Обзор средств работы с реляционными СУБД

История API доступа к БД

- До 1990х — у каждого из поставщиков СУБД был собственный API
- Call Level Interface (CLI)
  - разработан SQL Access Group в 1992 (затем — X/Open)
  - международный стандарт (ISO/IEC 9075-3:2003)

История API компании Microsoft для доступа к БД

- Open Database Connectivity (ODBC)
  - разработан Microsoft в 1992, основан на SQL/CLI
  - обычный процедурный API
  - стандартный интерфейс для получения и отправки данных источникам данных различных типов
  - поставщики различных баз данных создают драйверы, реализующие стандартные функции из ODBC API с учетом особенностей их продукта
  - существуют ODBC-драйверы для большинства СУБД и др. источников данных (электронные таблицы, текстовые файлы, XML)
  - существуют реализации для различных ОС
    - Microsoft ODBC, iODBC (для UNIX-систем), unixODBC (входит в большинство Linux-дистрибутивов)

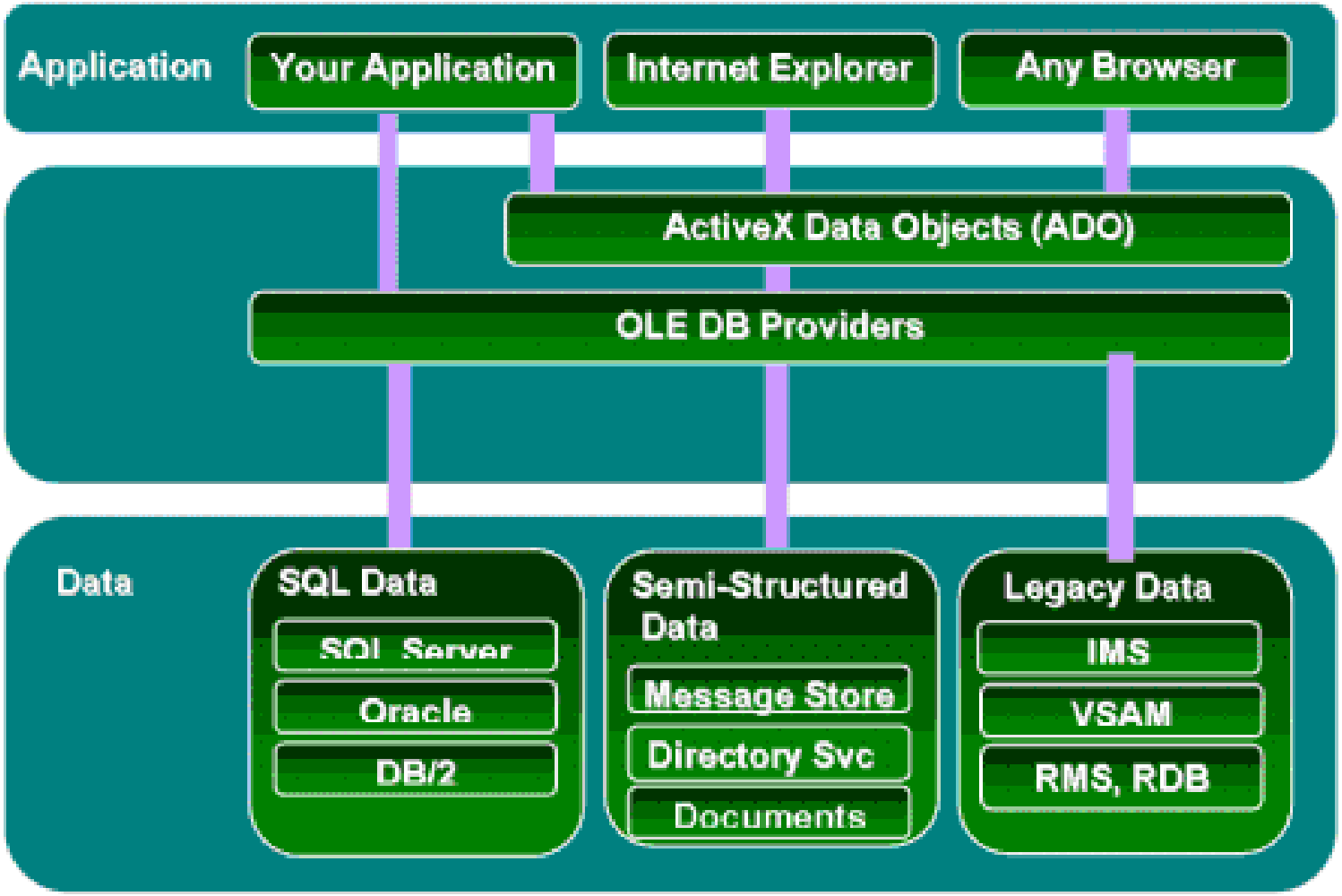
История API компании Microsoft для доступа к БД

- Data Access Objects (DAO)
  - объектно-ориентированный API для VB
  - появился в ноябре 1992 года как API для работы с СУБД Jet (в 2001 поддержка прекращена)
    - технология Jet поддерживала доступ к файлам формата MDB (Microsoft Access), ODBC-источникам данных и к источникам данных ISAM
- Remote Data Objects (RDO)
  - набор COM-объектов, инкапсулирующих ODBC API, и клиентская курсорная библиотека
  - появился в 1995 вместе с VB 4.0
  - RDO ориентирован на обработку данных на стороне сервера БД, в отличие от DAO, ориентированного на обработку данных на стороне клиента

История API компании Microsoft для доступа к БД

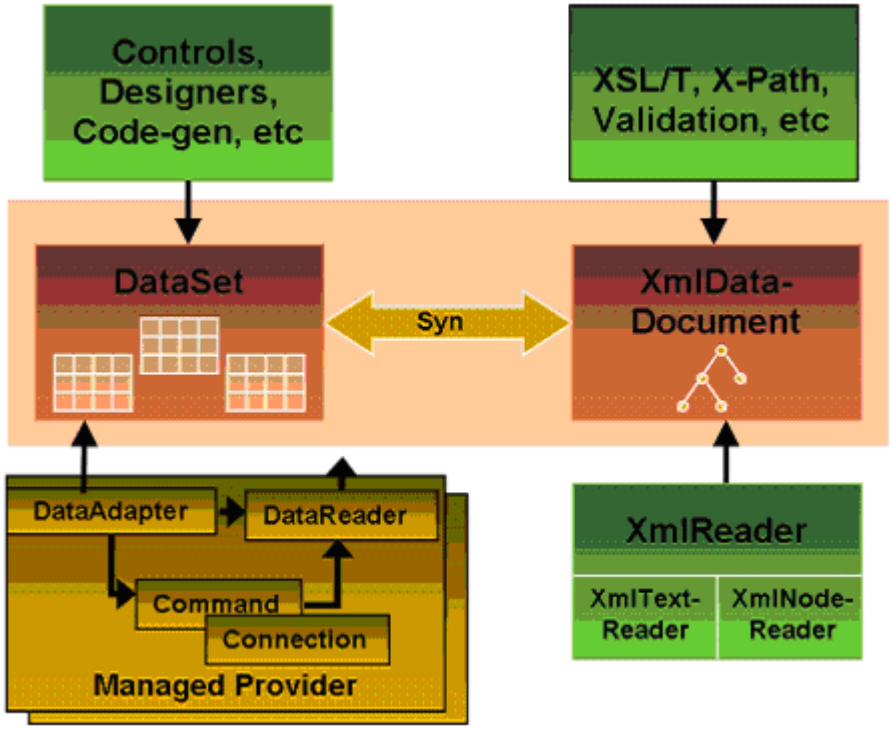
- Object Linking and Embedding, Database (OLE DB)
  - набор COM-интерфейсов, которые позволяют приложениям обращаться к данным из различных источников с помощью унифицированного набора абстракций
    - Не только РСУБД, не только SQL
    - Источник данных, сессия, команда, набор строк
- ActiveX Data Objects (ADO)
  - набор COM-объектов для обращения к источникам данных
  - прослойка между языками программирования и OLE DB
  - представлен в 1996

История API компании Microsoft для доступа к БД



История API компании Microsoft для доступа к БД

- ADO.NET
  - объектно-ориентированный API
  - выпущен в 2002 (последняя версия - 3.5)
  - основная модель доступа к данным для приложений, основанных на Microsoft .NET
  - компоненты ADO.NET входят в поставку оболочки .NET Framework

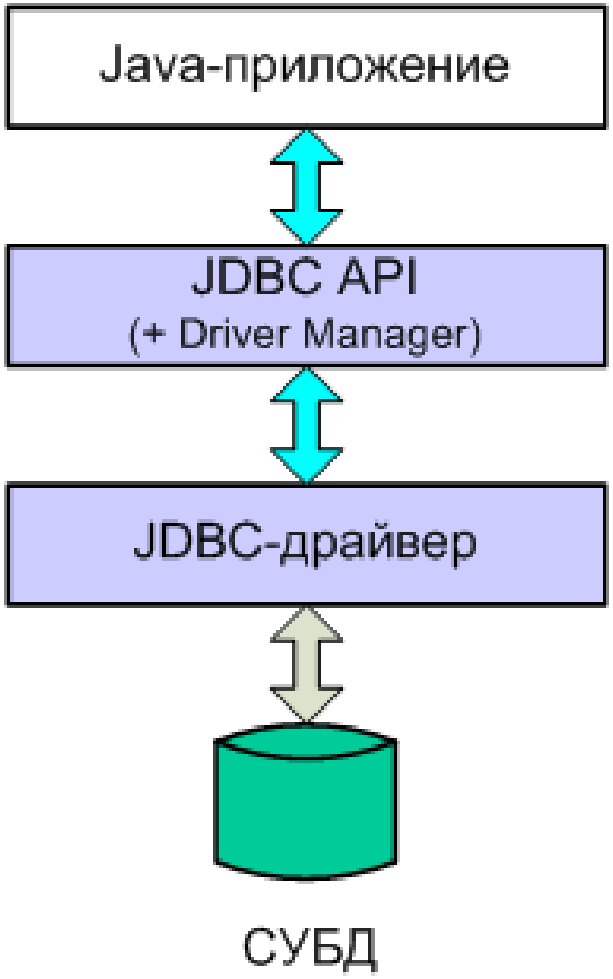


API компании Sun для доступа к БД

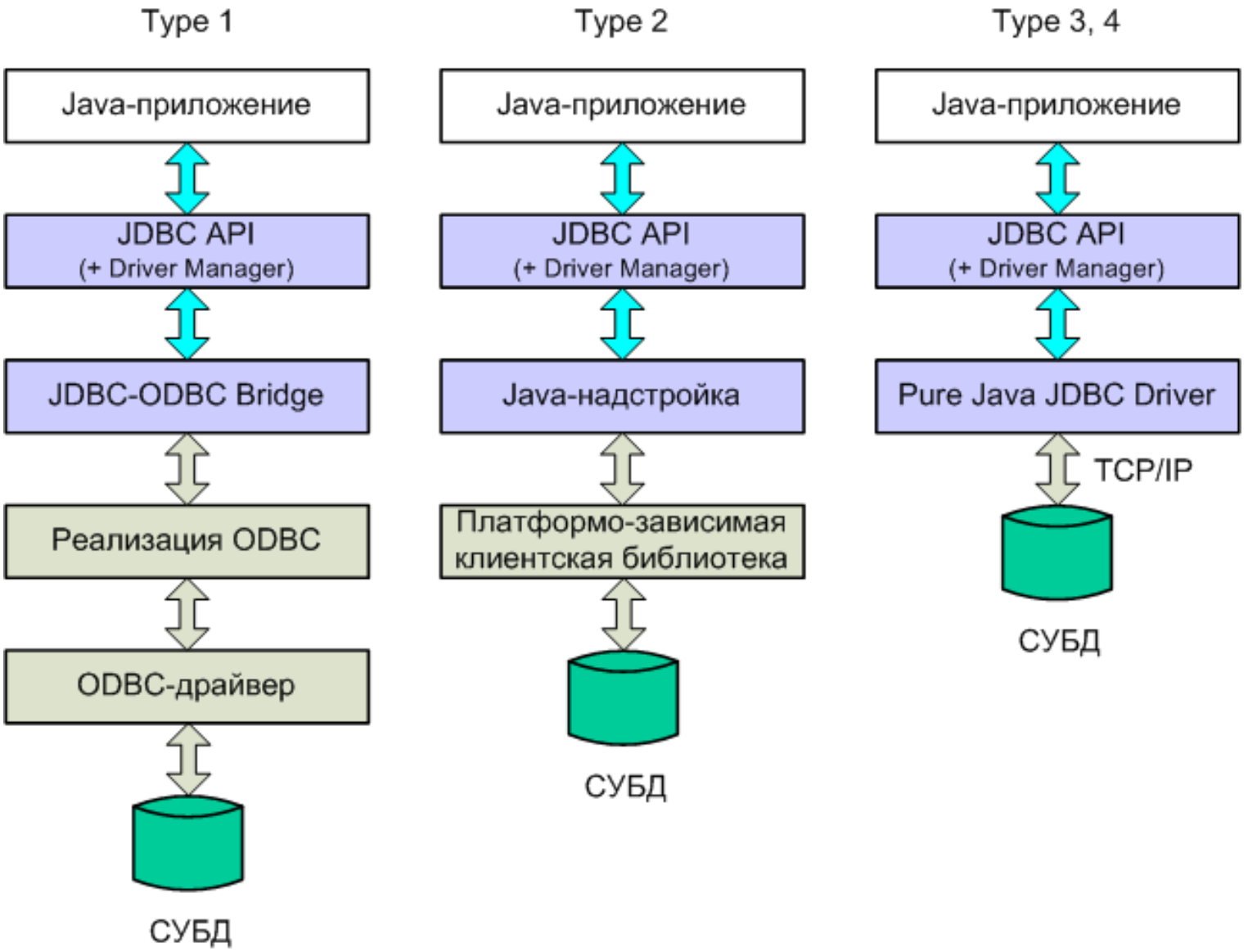
- JDBC™
  - Java Database Connectivity (неофициальная расшифровка)
  - объектно-ориентированный API для обращения к реляционным БД из программ на языке Java
  - входит в состав Java SE
    - Java 1.1 — JDBC 1.0
    - Java SE 6 — JDBC 4.0
  - обеспечивает переносимость приложений, работающий с реляционными БД
    - возможность замены одной СУБД на другую без внесения изменений в исходный код приложения и без его перекомпиляции

Архитектура JDBC

- JDBC API — набор Java-интерфейсов для работы с реляционной БД
  - пакеты `java.sql` и `javax.sql`
- JDBC Driver Manager — менеджер драйверов, абстрагирует приложение от конкретного драйвера
- JDBC-драйвер — взаимодействует с конкретной РСУБД и реализует интерфейсы JDBC API



Типы JDBC-драйверов



Порядок использования JDBC

1. Подключение к БД
2. Создание запроса
3. Выполнение запроса
4. Обработка результатов запроса
5. Обработка мета-данных (необязательно)
6. Освобождение ресурсов

Подключение к БД

- JDBC URL — строка для подключения к базе данных, может включать имя сервера, порт, схему БД, параметры подключения
  - jdbc:subprotocol_name:driver_dependant_databasename
- Имя протокола в JDBC URL однозначно определяет используемый драйвер
  - MySQL Connector/Java
    - jdbc:mysql://localhost/dbname
  - Oracle Thin driver
    - jdbc:oracle:thin:@machinename:1521:dbname
  - Oracle OCI driver
    - jdbc:oracle:oci:@machinename:1521:dbname

Подключение к БД

- java.sql.DriverManager
  - Рекомендуется только для Java SE-приложений
  - Сначала нужно загрузить драйвер и зарегистрировать в менеджере драйверов
    - 1) автоматически при запуске приложения
    - `java -Djdbc.drivers=com.mysql.jdbc.Driver MyApplication`
    - 2) либо загрузив класс из приложения
    - ```
      Class.forName("com.mysql.jdbc.Driver")
            .newInstance();
      ```
    - Не требуется для драйверов JDBC 4.0
  - Затем получить соединение с БД
    - ```
      Connection conn = DriverManager.getConnection(
      "jdbc:mysql://localhost/test?
      user=monty&password=greatsqldb");
      ```

Подключение к БД

- javax.sql.DataSource
  - Интерфейс, который реализуется в конкретном JDBC-драйвере
  - Выступает в роли «фабрики соединений»
    - ```
      DataSource ds = ...;
      Connection con = ds.getConnection();
      ```
  - Рекомендуется для Java EE-приложений, можно использовать и в Java SE-приложениях
  - Три вида реализаций:
    - Базовая
    - С поддержкой пула физических соединений
    - С поддержкой распределенных транзакций
      ```

Подключение к БД — DataSource

- При создании физического подключения к СУБД используются параметры источника данных

Название параметра	Тип параметра	Описание
`databaseName`	`String`	Имя физической базы данных
`serverName`	`String`	IP-адрес или доменное имя компьютера, на котором расположена СУБД
`portNumber`	`int`	Номер порта, который слушает СУБД для входящих запросов на подключение
`user`	`String`	Имя пользователя, под которым выполняется подключение
`roleName`	`String`	Имя роли пользователя, с которой выполняется подключение
`password`	`String`	Пароль пользователя

- Конкретный класс источника данных и параметры определяются в конфигурационном файле сервера приложений
- Также можно программно создать и настроить нужный объект источника данных
  - `com.mysql.jdbc.jdbc2.optional.MysqlDataSource`

Подключение к БД — пул соединений

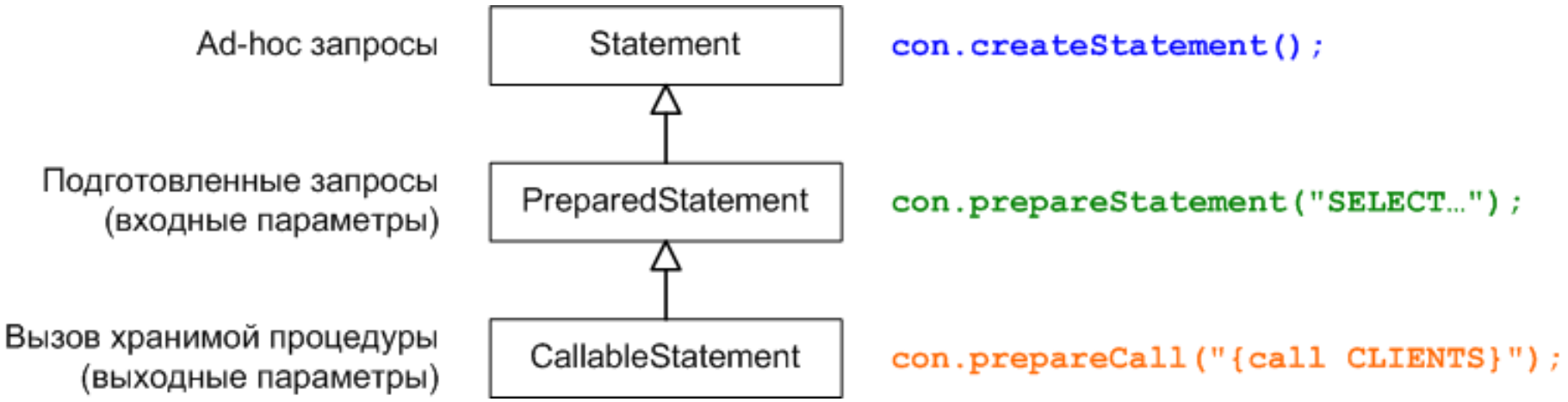
- Соединение с БД — это дорогой и ограниченный ресурс
  - С помощью пула малое количество физических соединение разделяется между большим количеством клиентов
  - Приложение работает с «логическим» соединением, но это скрыто интерфейсами JDBC
- Создание и удаление соединений с БД — это ресурсоемкие операции
  - При создании логического подключения выполняется выборка свободного физического подключения из пула или пул расширяется
  - При закрытии логического соединения выполняется возврат физического подключения обратно в пул

Подключение к БД — пул соединений

- Существенное отличие Java EE-приложений от Java SE-приложений:
  - Java SE -> нет пула: подключение создается как можно раньше, а освобождается как можно позже, и удерживается приложением в течение всего сеанса работы пользователя с приложением
  - Java EE -> есть пул: подключение создается как можно позже (именно тогда, когда оно действительно необходимо), а освобождается как можно раньше (сразу после того, как мы его использовали)

Создание запроса

- Виды запросов:



- В Java EE-приложениях рекомендуется использовать подготовленные запросы, даже если они не содержат параметров
  - > Подготовленный SQL-запрос заранее компилируется => выполняется быстрее неподготовленного
  - > JDBC-драйвер или СУБД кэширует подготовленные запросы
  - > Преобразование входных данных (обработка строк и дат)

Выполнение запроса

- Параметры подготовленного запроса обозначаются **?**
- Перед выполнением запроса необходимо установить значения параметров с помощью set-методов
- Если запрос возвращает число измененных записей (INSERT, UPDATE, DELETE), то

```
> stmt.executeUpdate()
```

- Если запрос возвращает набор результатов (SELECT), то

```
> stmt.executeQuery()
```

Выполнение запроса — пример

```
try {
    PreparedStatement stmt = conn.prepareStatement(
        "SELECT * FROM book WHERE title LIKE ?");
    stmt.setString(1, "%Java%");
    ResultSet result = stmt.executeQuery();
} catch (SQLException ex) {
    // обработка ошибок
}
```

Обработка результатов запроса

- Интерфейс java.sql.ResultSet представляет таблицу результатов SELECT-запроса
- Характеристики таблицы результатов (определяются при создании запроса):
  - Тип — определяет возможность последовательного просмотра таблицы результатов и ее чувствительность к изменениям, параллельно вносимым в БД
  - Возможность обновления курсора
  - Время жизни таблицы результатов

Обработка результатов запроса

- Доступ к таблице осуществляется на основе указателя текущей записи – курсора
  - Изначально курсор располагается перед первой записью
  - Для любого типа таблицы поддерживается метод next(), переводящий курсор на следующую запись
- Для доступа к столбцам текущей записи - get-методы
  - По индексу столбца, начиная с 1
  - По имени столбца

Обработка результатов запроса

```
try {
    ResultSet result = conn.createStatement().executeQuery(
        "SELECT title, author, price FROM book");
    while (result.next()) {
        String title = result.getString(1);
        String author = result.getString("author");
        float price = result.getFloat(3);
        System.out.println(author+" "+title+" - "+price);
    }
} catch (SQLException ex) {
    // обработка ошибок
}
```

Соответствие типов SQL и Java

Тип данных SQL	Тип объекта Java
CHAR, VARCHAR	java.lang.String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT, BOOLEAN	java.lang.Boolean
TINYINT	java.lang.Byte
SMALLINT	java.lang.Short
INTEGER	java.lang.Integer
BIGINT	java.lang.Long
REAL	java.lang.Float
FLOAT, DOUBLE	java.lang.Double
BINARY, VARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CLOB	java.sql.Clob
BLOB	java.sql.Blob
ARRAY	java.sql.Array
STRUCT	java.sql.Struct
DATALINK	java.net.URL

Обработка мета-данных

- С помощью интерфейсов DatabaseMetaData и ResultSetMetaData можно получать сведения о базе данных и результатах запроса

```
Statement statement = connection.createStatement();
String query = "SELECT * FROM " + tableName;
ResultSet resultSet = statement.executeQuery(query);
out.println("<TABLE BORDER=1>");
ResultSetMetaData resultsMetaData = resultSet.getMetaData();
int columnCount = resultsMetaData.getColumnCount();
out.println("<TR>");
for(int i=1; i<columnCount+1; i++)
    out.print("<TH>" + resultsMetaData.getColumnName(i) + "</TH>");
out.println("</TR>");
while(resultSet.next()) {
    out.println("<TR>");
    for(int i=1; i<columnCount+1; i++)
        out.print("<TD>" + resultSet.getString(i) + "</TD>");
    out.println("</TR>");
}
```

Освобождение ресурсов

- Соединения с базами данных, запросы и таблицы результатов — это одновременно JDBC-ресурсы и ресурсы СУБД
  - Для эффективной работы СУБД в многопользовательском режиме рекомендуется своевременно освобождать занятые ресурсы
- Для явного освобождения ресурсов в интерфейсах Connection, Statement и ResultSet есть методы close()
  - При закрытии запроса автоматически закрываются связанные с ним таблицы результатов
  - При закрытии соединения автоматически закрываются связанные с ним запросы

Освобождение ресурсов

- В Java EE-приложениях соединение с БД нужно освобождать сразу после окончания операций с БД

```
try {
    Connection conn = ds.getConnection();
    PreparedStatement stmt = conn.prepareStatement(...);
    ResultSet result = stmt.executeQuery();
    // операции с таблицей результатов
    ...
    stmt.close(); // также закрывается таблица результатов
} catch (SQLException ex) {
    // обработка ошибок
} finally {
    try {
        conn.close();
    } catch (SQLException ex) {
        // обработка ошибок при закрытии соединения
    }
}
```