компоновщик
Исполняемый код

```
EXEC SQL BEGIN DECLARE  SECTION;
integer emp_no;
char [30] fname, [30]lname;
EXEC SQL END DECLARE SECTION;

EXEC SQL

CONNECT '. . .\employee.fdb'  USER 'ACCT_REC'
PASSWORD 'peanuts';

EXEC SQL

SELECT EMP_NO, FIRSTNAME, LASTNAME

FROM EMPLOYEE WHERE EMP_NO = 1888

INTO :emp_no, :fname, :lname;
```

Основные концепции встроенного SQL

- Смешение операторов базового языка и операторов SQL. Необходимо использовать препроцессор
- Операторы SQL могут ссылаться на базовые переменные
- Для определения NULL значений используются переменные-индикаторы
- Для обработки многострочных результатов запроса используются курсоры
- Для получения служебной информации используется специальная область SQLDA

Проблемы и недостатки

- Если один и тот же запрос требуется повторить многократно?
- Если нужно внести изменения в код?

Динамический SQL

- Оператор SQL формируется программой в виде строки и передается для выполнения
- Все этапы выполнения оператора SQL проходят в процессе выполнения программы

EXECUTE IMMEDIATE

```
char * buffer;
. . .
sprintf(buffer,
        ”delete from employee where emp_no=2”);
EXEC SQL   EXECUTE IMMEDIATE :buffer;
```

Новая проблема -параметры

- При сборке команды в текстовую строку возможны Sql-инъекции

```
string query;
query =”select *from T1 where id=”
                    + param;

param =” 2; DELETE  FROM T1;”;
```

Выполнение в два этапа

```
sprintf(stmtbuf, ”update employee set salary=? where
                                        emp_no=?”);
EXEC SQL PREPARE mystmt FROM :stmtbuf;
if  (!sqlca.sqlcode)
{       new_salary=. . .; empn=. . .;
        EXEC SQL EXECUTE mystmt USING :new_salary,
                                :empno;
        if (sqlca.sqlcode) {
                ….
   }}
```

Запросы на чтение

- Курсор связывается с оператором SELECT
- Работа с курсором – OPEN, FETCH, CLOSE
- Оператор OPEN запрашивает у СУБД описание таблицы результатов запроса и помещает его в SQLDA

```
EXEC SQL
    DECLARE to_be_hired CURSOR FOR
    SELECT D.DEPARTMENT, D.LOCATION,
                                        P.DEPARTMENT
     FROM DEPARTMENT D JOIN DEPARTMENT P
                ON (AND D.HEAD_DEPT = P.DEPT_NO)
          WHERE D.MNGR_NO IS NULL;
EXEC SQL
   OPEN to_be_hired;
```

```
EXEC SQL   FETCH to_be_hired
                    INTO :deptname, :lname, :head_deptname;
while (! sqlca.sqlcode )
{
printf("%s %s works in the %s department.\n",
                            fname,  lname, deptname);
EXEC SQL
    FETCH to_be_hired
          INTO :deptname, :lname, :head_deptname;
}
EXEC SQL CLOSE to_be_hired;
```

Интерфейсы программирования приложений

- API – интерфейсы обеспечивают доступ к БД через библиотеку функций (или через библиотеку классов)

SQL Call-Level Interface

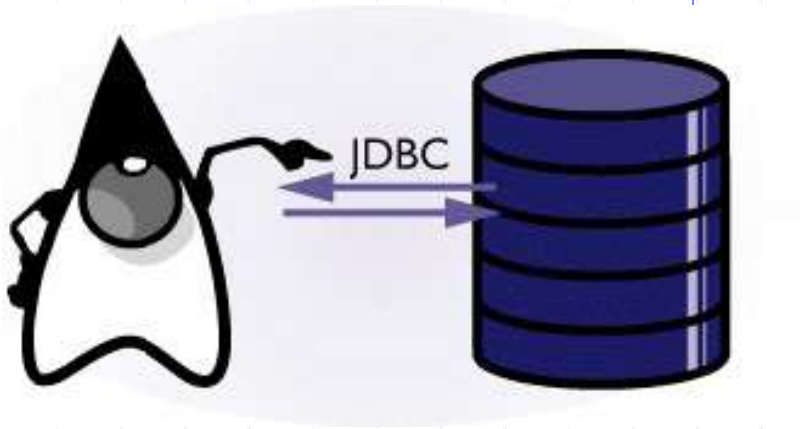
- Стандарт интерфейса X/Open SQL CLI
- Приложения на базовом языке обращаются к БД, используя библиотеку, предоставляемую разработчиком СУБД. Библиотека должна соответствовать стандарту

ПРИМЕРЫ

- ODBC – Microsoft
- JDBC – Sun
- ADO.Net

Работа с базами данных

JDBC



JDBC, также как и Microsoft ODBC и Borland DataBase Engine (BDE), базируется на X/Open SQL CLI (Call Level Interface).

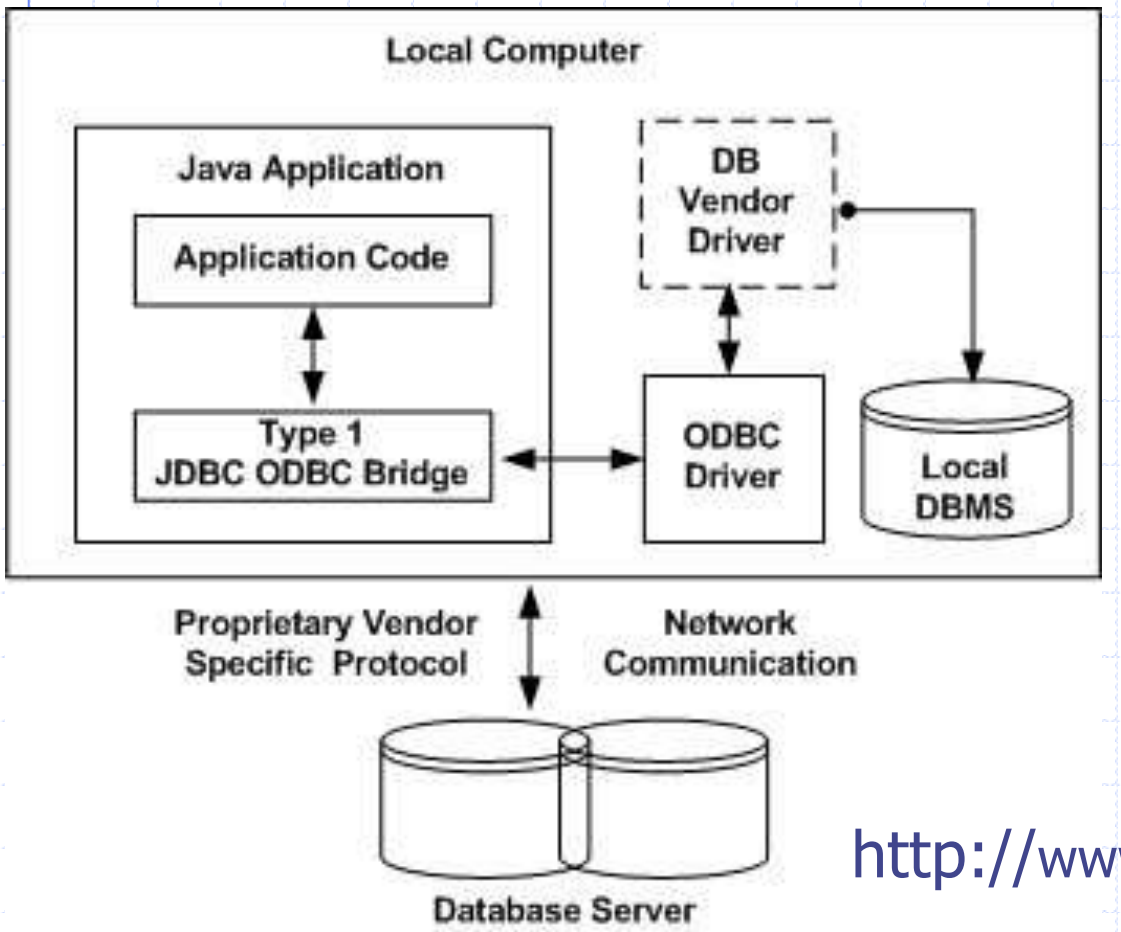
Авторы спецификации обращают особое внимание на то, что их основная задача состоит в описании основных абстракций и концепций, определенных в X/Open CLI, в виде натуральных ("родных") интерфейсов Java.

Что такое JDBC

- В Java – пакет, описывающий интерфейсы для доступа к реляционным БД
- Реализация этих интерфейсов собрана в библиотеку(пакет), называемый JDBC-драйвером.

Типы JDBC драйверов

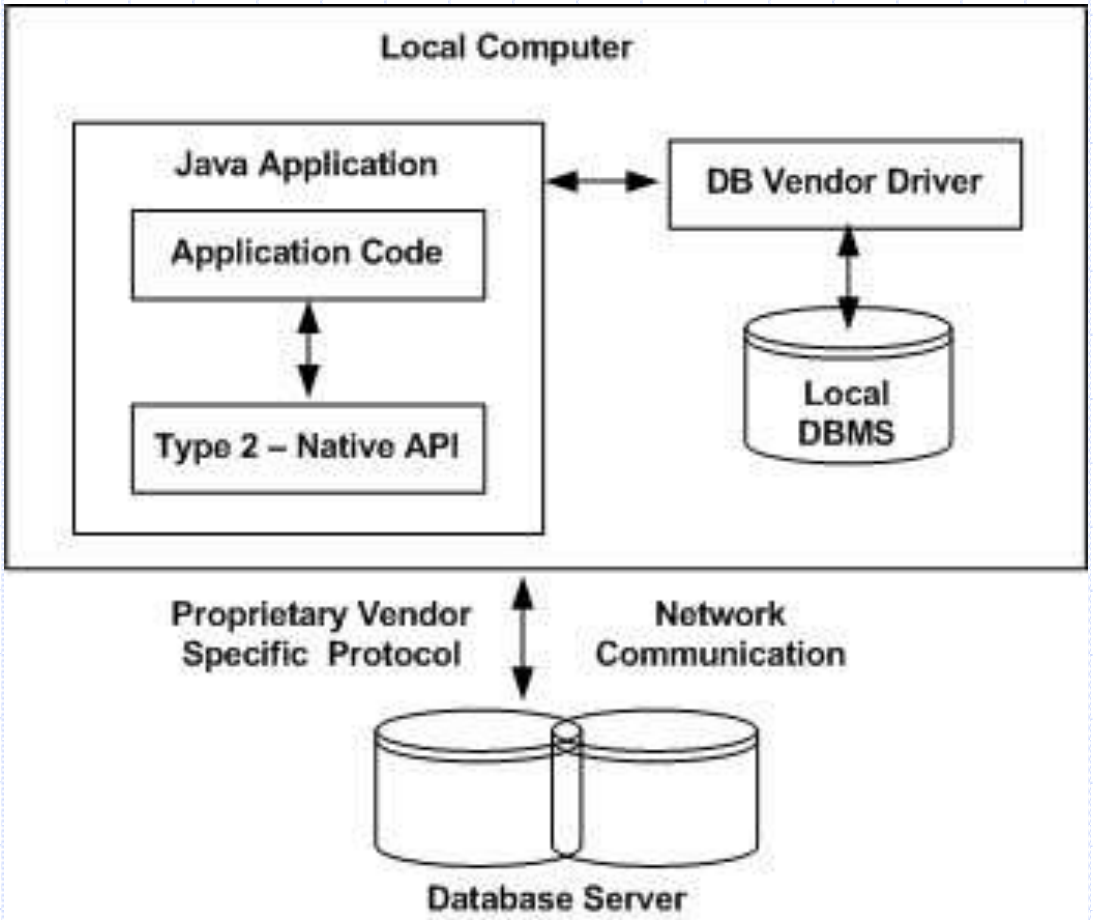
Type 1: JDBC-ODBC Bridge Driver



http://www.tutorialspoint.com/jdbc/

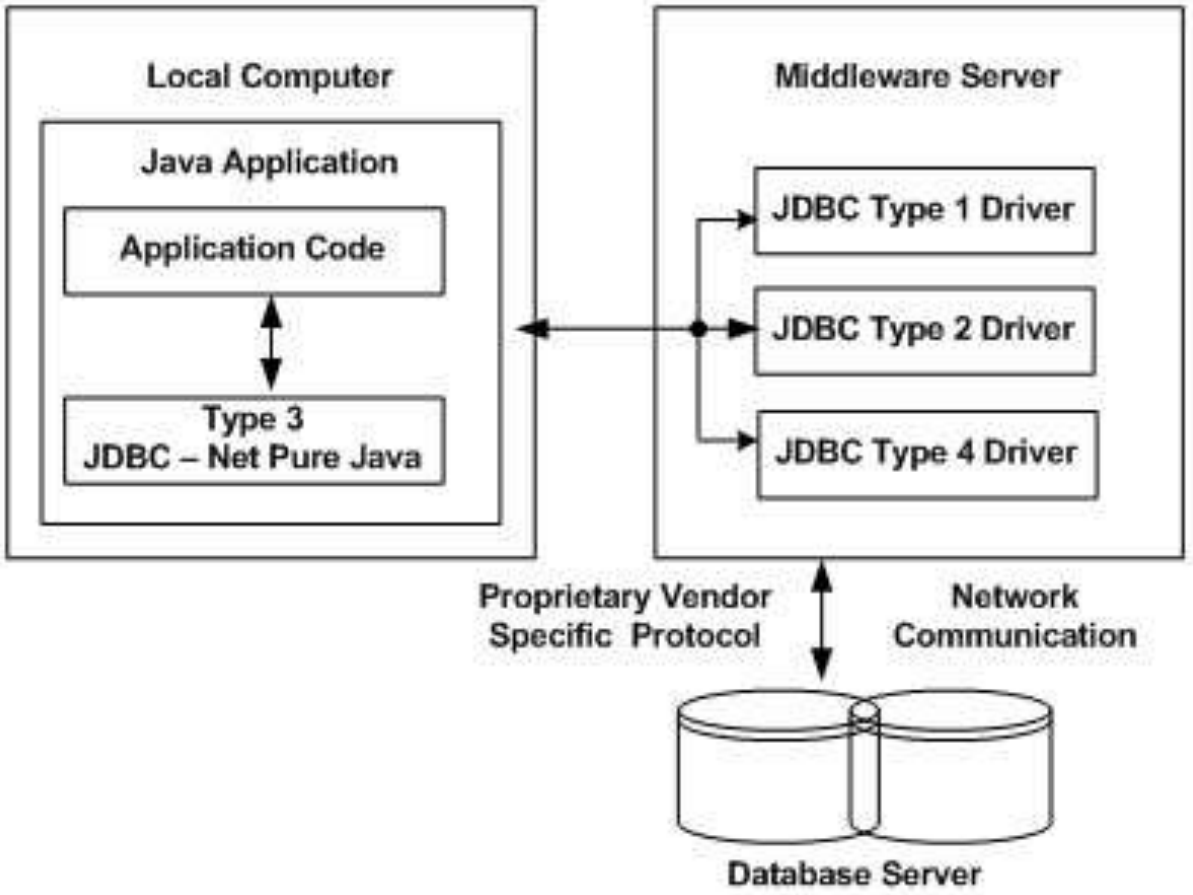
Типы JDBC драйверов

Type 2: JDBC-Native API



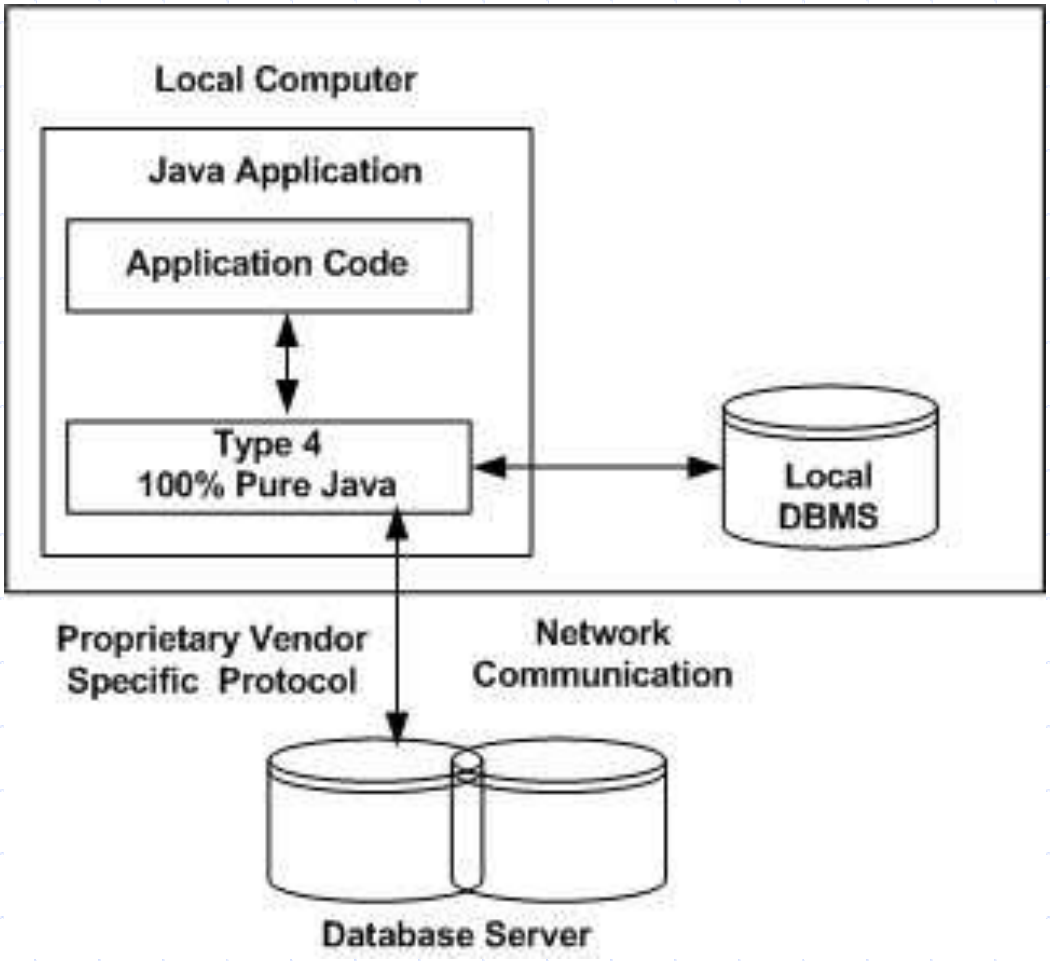
Типы JDBC драйверов

Type 3: JDBC-Net pure Java



Типы JDBC драйверов

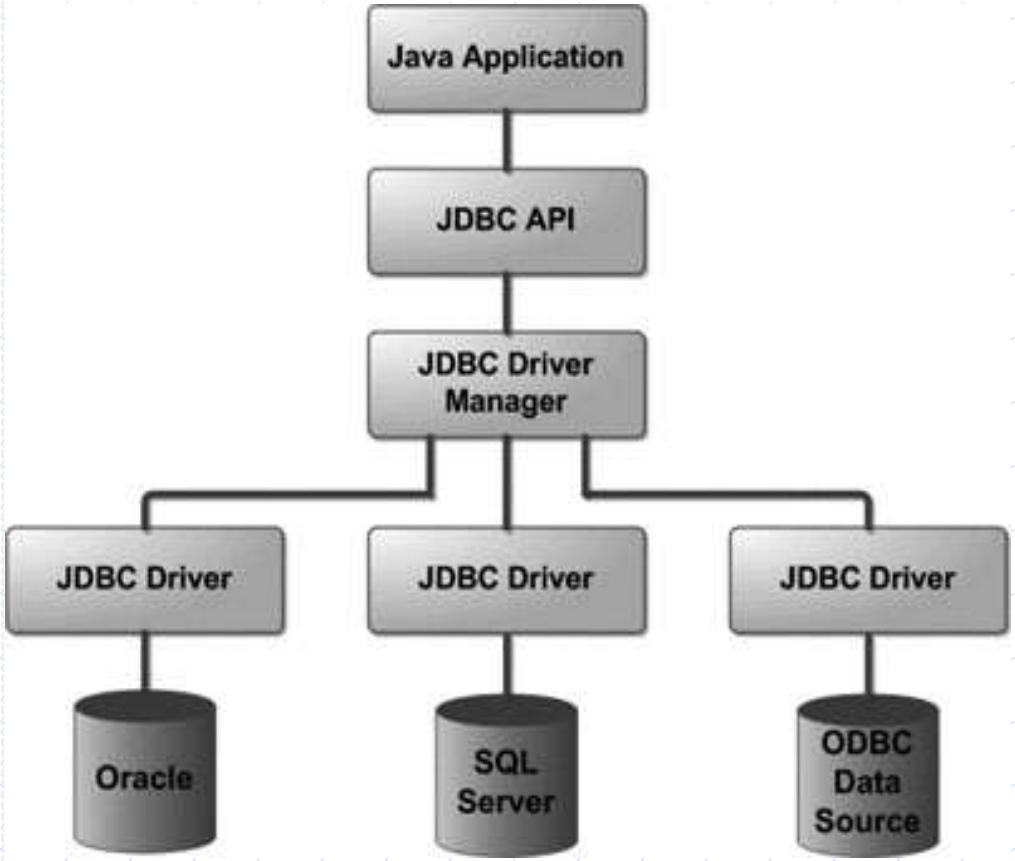
Type 4: 100% pure Java



Основы взаимодействия Java и DB

- JDBC предполагает передачу команды SQL к базе данных в виде строки
- Базовым требованием JDBC, является удовлетворение входного уровня ANSI-стандарта SQL-92

Общая схема доступа



Основные классы и интерфейсы

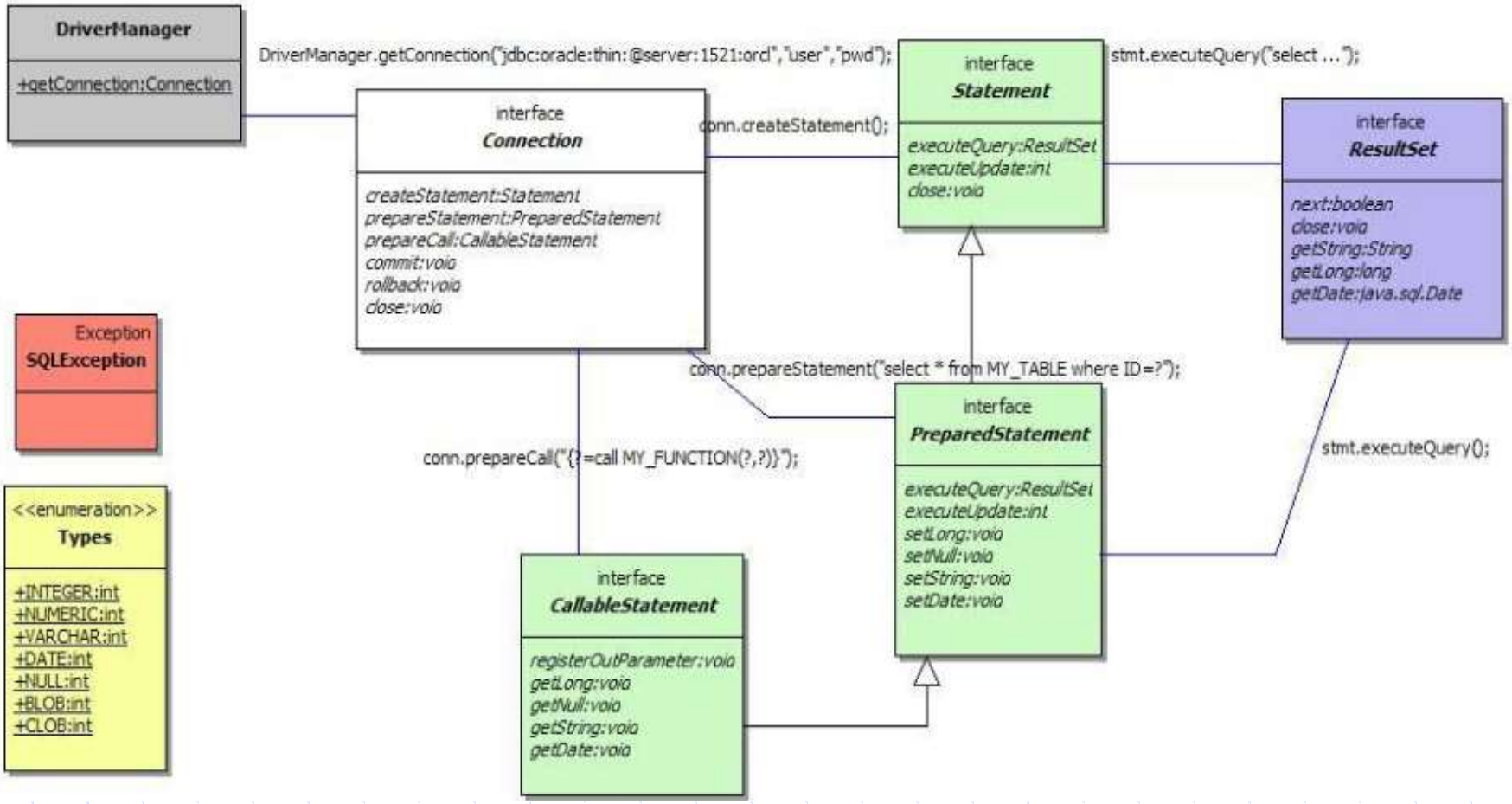


Схема работы с JDBC

- загрузить драйвер в память виртуальной машины (или определить, что он уже загружен);
- получить соединение (Connection) с базой данных;
- подготовить выражение-запрос (Statement);
- выполнить запрос;
- разобрать полученные данные (ResultSet).

JDBC драйверы поставляются в виде библиотек – архивов

Например:

pg74jdbc3.jar

firebirdsql-full.jar

- Класс DriverManager отвечает за выбор нужного драйвера базы данных и создание нового соединения с базой данных
- Классы драйверов обычно имеют имена вида

  org.postgressql.Driver

  org.firebirdsql.jdbc.FBDriver

Регистрация драйвера

- Системное свойство jdbc.drivers

- В командной строке

java –Djdbc.drivers=org.firebirdsql.jdbc.FBDriver <класс>

- В приложении установить

System.setProperty(”jdbc.drivers”,” org.firebirdsql.jdbc.FBDriver”);

- или загрузить класс

Class.forName(” org.firebirdsql.jdbc.FBDriver”);

Соединение с БД

- Через менеджер драйверов.

Метод getConnection() принимает ссылку на БД, имя пользователя и его пароль:

```
Connection con =
    DriverManager.getConnection(url,"Login","Password");
```

Возвратит ссылку на экземпляр класса Connection, представляющий собой соединение с базой данных.

Формат URL для соединения

jdbc: название_подпротокола: другие_сведения

Например

jdbc:firebirdsql:dom.serv.ru/3050:/fbdata/employee.fdb

Выполнение запросов

Statement - интерфейс предназначен для разового выполнения статических запросов

```
Statement stmt = con.createStatement();
ResultSet result =
    stmt.executeQuery
        ("SELECT FULL_NAME, JOB_COUNTRY  FROM EMPLOYEE");
```

Еще методы Statement:

```
bollean execute(. . .)
int executeUpdate(. . .)
```

Предварительно подготавливаемые команды

Это более развитый вид запроса.

Он предварительно компилируется и кэшируется сервером баз данных

PreparedStatement может принимать подставляемые в SQL-выражения параметры

```
PreparedStatement p_stmt =
        con.prepareStatement (
    "SELECT * FROM CUSTOMERS WHERE ID = ? AND LAST_NAME LIKE ?");
```

Подстановка параметров

```
p_stmt.setInt(1, 12345);
p_stmt.setString(2, ”Alex");
```

Выполнение запроса

```
ResultSet result =p_stmt.execute();
```

Результаты запроса

Класс результата - ResultSet
Методы для получения возвращаемых результатов:

boolean next()
String getString(int column)
String getString(String name)
TimeStamp getTimeStamp(…)
и т.д. для всех типов данных

```
while( result.next() )
{
System.out.println(result.getString("FULL_NAME"));
}
```

Интерфейс java.sql.ResultSetMetaData позволяет получить информацию о типах и свойствах столбцов в ResultSet.

Эта возможность особенно важна при построении динамических систем, таких, как среда разработки или инструментарий для написания запросов, когда структуры таблиц базы данных неизвестны заранее.

```
ResultSetMetaData rsM=rs.getMetaData();

System.out.println(rsM.getColumnCount() );

for (int i=0; i<  rsM.getColumnCount(); i++){

   System.out.print(rsM.getColumnName(i+1)
                    +" - "+rsM.getColumnTypeName(i+1));
 }
```

Интерфейс java.sql.DatabaseMetaData запрашивает у базы данных информацию о ее структуре.

Он используется при разработке приложений, которые "не знают" практически ничего о базе данных, к которой обращаются

Хранимые процедуры

Интерфейс CallableStatment используется для обращения к хранимым процедурам

```
CallableStatement c_stmt =
   con.prepareCall( " { ? = call my_stored_procedure ( ?  ) }"
);
c_stmt.registerOutParameter( 1, Types.VARCHAR );
c_stmt.setDouble( 2, 3.14 );
c_stmt.executeQuery();
String s=c_stmt.getString(1);
```

Обработка ошибок

java.sql.SQLException

- getSQLCode()
- getErrorCode()

java.sql.SQLWarning

- getNextWarning()

Встроенные методы драйвера

- Отдельные команды можно выполнять через встроенные методы

Вместо:

```
statement st = con.createStatement();
st.executeQuery("commit");
```

Можно использовать:

```
con.commit();
```

Завершение работы

```
finally{
  try{  if (rs!=null) rs.close();} catch(SQLException e){}
  try{  if (s!=null)  s.close();} catch(SQLException e){}
  try{  if (c!=null)  c.close();} catch(SQLException e){}
  }
```