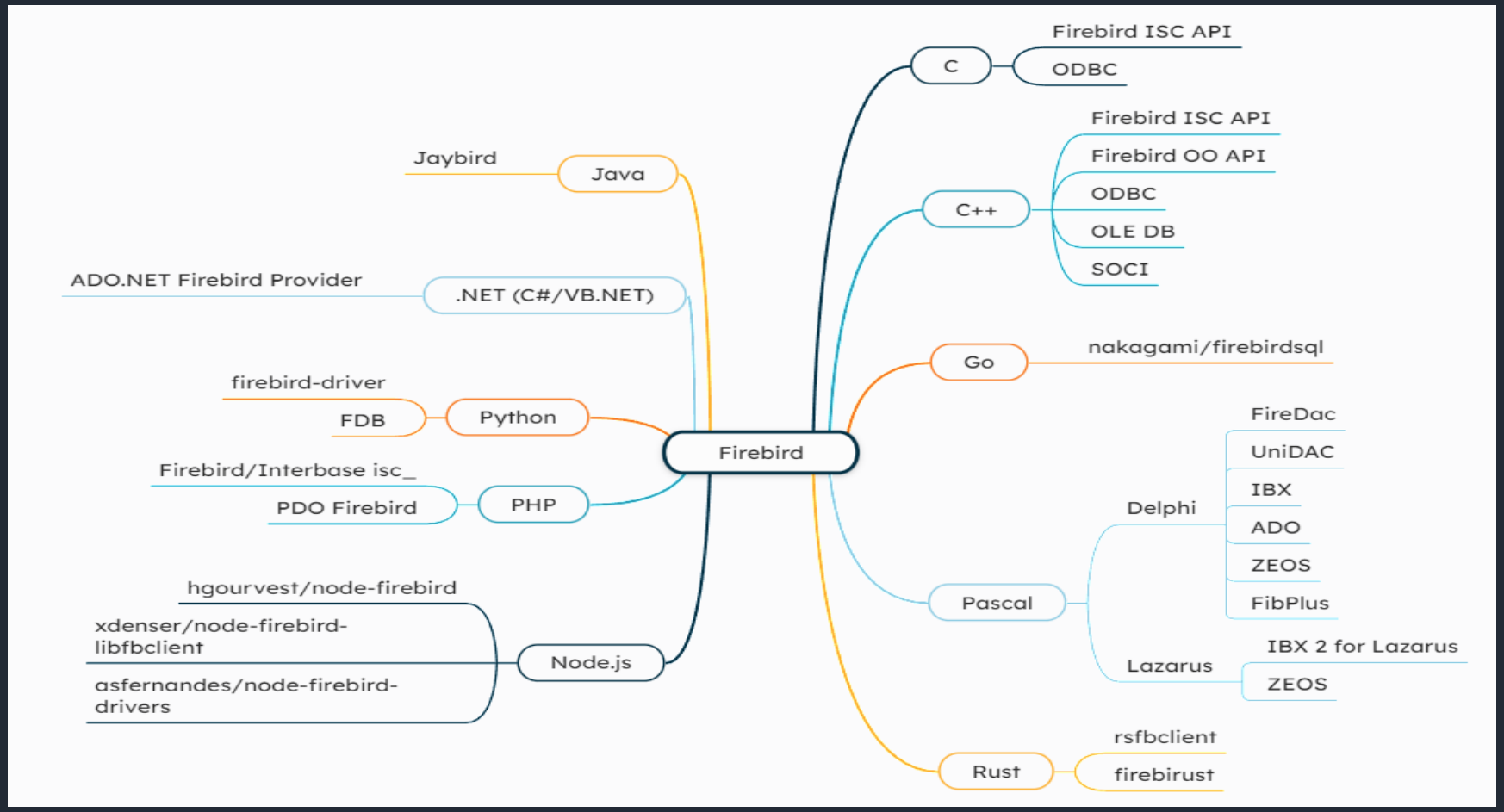
Драйвера, фреймворки и компоненты доступа для Firebird

Денис Симонов
Системный архитектор.
Автор русскоязычной документации Firebird

FIREBIRD CONF 2024 | РЕДСОФТ

Драйвера в различных языках программирования



Классификация компонентов доступа/драйверов

- По способу реализации
  - использующие fbclient
    - Legacy isc API
    - OO API
  - самостоятельная реализация сетевого протокола
- Универсальность
  - Универсальные компоненты доступа
    - Одна транзакция на соединение или множество
    - События firebird
    - Работа с Service API
  - Специализированные
- Стандартизованные интерфейсы:
  - ODBC, OLE DB (IBProvider)

Legacy ISC API

- Plain функции на языке C
- Функции начинаются с префикса isc_ или fb_
- Возвращают статус ISC_STATUS
- Сгруппированы по хендлам
- Для входных и выходных сообщений используется структура XSQLDA
- Размеры имён метаданных ограничена 31 байтом
- Нет возможности использовать Batch API

Объектно ориентированный API

- Доступен начиная с Firebird 3.0
- Набор интерфейсов (абстрактных классов)
- Методы возвращают указатели или другие значения
- Ошибки обрабатываются через IStatus
- Для входных и выходных сообщений используется IMetadata + плоский буфер
- Размеры имён метаданных ограничены интерфейсом (в движке 63 символа UTF-8 для FB 4.0+)
- Batch API (Firebird 4.0 и выше)
- Есть функции для перехода от isc API в OO и обратно
- IUtil содержит дополнительные функции для работы с новыми типами
- CLOOP инструмент для генерирования API для языков C, C++, Pascal

ODBC

- Стандартизованный интерфейс в Window, Linux, Mac OS
- Последняя стабильная версия 2.0.5.156 (2017 год)
- В разработке 3.0.0 (только Windows)
- Требует fbclient Firebird 3.0 и выше
- Контрибьютор Юрий Николаев
- https://github.com/FirebirdSQL/firebird-odbc-driver/wiki

C++

- Firebird API
- ODBC/OLEDB
- SOCI (https://github.com/SOCI/soci)
- SQLAPI++ (платная)
- IBPP (https://sourceforge.net/projects/ibpp/)
- QIBASE (Qt)
- C++ Builder (почти всё что есть в Delphi)

Delphi компоненты

Для них есть более или менее унифицированный интерфейс.

Классы компонентов в основном наследуются от

- TCustomConnection
- TDataSet

TDataSet – абстракция над курсором (набор данных)
Может быть однонаправленный или двунаправленный.

Доступ к столбцам курсора осуществляются через TField

Delphi компоненты универсальные

- FireDac
- UniDac (IBDac)
- DBX (db Express)
- Zeos
- ADO (Firebird ODBC, OLE DB)
- BDE (не поддерживается)

FireDac (AnyDac)

- Универсальные компоненты доступа
- Доступны начиная Delphi XE3 (с редакции Enterprise к удалённым БД, до - локальным)
- СУБД специализируется через FireDac Links (TFDPhysFBDriverLink)
- Использует fbclient. Путь можно указать в TFDPhysFBDriverLink
- Поддерживают все версии Firebird, в том числе 5.0 начиная с Delphi 12.1
- TFDQuery, TFDTable и TFDStoredProc унаследованы от TFDRdbmsDataSet.
- Наборы данных для Firebird/Interbase поддерживают раздельные транзакции для чтения и записи
- Поддерживают события с помощью FDEventAlerter
- Isc API + OO API (fbclient 3.0+)
- Есть компоненты для Service API (backup, restore, nbackup, trace, ...)

DB Express

- Универсальные компоненты доступа
- Наборы данных (TSQLDataSet, TSQLQuery, TSQLTable) однонаправленные
- Для двунаправленных наборов данных надо использовать TClientDataSet и/или DataSetProvider
- BOOLEAN не поддерживается
- Можно указать путь до клиентской библиотеки через VendorLib
- Управление транзакциями ограничено
- События не поддерживаются
- Isc API
- Service API не поддерживается

ZEOS

- Универсальные компоненты доступа
- СУБД специализируется через свойство protocol
- Путь к fbclient можно указать в свойстве LibraryLocation
- BOOLEAN поддерживается
- Для наборов данных ZReadonlyQuery, ZQuery, Ztable, ZStoredProc поддерживаются отдельные транзакции для чтения и записи (начиная с 8 версии)
- Компоненты транзакций (начиная с 8-ой версии)
- Поддерживаются события
- Используется OO API, если поддерживается fbclient (3.0+), в противном случае isc API
- Service API не поддерживается

Delphi компоненты специализированные

- IBX (Interbase Express)
- IB Objects
- FIBPlus (не поддерживается)
- UIB (не поддерживается)

IBX (Interbase Express)

- Компоненты доступа для Interbase
- Работаю с Firebird с ограничениями (API Firebird 1.0/Interbase 6.0)
- Тип BOOLEAN не совместим с Firebird
- Требует gds32.dll (причём с подменённой версией, для поддержки этих компонентов возвращается версия «interbase» 6.3.x)
- Наборы записей TIBTable, TIBQuery, TIBDataset имеют одну транзакцию для чтения и записи
- Поддерживаются события (TIBEvent)
- Isc API
- Поддерживается Service API (ограниченно)

FibPlus

- Компоненты доступа для Interbase и Firebird
- Продажи прекращены, автор не отвечает
- Совместимо с Firebird 3.0
- Поддерживает родной для Firebird тип BOOLEAN
- Можно указать путь до клиентской библиотеки
- Наборы записей TFibTable, TFibQuery, TFibDataset имеют раздельные транзакции для чтения и записи
- Поддерживаются события
- Isc API
- Поддерживается Service API (которые были до Firebird 3.0)

Lazarus компоненты

- IBX 2 (for Lazarus)
- SQL DB
- ZEOS

IBX 2 (for Lazarus)

- Компоненты доступа для Firebird
- Реализована низкоуровневая обёртка над API

(isc для fbclient <= 2.5, OO API для fbclient >= 3.0)

- Типы BOOLEAN, DECFLOAT, INT128, NUMERIC(38, x), TIME ZONE
- Требует fbclient в дизайнтайме. В рантайме можно указывать путь до библиотеки с помощью свойства FirebirdLibraryPathName
- Наборы записей TIBTable, TIBQuery, TIBDataset имеют одну транзакцию для чтения и записи
- Поддерживаются события (TIBEvent)
- TIBSQL умеет использовать Batch API
- Поддерживается Service API

SQL DB (Lazarus)

- Универсальные компоненты доступа
- Соединение через TSQLConnector с указания драйвера, для Interbase/Firebird можно сразу установить компонент IBConnection
- Поддерживает тип BOOLEAN из Interbase/Firebird
- Путь к клиентской библиотеке можно указать в компоненте TSQLDBLibraryLoader
- Набор записей TSQLQuery использует одну транзакцию для чтения и записи
- Поддерживаются события (TFBEventMonitor)
- Поддерживается Service API (TFBAdmin)
- Isc API

Java

- Драйверы
  - Jaybird
- Библиотека
  - jOOQ
- ORM
  - Hibernate

Jaybird

- Грант Firebird Foundation
- Работает через универсальный интерфейс JDBC (Type 4)
- Собственная реализация сетевого протокола, для embedded взаимодействует с fbclient (JNA - Java Native Access)
- Поддержка Firebird 5.0
- Только одна транзакция на соединение
- Работает с Service API
- Работает с событиями Firebird

Jaybird

```
Class.forName("org.firebirdsql.jdbc.FBDriver");
Connection connection = DriverManager.getConnection(
    "jdbc:firebird://localhost:3050/c:/db/employee.fdb", "SYSDBA", "masterkey");

try (PreparedStatement stmt1 = connection.prepareStatement(
          "SELECT userId FROM users WHERE " +
          "lastName = ? AND firstName = ?")) {
  stmt1.setString(1, "Doe");
  stmt1.setString(2, "Joe");
  try (ResultSet rs = stmt1.executeQuery()) {

    if (rs.next()) {
      int userId = rs.getInt(1);

  }
}
```

jOOQ

- Это не ORM, а надстройка которая позволяет конструировать запросы и получать результат
- https://www.jooq.org
- Позволяет в ОО виде писать сложные запросы в том числе MERGE, использовать оконные функции, вызов хранимых процедур, DDL и многое другое
- Поддерживает сборку запросов по частям
- Поддерживает подстановку литералов, именованные и неименованные параметры
- Статья с примером https://habr.com/ru/articles/331098/

jOOQ

```
Result<Record> result =
dsl.select()
   .from(AUTHOR)
   .join(BOOK).on(AUTHOR.ID.equal(BOOK.AUTHOR_ID))
   .where(AUTHOR.YEAR_OF_BIRTH.greaterThan(1920)
   .and(AUTHOR.FIRST_NAME.equal("Paulo")))
   .orderBy(BOOK.TITLE)
   .fetch();
```

```
SELECT *
  FROM author
  JOIN book ON author.id = book.author_id
 WHERE author.year_of_birth > 1920
   AND author.first_name = 'Paulo'
 ORDER BY book.title
```

jOOQ

```
dsl.update(AUTHOR)
   .set(AUTHOR.FIRST_NAME, "Hermann")
   .set(AUTHOR.LAST_NAME, "Hesse")
   .where(AUTHOR.ID.equal(3))
   .execute();
```

```
UPDATE AUTHOR
   SET FIRST_NAME = 'Hermann',
       LAST_NAME = 'Hesse'
 WHERE ID = 3;
```

Hibernate

- ORM реализующая спецификацию JPA (Java Persistence API)
- Таблицы отображаются на объекты классов (сущности). Классы необходимо размечать специальными аннотациями
- Поддерживаются связи между сущностями «один ко многим», «многие ко многим»
- Поддержка миграций (обновление схемы БД по объектной модели)
- Может оперировать сущностями напрямую или использовать SQL-подобный язык HQL (Hibernate Query Language)
- org.hibernate.dialect.FirebirdDialect

Hibernate

```java
@Entity
public class Author {
    @Id
    private Integer id;
    private String firstName;
    private String lastName;

    @ManyToMany
    private List<Book> books;
}

@Entity
public class Book {
    @Id
    private Integer id;
    private String title;
    private String isbn;

    @ManyToOne
    private Publisher publisher;

    @ManyToMany
    private List<Author> authors;
}
```

Hibernate

```
@Autowired
private SessionFactory sessionFactory

public void getTasks(Long userId) {
    ...
    Session session = sessionFactory.openSession();
    User user = (User) session.load(User.class, userId);
    Session session = sessionFactory.openSession();
    List<Task> tasksList = user.getTasks();
    ...
 }

 public Long saveUser(User user) {
   Session session = sessionFactory.openSession();
   session.openTransaction();
   Task task = new Task();
   task.setName("Задача 1");
   task.setTaskDate(new Date());
   task.setUser(user);
   session.saveOrUpdate(task);
   session.flush();
   session.getTransaction().commit();
   return task.getTaskId();
}
```

Hibernate (HQL)

HQL: from User

Преобразуется в

SQL: **SELECT** * **FROM** USER

SQL и HQL: select distinct u.name from User u where u.created > '2020-01-01'

```
public List<String> getUserNames() {
    try (Session session = sessionFactory.openSession()) {
            String hql = "select distinct u.name from User u where u.created > '2020-01-01'";
            Query<String> query = session.createQuery(hql , String.class);
            return query.list();
    }
}
```

.NET

- Драйверы
  - Firebird ADO.NET Data Provider
- Фреймворки
  - .NET Core
- ORM
  - Entity Framework
  - NHibernate

ADO.NET Data provider

- Грант Firebird Foundation
- Реализует интерфейс взаимодействия с БД – ADO.NET
- Собственная реализация сетевого протокола, есть вариант использования fbclient
- Firebird 5.0 (DECLFLOAT, INT128, BatchAPI)
- Одна транзакция на соединение
- Поддерживает Service API
- Поддерживает события

ADO.NET Provider

```
FbConnectionStringBuilder connStrBuilder = new FbConnectionStringBuilder();
connStrBuilder.Database = "localhost/3051:horses";
connStrBuilder.UserID = "SYSDBA";
connStrBuilder.Password = "masterkey";
connStrBuilder.Charset = "WIN1251";
using (FbConnection conn = new FbConnection(connStrBuilder.ToString())) {
    string sql = "SELECT CODE_HORSE, REMARK FROM HORSE FETCH FIRST 10 ROWS ONLY";
    conn.Open();
    FbCommand selectCommand = new FbCommand(sql, conn);
    FbDataReader reader = selectCommand.ExecuteReader();

    if (reader.HasRows) {
        while (reader.Read()) {
           Console.WriteLine("{0}\t{1}", reader["CODE_HORSE"],
                             reader["REMARK"]);
        }
    }
    reader.Close();
}
```

Entity Framework

- ORM. Поддерживает генерация сущностей на основе БД (Database First) или генерация базы данных по сущностям (Code First)
- Таблицы отображаются на объекты классов (сущности). Классы необходимо размечать специальными атрибутами
- Поддерживаются связи между сущностями «один ко многим», «многие ко многим»
- Поддержка миграций (обновление схемы БД по объектной модели)
- Может оперировать сущностями напрямую или использовать SQL-подобный язык LINQ
- Статья https://habr.com/ru/articles/278405/

Entity Framework

```
[Table("Firebird.INVOICE")]
public partial class INVOICE {
    public INVOICE() {
        INVOICE_LINES = new HashSet<INVOICE_LINE>();
    }

    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.None)]
    public int INVOICE_ID { get; set; } …

    public virtual CUSTOMER CUSTOMER { get; set; }

    public virtual ICollection<INVOICE_LINE> INVOICE_LINES { get; set; }
}
…
public partial class DbModel : DbContext {
…
protected override void OnModelCreating(DbModelBuilder modelBuilder) {
…
    modelBuilder.Entity<INVOICE>()
                .HasMany(e => e.INVOICE_LINES)
                .WithRequired(e => e.INVOICE)
                .WillCascadeOnDelete(false);
}
```

Entity Framework

```
var dbContext = AppVariables.getDbContext();
// создание нового экземпляра сущности
var customer = (CUSTOMER)bindingSource.AddNew();
customer.CUSTOMER_ID = dbContext.NextValueFor("GEN_CUSTOMER_ID");
// добавляем нового заказчика
dbContext.CUSTOMERS.Add(customer);
// пытаемся сохранить изменения
dbContext.SaveChanges();
// и обновить текущую запись
dbContext.Refresh(RefreshMode.StoreWins, customer);
```

Entity Framework + LINQ

```
// запрос на LINQ преобразуется в SQL
var invoices =
    from invoice in dbContext.INVOICES
    where (invoice.INVOICE_DATE >= AppVariables.StartDate) &&
          (invoice.INVOICE_DATE <= AppVariables.FinishDate)
    orderby invoice.INVOICE_DATE descending
    select new InvoiceView
    {
        Id = invoice.INVOICE_ID,
        Cusomer_Id = invoice.CUSTOMER_ID,
        Customer = invoice.CUSTOMER.NAME,
        Date = invoice.INVOICE_DATE,
        Amount = invoice.TOTAL_SALE,
        Paid = (invoice.PAID == 1) ? "Yes" : "No"
    };

masterBinding.DataSource = invoices.ToBindingList();
```

Python

- Драйверы
  - firebird-driver
  - FDB
  - KInterbasDB (устаревший)
- Фреймворки
  - Django
  - Flask
  - Tornado
- ORM
  - SQLAlchemy
  - Peewee

firebird-driver

- Грант Firebird Foundation
- Установка pip install firebird-driver
- Требует Python 3.8+
- Использует fbclient (Firebird 3.0 и выше)
- Использует OO API
- Реализует Python Database API 2.0

fdb

- Установка pip install fdb
- Требует Python 2.7+ или 3.4+
- Использует fbclient (Firebird 2.0 и выше)
- Использует isc API
- Реализует Python Database API 2.0

firebird-driver

```
from firebird.driver import connect
from firebird.driver import driver_config

driver_config.fb_client_library.set_value("c:\\firebird\\5.0\\fbclient.dll")

def print_countries():
    with connect('localhost/3055:employee', user='SYSDBA', password='masterkey') as con:
        cur = con.cursor()
        SELECT = "select country, currency from country"

        cur.execute(SELECT)
        for (country, currency) in cur:
            print(f"{country} uses {currency} as currency.")

if __name__ == '__main__':
    print_countries()
```

SQLAlchemy

- Установка pip install sqlalchemy-firebird
- Может работать с драйвером fdb или firebird-driver
- Может создавать структуру БД по модели
- Поддерживаются связи между сущностями «один ко многим», «многие ко многим»

SQLAlchemy

```
from sqlalchemy import Column, Integer
from sqlalchemy.orm import sessionmaker, declarative_base
from sqlalchemy_firebird.types import *
import sqlalchemy as sa


# создаем базовый класс для моделей
Base = declarative_base()


# создаем модель, объекты которой будут храниться в бд
class Employee(Base):
    __tablename__ = "EMPLOYEE"

    id = Column(Integer, primary_key=True, name="emp_no")
    first_name = Column(FBVARCHAR(50), nullable=False)
    last_name = Column(FBVARCHAR(50))
```

SQLAlchemy

```
# firebird+firebird://sysdba:masterkey@localhost:3055/employee
dburl = sa.engine.url.URL('firebird+firebird', username='sysdba', password='masterkey',
                          database='employee', host='localhost', port=3055,
                          query={'fb_client_library': "c:\\firebird\\5.0\\fbclient.dll",
                                 'charset': 'UTF-8'})
engine = sa.create_engine(dburl)

# create tables
# metadata.create_all(engine)

session = sessionmaker(autoflush=False, bind=engine)
db = session()
# получение всех объектов emp_no > 50
employees = db.query(Employee).filter(Employee.id > 50).all()
for p in employees:
    print(f"{p.id} {p.first_name} ({p.last_name})")

db.commit()
db.close()
engine.dispose()
```

PHP

- Драйвера:
  - Ibase_ функции
  - PDO firebird
- Фреймворки (ORM):
  - Laravel (Eloquent ORM)
  - Doctrine ORM
  - Yii Framework

PHP Interbase/Firebird

- Набор функций начинающихся с префикса ibase_ или fbird_
- Deprecated, а потому перенесён в PECL
- Необходимо вручную следить за ресурсами (закрывать соединение, курсор, транзакцию)
- Есть возможность работать с расширениями Service API
- Необходим проверять результаты функций, и вызывать ibase_errmsg, ibase_errcode для получения информации об ошибке
- Даёт возможность задавать параметры транзакции
- Работа с параметрами запросов только по номеру (нет именованных)
- Не используется во фреймворках

PDO Firebird

- Универсальный объектно-ориентированный интерфейс для всех СУБД
- Параметры запросы именованные и неименованные
- Не даёт задавать параметры транзакций, но можно стартовать транзакцию запросом SET TRANSACTION
- Для обработчики ошибок можно использовать исключения
- Используется во фреймворках

Laravel (Eloquent ORM)

- Статья на хабре https://habr.com/ru/articles/317458/
- Реализует паттерн Active Record
- https://github.com/jacquestvanzuydam/laravel-firebird

```
DB::transaction(function () {
  // Создаём новую позицию в счёт фактуре
  $line = new App\InvoiceLine();
  $line->CUSTOMER_ID = 45;
  $line->PRODUCT_ID = 342;
  $line->QUANTITY = 10;
  $line->COST = 12.45;
  $line->save();

  // добавляем сумму позиции по строке к сумме накладной
  $invoice = App\Invoice::find($line->CUSTOMER_ID);
  $invoice->INVOICE_SUM += $line->SUM_PRICE;
  $invoice->save();
});
```

JavaScript (Node.js)

- Драйверы
  - https://github.com/hgourvest/node-firebird
  - https://github.com/xdenser/node-firebird-libfbclient
  - https://github.com/asfernandes/node-firebird-drivers

https://github.com/xdenser/node-firebird-libfbclient

- Использует fbclient
- Устанавливается командой npm install firebird
- Асинхронный и синхронный интерфейс
- Поддерживается Service API
- Поддерживается 2.5 и 3.0

https://github.com/xdenser/node-firebird-libfbclient

```
var fb  = require("./firebird");
sys = require("sys");
var con = fb.createConnection();
con.connectSync('test.fdb','sysdba','masterkey','');
con.querySync("insert into test (id,name) values (5, 'new one')");
var res = con.querySync("select * from test");
con.commitSync();
var rows = res.fetchSync("all",true);
console.log(sys.inspect(rows));
```

https://github.com/hgourvest/node-firebird

- Собственная реализация сетевого протокола
- Устанавливается командой npm install node-firebird
- Поддерживается пул соединений
- Асинхронный интерфейс
- Поддерживается Service API
- Поддерживается только сетевой протокол 2.5, для более новый менять конфигурацию

```
AuthServer = Srp256, Srp, Legacy_Auth
WireCrypt = Disabled
UserManager = Legacy_UserManager
```

https://github.com/hgourvest/node-firebird

```
// 5 = the number is count of opened sockets
var pool = Firebird.pool(5, options);

// Get a free pool
pool.get(function(err, db) {

    if (err)
        throw err;

    db.query('SELECT * FROM TABLE', function(err, result) {
        // IMPORTANT: release the pool connection
        db.detach();
    });
});

// Destroy pool
pool.destroy();
```

https://github.com/asfernandes/node-firebird-drivers

- Использование OO API fbclient
- Написан на TypeScript + (CLOOP)
- Поддерживается Firebird 3.0, 4.0 и 5.0
- Асинхронные методы возвращают Promise
- Устанавливается командой

```
npm install node-firebird-driver
npm install node-firebird-native-api
Npm install node-firebird-driver-native
```

https://github.com/asfernandes/node-firebird-drivers

```
async function test() {
    const client = createNativeClient(getDefaultLibraryFilename());
    const attachment = await client.connect('localhost:employee', {
        username: 'sysdba', password: 'masterkey' });
    const transaction = await attachment.startTransaction();
    const resultSet = await attachment.executeQuery(transaction,
        'select n, d from t1 where n <= ?', [2]);
    const rows = await resultSet.fetch();
    for (const columns of rows)
        console.log('n: ' + columns[0] + ', d: ' + columns[1]);
    await resultSet.close();
    await transaction.commit();
    await client.dispose();
}
test().then(() => console.log('Finish...'));
```

Golang

- Универсальный доступ к БД через пакет database/sql
- Драйвер для Firebird можно получить здесь https://github.com/nakagami/firebirdsql
- Установка одной командой

```
go get github.com/nakagami/firebirdsql
```

- GORM

go get github.com/flylink888/gorm-firebird

Пример на Go

```
package main

import (
  "database/sql"
   "fmt"
    _ "github.com/nakagami/firebirdsql"
)

type Employee struct {
  emp_no     int
  first_name string
  last_name  string
}
```

Пример на Go (продолжение)

```
func main() {
  conn, err := sql.Open("firebirdsql", "SYSDBA:masterkey@localhost:3055/employee")
  if err != nil {
    panic(err)
  }
  defer conn.Close()
  tra, err := conn.Begin()
  if err != nil {
    panic(err)
  }
  defer tra.Rollback()
```

Пример на Go (продолжение)

```
   rows, err := tra.Query("SELECT EMP_NO, FIRST_NAME, LAST_NAME FROM employee WHERE EMP_NO > ?",
50)
  if err != nil {
    panic(err)
  }
  defer rows.Close()
  fmt.Println("| EMP_NO    | FIRST_NAME   | LAST_NAME     |")
  fmt.Println("| --------- | ------------ | ------------- |")
  for rows.Next() {
    var emp = Employee{}
    rows.Scan(&emp.emp_no, &emp.first_name, &emp.last_name)
    fmt.Printf("|%10d | %-12s | %-13s |\n", emp.emp_no, emp.first_name, emp.last_name)
  }
  tra.Commit()
}
```

Rust

- Драйвер https://github.com/fernandobatels/rsfbclient (0.24.0)
- Драйвер https://github.com/nakagami/firebirust (0.4.3)
- Оба в зачаточном состоянии, но работать можно
- Устанавливаются автоматически с помощью менеджера пакетов Cargo

Драйвер rsfbclient

- https://github.getafreenode.com/fernandobatels/rsfbclient
- Существуют варианты соединения с базой данных через fbclient (динамической линковкой или динамической загрузкой) и с собственной реализацией сетевого протокола
- Параметры транзакции могут быть установлены для транзакции по умолчанию для соединения или в отдельном объекте транзакции
- Для использования в Cargo.toml указать

```
[dependencies]
rsfbclient = "0.24.0"
```

Пример на Rust

```rust
#![allow(unused_variables, unused_mut)]

use rsfbclient::{prelude::*, FbError};

fn main() -> Result<(), FbError> {
    let mut conn_builder = rsfbclient::builder_native()
        .with_dyn_link()
        .with_remote()
        .host("localhost")
        .db_name("employee")
        .user("SYSDBA")
        .pass("masterkey")
        .clone();

    let mut conn = conn_builder
        .with_transaction(|tr|
           tr.read_only().with_read_commited(TrRecordVersion::RecordVersion))
        .connect()?;
```

Пример на Rust (продолжение)

```
 let rows = conn.query_iter("select EMP_NO, FIRST_NAME, LAST_NAME from employee", ())?;

 println!("| EMP_NO | FIRST_NAME   | LAST_NAME     |");
 println!("| ------ | ------------ | ------------- |");
 for row in rows {
        let (emp_no, first_name, last_name): (i32, String, String) = row?;

        println!("| {:^6} | {:12} | {:12} |", emp_no, first_name, last_name);
 }

 Ok(())
}
```

Драйвер firebirust

- https://github.com/nakagami/firebirust
- Собственная реализация сетевого протокола
- В транзакцию нельзя передавать параметры старта
- Драйвер ещё очень сырой
- Для использования в Cargo.toml указать

```
[dependencies]
firebirust = "0.4.3"
```

Рекомендации

Язык	Драйвер/Компонент	ORM	Библиотека/Фреймворк
C	Legacy ISC API		
C++	OO API		
Delphi	FireDac		
Lazarus	IBX 2 for Lazarus		
Java	Jaybird	Hibernate	jOOQ
.NET (C#)	Firebird ADO.NET Provider	Entity Framework	.NET Core
Python	firebird-driver	sqlalchemy-firebird	
PHP	PDO firebird		
Node.js	asfernandes/node-firebird-drivers		

Вопросы?

Спасибо за внимание!