FIREBIRD CONF 2025

Рустам Ахматов «СОФТ72»

СОФТ72

Опыт миграции действующей линейки продуктов на СУБД Ред База Данных

Кто мы?

СОФТ72

Команда опытных Fullstack разработчиков, использующих в качестве основных языков программирования Python и JavaScript

Мы развиваем два параллельных направления:

- собственная продуктовая линейка
- разработка «под ключ» под задачи заказчиков

ПАК «ДОЗОР»

СОФТ72

Для кого:

- для всех объектов транспортной инфраструктуры, а также мест массового скопления людей, предусматривающих наличие досмотровых комплексов

Почему это важно?

- Система ДОЗОР позволяет «строить» безопасные объекты за счет создания единой централизованной платформы, объединяющей различные типа досмотрового оборудования (от металлодетекторов до интроскопов) и системы видеонаблюдения.
- Наши заказчики получают возможность оперативно и быстро реагировать на триггерные события, проводить ретроспективный анализ, а также собирать накопленные данные для превентивного анализы и принятия грамотных управленческих решений по организации безопасности на объектах

Система аудиозаписи и речевой аналитики «ВОЛНА»

Для кого:

- Медицинские учреждения
- МФЦ
- Объекты обслуживания населения
- Финансовые организации
- Ритейл

ВОЛНА помогает:

- Быстро разбирать инциденты за счет анализа аудиозаписей разговоров в удобном формате (аудио и/или текст).
- Формировать собственную уникальную базу знаний для обучения
- Объективно оценивать и выставлять kpi сотрудникам
- Интегрироваться с имеющимися АИС для оптимизации процесса оказания услуги

Причины миграции действующей линейки продуктов

1. Технические особенности и возможности
2. Удобство использования
3. Безопасность и соответствие стандартам
4. Происхождение и лицензирование

СУБД РЕД База Данных для сокращения пути миграции

Используемые технологии в наших продуктах, трудности при адаптации

- Python 3.x
- ORM – pyDAL
- ORM – SQLAlchemy
- Driver - firebird.py

Эволюция подходов и решений

ERROR 1

Dynamic SQL Error SQL error code = -104 String literal with 72116 bytes exceeds the maximum length of 65535 bytes

Решение заключалось в параметризации запросов, на уровне ORM. Наследовали основной класс, переопределили методы на параметризированные SQL запросы

Python

```python
def _build_value(self, field, value, r_values, query_env=None):
    if field.type == "text":
        r_values.append(value)
        return "?"
    return self.expand(value, field.type, query_env=query_env)
```

Эволюция подходов и решений

Где потом используется ?

В методе *insert, при формировании запроса:*

```python
",".join(self._build_value(f, v, r_values) for f, v in fields)
```

и при вызове:

```python
self.execute(query, *values)
```

где values — это список параметров для плейсхолдера (`?`).

Эволюция подходов и решений

ERROR 2

Dynamic SQL Error SQL error code = -104 String literal with 72116 bytes exceeds the maximum length of 65535 bytes

arm/services/arm_proc/src/database/models.py

```
...  ...  @@ -59,7 +59,7 @@ role: роль
59   59   email: электронная почта
60   60       """
61   61       __tablename__ = 'user'
62        -   uuid = Column(UUID(as_uuid=True), primary_key=True, index=True, default=uuid.uuid4)
     62   +   uuid = Column(String(16), primary_key=True, index=True, default=uuid.uuid4)
63   63       username = Column(String)
64   64       password = Column(String)
65   65       role = Column(Enum(RoleEnum))
```

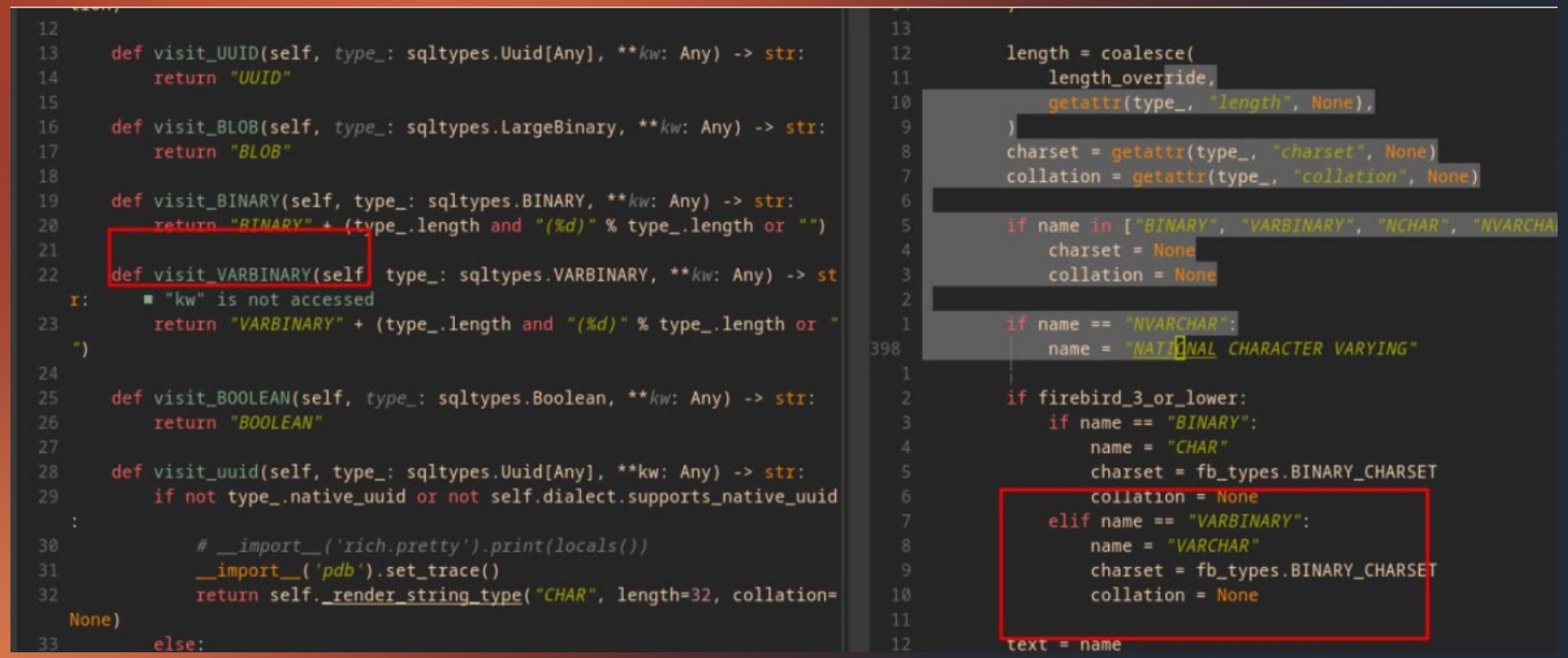
Эволюция подходов и решений

```
85  85    link: ссылка на видеофайл
86  86        """
87  87        __tablename__ = 'screening'
88      -     uuid = Column(UUID(as_uuid=True), primary_key=True, index=True, default=uuid.uuid4)
    88  +     uuid = Column(String(16), primary_key=True, index=True, default=uuid.uuid4)
89  89        datetime = Column(DateTime)
90  90        level = Column(Integer)
91      -     zone = Column(ARRAY(Integer))
    91  +     zone = Column(String(50))
92  92        rad_level = Column(Integer)
93  93        is_synchronize = Column(Boolean)
94  94        photo_link = Column(String)
95      -     video_links = Column(ARRAY(String))
96      -     event_data = Column(JSON, nullable=True)
    95  +     video_links = Column(String(100))
    96  +     event_data = Column(String(1000), nullable=True)
97  97
98      -     detector_uuid = Column(UUID(as_uuid=True), ForeignKey('detector.uuid', ondelete='CASCADE'))
    98  +     detector_uuid = Column(String(16), ForeignKey('detector.uuid', ondelete='CASCADE'))
99  99        detector = relationship('DbDetector', back_populates='screening')
```

Эволюция подходов и решений

ERROR 3

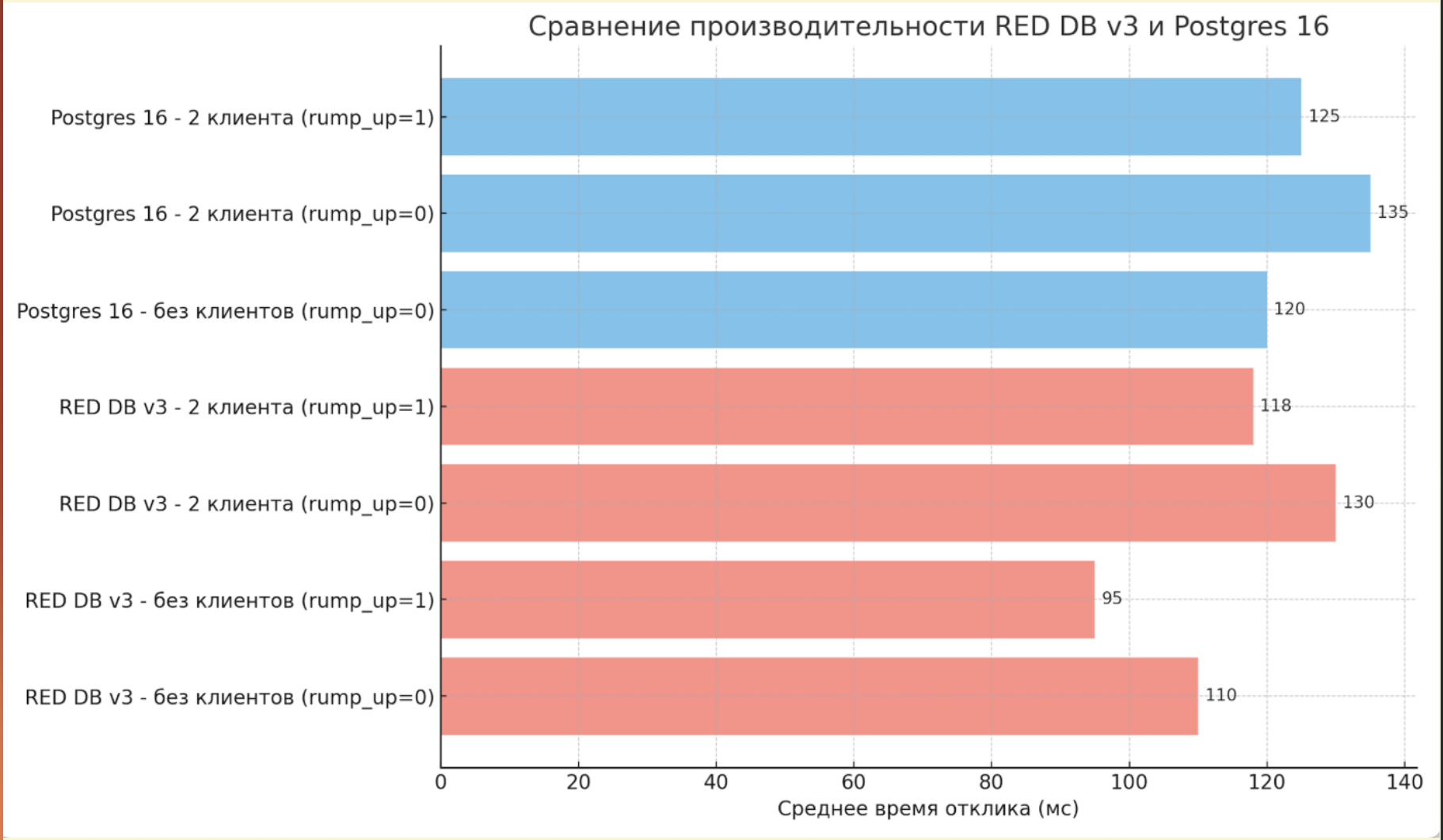
Внесли изменения на уровне ORM SQLAlchemy, наследовали интерфейс и переопределили используемые типы данных



```
    def visit_UUID(self, type_: sqltypes.Uuid[Any], **kw: Any) -> str:
        return "UUID"

    def visit_BLOB(self, type_: sqltypes.LargeBinary, **kw: Any) -> str:
        return "BLOB"

    def visit_BINARY(self, type_: sqltypes.BINARY, **kw: Any) -> str:
        return "BINARY" + (type_.length and "(%d)" % type_.length or "")

    def visit_VARBINARY(self, type_: sqltypes.VARBINARY, **kw: Any) -> st
r:      ■ "kw" is not accessed
        return "VARBINARY" + (type_.length and "(%d)" % type_.length or "
")

    def visit_BOOLEAN(self, type_: sqltypes.Boolean, **kw: Any) -> str:
        return "BOOLEAN"

    def visit_uuid(self, type_: sqltypes.Uuid[Any], **kw: Any) -> str:
        if not type_.native_uuid or not self.dialect.supports_native_uuid
:
            # __import__('rich.pretty').print(locals())
            __import__('pdb').set_trace()
            return self._render_string_type("CHAR", length=32, collation=
None)
        else:
```

```
        length = coalesce(
            length_override,
            getattr(type_, "length", None),
        )
        charset = getattr(type_, "charset", None)
        collation = getattr(type_, "collation", None)

        if name in ["BINARY", "VARBINARY", "NCHAR", "NVARCHA
            charset = None
            collation = None

        if name == "NVARCHAR":
            name = "NATIONAL CHARACTER VARYING"

        if firebird_3_or_lower:
            if name == "BINARY":
                name = "CHAR"
                charset = fb_types.BINARY_CHARSET
                collation = None
            elif name == "VARBINARY":
                name = "VARCHAR"
                charset = fb_types.BINARY_CHARSET
                collation = None

        text = name
```

Эволюция подходов и решений



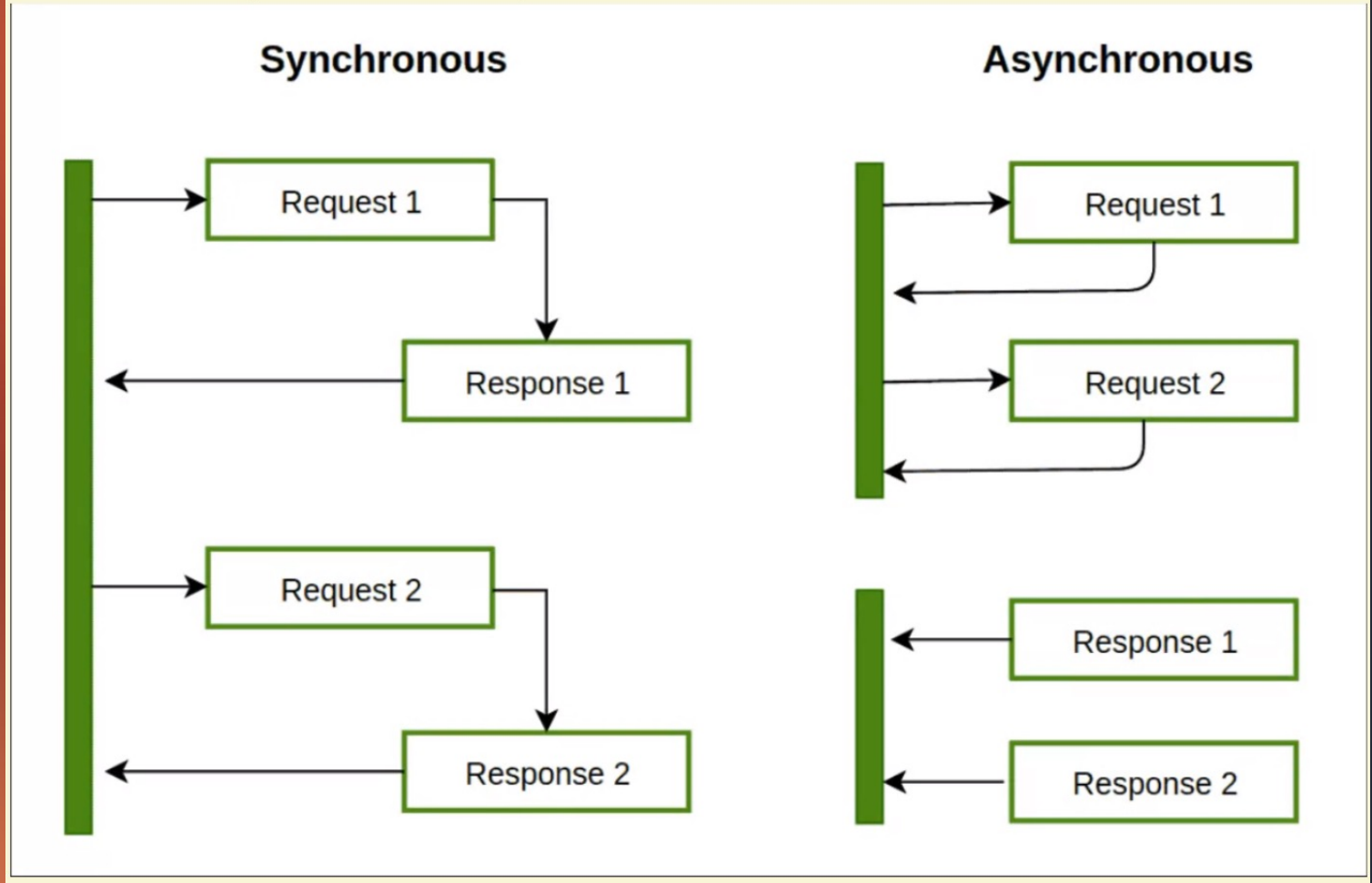
Эволюция подходов и решений

ERROR 4

```
    @pytest.mark.asyncio
    async def test_postgres_usage(database_config: DatabaseConfig):
>       database = AsyncPostgresDatabase(cast(PostgresConfig, database_config))

tests/core/test_database.py:55:
------------------------------------------------------------------------------------------------------------------------
src/library/core/database/postgres.py:38: in __init__
    self.engine: AsyncEngine = create_async_engine(
.venv/lib/python3.12/site-packages/sqlalchemy/ext/asyncio/engine.py:121: in create_async_engine
    return AsyncEngine(sync_engine)
------------------------------------------------------------------------------------------------------------------------

self = <sqlalchemy.ext.asyncio.engine.AsyncEngine object at 0x7f42b92c9b90>, sync_engine = Engine(firebird://dozor:***@127.0.0.1:54320/dozor.fdb)

    def __init__(self, sync_engine: Engine):
        if not sync_engine.dialect.is_async:
>           raise exc.InvalidRequestError(
                "The asyncio extension requires an async driver to be used. "
                f"The loaded {sync_engine.dialect.driver!r} is not async."
            )
E           sqlalchemy.exc.InvalidRequestError: The asyncio extension requires an async driver to be used. The loaded 'firebird-driver' is not async.

.venv/lib/python3.12/site-packages/sqlalchemy/ext/asyncio/engine.py:1033: InvalidRequestError
=========================================================================== short test summary info ============================================================================
FAILED tests/core/test_database.py::test_postgres_connection - sqlalchemy.exc.InvalidRequestError: The asyncio extension requires an async driver to be used. The loaded 'firebird-driver' is n
t async.
FAILED tests/core/test_database.py::test_postgres_session - sqlalchemy.exc.InvalidRequestError: The asyncio extension requires an async driver to be used. The loaded 'firebird-driver' is not
sync.
FAILED tests/core/test_database.py::test_postgres_user - sqlalchemy.exc.InvalidRequestError: The asyncio extension requires an async driver to be used. The loaded 'firebird-driver' is not asy
c.
FAILED tests/core/test_database.py::test_postgres_database - sqlalchemy.exc.InvalidRequestError: The asyncio extension requires an async driver to be used. The loaded 'firebird-driver' is not
async.
FAILED tests/core/test_database.py::test_postgres_usage - sqlalchemy.exc.InvalidRequestError: The asyncio extension requires an async driver to be used. The loaded 'firebird-driver' is not as
nc.
============================================================================ 5 failed in 0.51s =============================================================================
```

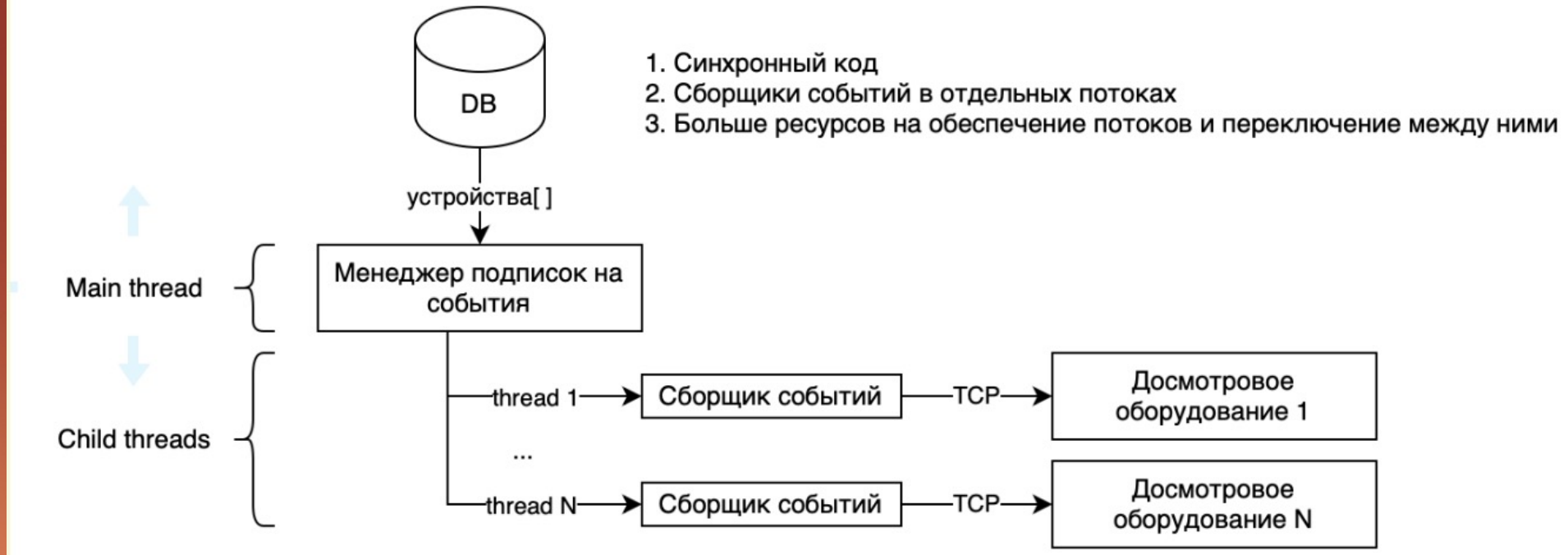
АСИНХРОННАЯ АРХИТЕКТУРА

Различия синхронного и асинхронного подхода



АСИНХРОННАЯ АРХИТЕКТУРА

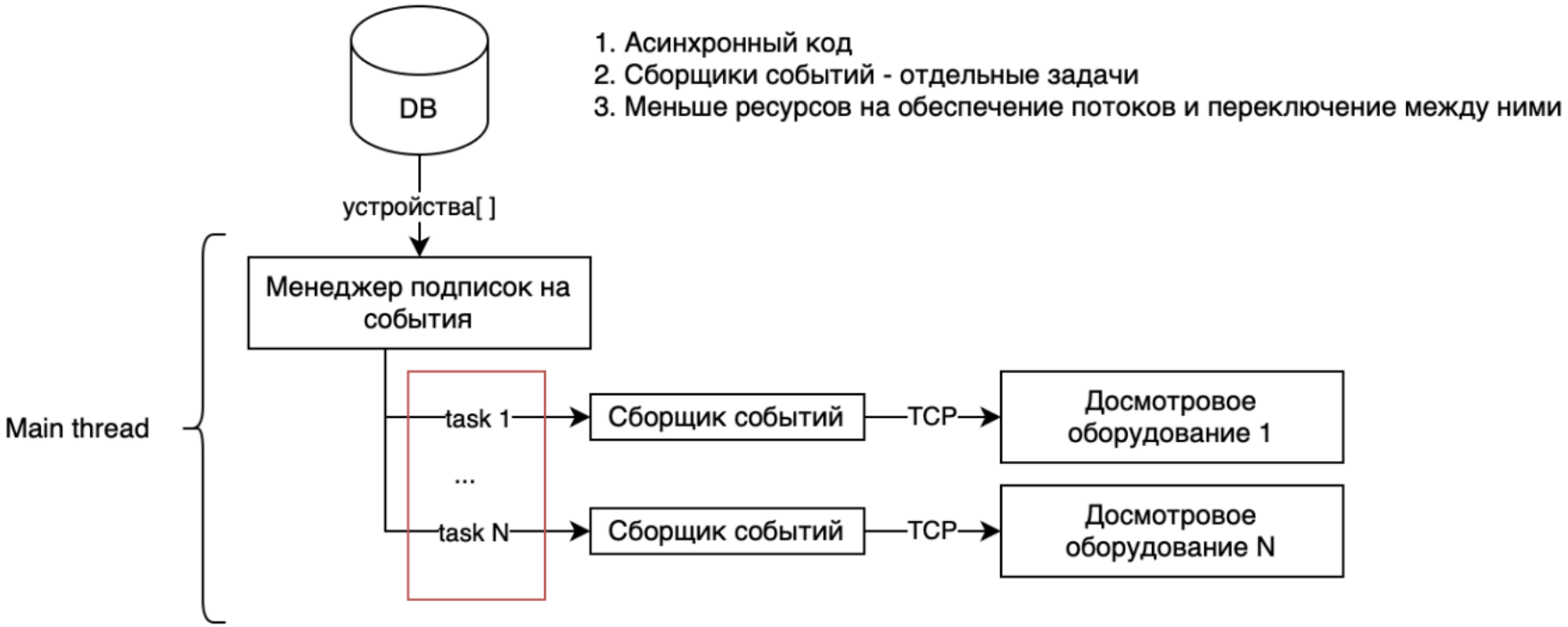
До ассинхронной архитектуры



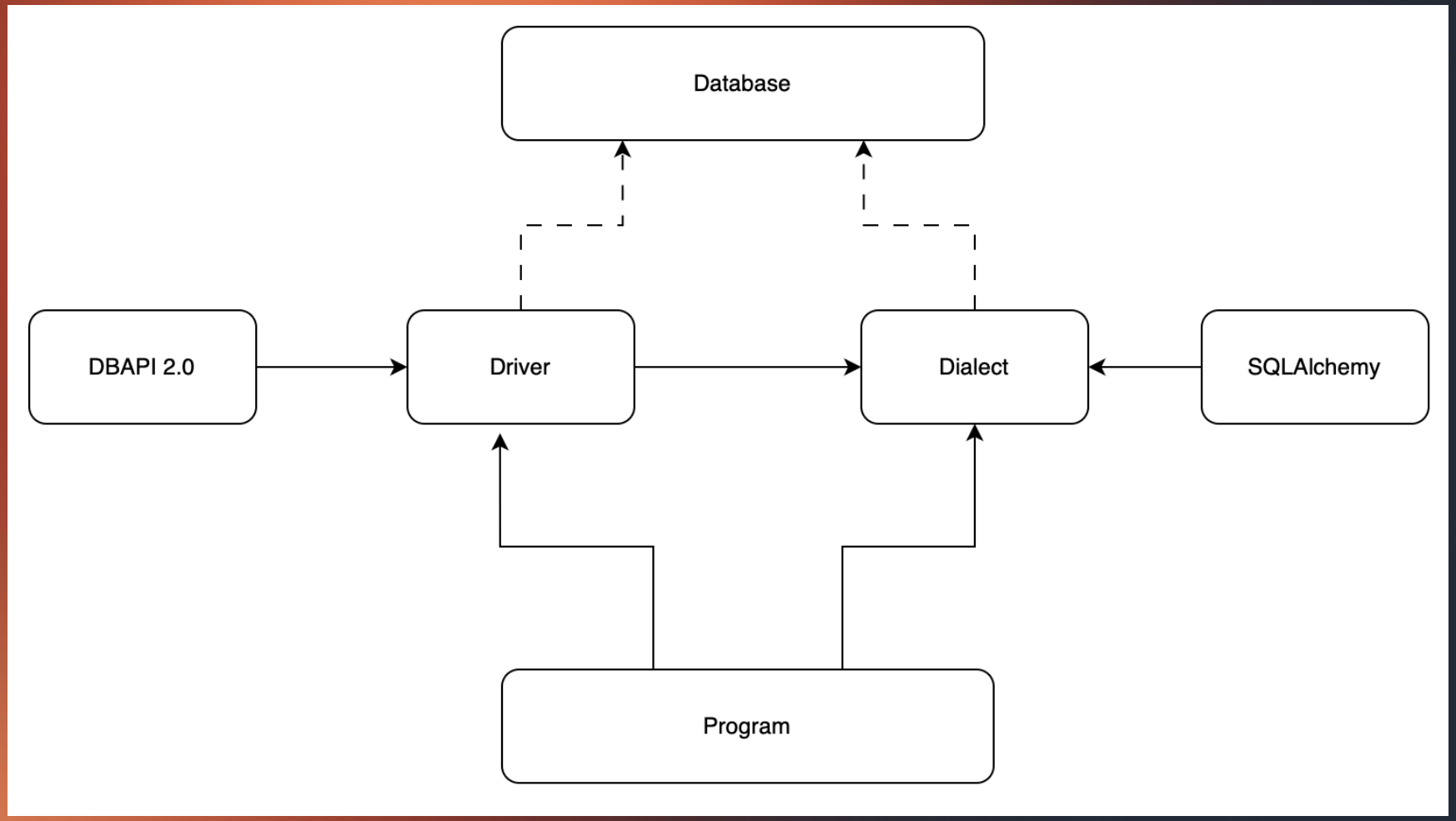
1. Синхронный код
2. Сборщики событий в отдельных потоках
3. Больше ресурсов на обеспечение потоков и переключение между ними

АСИНХРОННАЯ АРХИТЕКТУРА

После



СТРУКТУРА ORM



🧰 Реализация асинхронного диалекта для СУБД РЕД База Данных в SQLAlchemy

Преимущество - использование ORM библиотеки. Мы можем описать табличную модель данных с использованием встроенных типов данных языка программирования Python и в 95% модель пригодна

🔷 Цель:

Создать полноценный диалект для СУБД РЕД БД на базе SQLAlchemy, используя асинхронный драйвер firebirdsql, что обеспечит поддержку СУБД РЕД БД в асинхронных Python-приложениях.

🧰 Реализация асинхронного диалекта для СУБД РЕД База Данных в SQLAlchemy

📦 Что мы используем:

- Драйвер: firebirdsql — поддерживает asyncio и реализует DB API.
- Базовая структура: интерфейсы SQLAlchemy для создания пользовательского диалекта (набор абстрактных классов и API).
- Асинхронный подход: позволяет работать с БД эффективно в рамках event loop без блокировки потока.

🧰 Реализация асинхронного диалекта для СУБД РЕД База Данных в SQLAlchemy

✅ Что уже реализовано:

Словарь зарезервированных слов

- Включены ключевые слова SQL, типы данных и операторы.
- Используется в парсере и компиляторе SQLAlchemy.

Интеграция с драйвером через DB API

- Установлено соединение между ORM-уровнем и драйвером firebirdsql.
- Базовые операции передачи SQL работают (connect, execute, fetch).

🧰 Реализация асинхронного диалекта для СУБД РЕД База Данных в SQLAlchemy

🚧 Что в процессе / осталось реализовать:

Парсер и компилятор SQL (FirebirdCompiler)

- Создан интерфейс для компиляции ORM-запросов в SQL и обратно.
- Требуется дописать обработку всех SQL-конструкций (JOIN, LIMIT, функции и т.п.).

Верификация настроек SQLAlchemy

- Корректное отображение схем, таблиц, колонок и типов (в том числе UUID, BLOB и др.).
- Проверка типов, constraints, autoincrement и транзакционного поведения.

Адаптация блока тестов SQLAlchemy

- Использование встроенной тестовой инфраструктуры SQLAlchemy.
- Покрытие всех CRUD-операций, транзакций, типов данных.
- Добавление недостающих кейсов (например, UUID, sequences, RETURNING).

Эволюция подходов и решений

🎯 Результат:

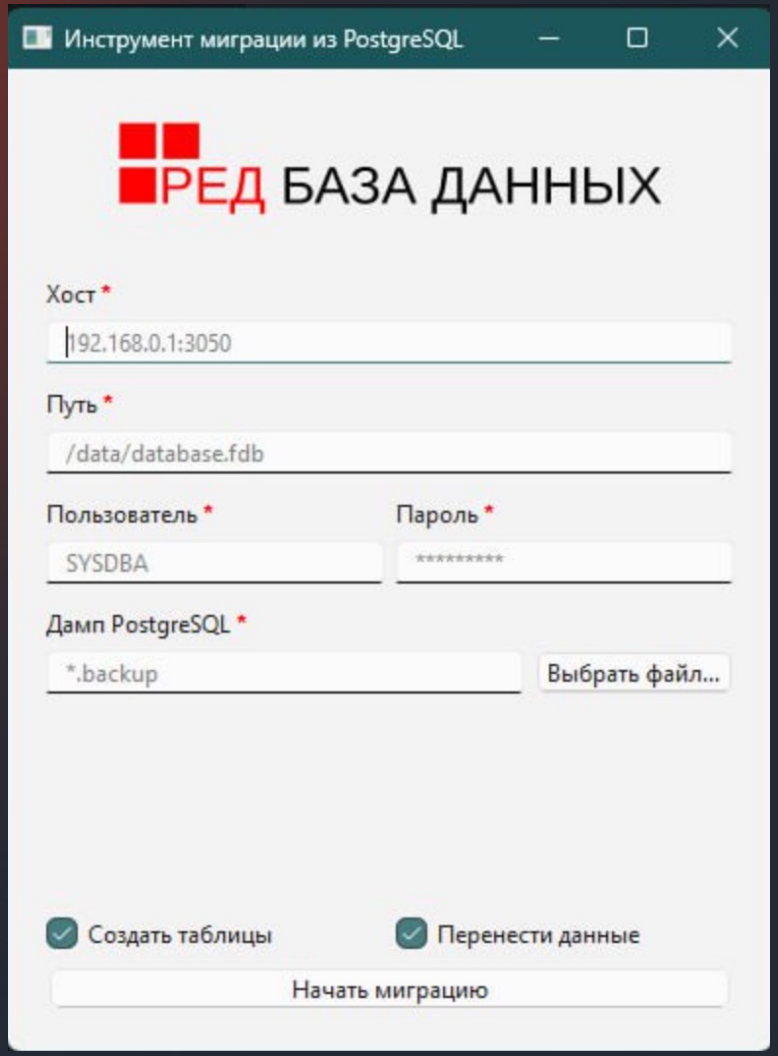
Диалект обеспечит полную интеграцию СУБД РЕД База Данных с SQLAlchemy, включая асинхронную работу через asyncio, поддержку ORM, гибкие запросы и тестируемость.

Экспорт и импорт данных

🎯 Цель проекта:

Графическое приложение на PySide6, позволяющее импортировать дамп PostgreSQL (*.backup) и перенести данные и структуру в базу СУБД Ред База Данных.

Пользователь указывает параметры подключения и запускает миграцию.



Экспорт и импорт данных

🔄 Что именно меняется:

PostgreSQL тип	Firebird тип	Комментарий
`text`	`blob sub_type 1`	Firebird не имеет прямого TEXT, заменён на BLOB
`timestamp without time zone`	`timestamp`	Упрощено
`"имя" тип, "имя" тип, ...`	`"имя" тип, ...`	Все имена оборачиваются в кавычки

Экспорт и импорт данных

** 📎 Пример миграции таблицы:

PostgreSQL исходник:

```SQL
CREATE TABLE books (
    id serial PRIMARY KEY,
    title text,
    published timestamp without time zone
);
```

В Firebird (через FirebirdDump):

```SQL
create table "books"("id" integer, "title" blob sub_type 1, "published"
timestamp);
alter table "books" add constraint "books_pkey" primary key ("id");
```

Опыт взаимодействия с командой тех поддержки Ред Базы Данных

- Создан отдельный рабочий чат
- Оперативное подключение по VPN
- Анализ логов и конфигурации СУБД
- Настройка параметров системы
- Скорость решения вопросов

Сложности, с которыми сталкиваются «новички»

- Настройки мониторинга
- Создание statements.fdb – не очевидно
- "Auth_plugins": ["Srp256"] – для 5-ой версии

"Не для опыта, для развития"

Для чего нам это как компании разработчику?

- Кроссбазность наших продуктов в портфеле
- Возможность создания ПО в рамках заказной разработки основовываясь на уникальном опыте

"Не для опыта, для развития"

Приобретенные компетенции:

1. Гибкость для заказчиков
2. Импортозамещение и регуляторные требования
3. Тестируемость и надёжность
4. Расширение рынка

💡 Что готова решать компания с такими компетенциями:

- Создание продуктов уровня “enterprise-ready”, независимых от инфраструктуры.
- Массовая адаптация под клиентов с разными СУБД — без отдельной ветки разработки.
- Переезд с одной СУБД на другую без глобальных изменений архитектуры.
- Формирование собственной библиотеки совместимости / диалектов как стратегического актива.

Благодарю за внимание!

29 МАЯ | МОСКВА

FIREBIRD CONF 2025