DIGITAL DESIGN

Технические особенности портирования кода и данных из MS SQL в PG на примере перевода СДУ «Приоритет»

Жуковец Юрий

Компания Digital Design

Digital Design — одна из ведущих ИТ-компаний России — оказывает комплексные услуги по оптимизации бизнеса с помощью последних достижений информационных технологий. Компания обладает обширной и уникальной экспертизой в сфере автоматизации: от внедрения готовых решений до сложных заказных разработок, позволяющих удовлетворить любые индивидуальные потребности клиентов.

С 1992 года экспертами Digital Design реализовано более 2 500 проектов для организаций, работающих в различных отраслях бизнеса: транспортных компаний, банков, промышленных предприятий, торговых сетей, а также для государственных организаций.

Мы постоянно думаем о людях, которые будут пользоваться нашими решениями. В Digital Design внедрена практика User eXperience (UX), что позволяет получить удобный и красивый пользовательский интерфейс.

Представительства Digital Design расположены в Москве, Санкт-Петербурге, Мурманске и Саратове.

Краткая информация о СДУ «Приоритет»

- Разработана на базе **российской платформы** Docsvision Core
- Реализация решений на уровне программного API.
- Проекты с более чем 50000 одновременных подключений на сервер и объемом документооборота в десятки миллионов документов.
- Система соответствует требованиям **российских и международных стандартов** и нормативных документов в области делопроизводства (ГСДОУ, ГОСТ Р 51141-98, 6.30-2003, 15489-1-2007).

Нам доверяют

Система внедрена более чем в 50 организациях:

Postgres

РОСИМУЩЕСТВО

FEDERAL AGENCY FOR STATE PROPERTY MANAGEMENT

MS-SQL

Министерство экономического развития РФ

МИНПРОМТОРГ РОССИИ

Агентство по страхованию вкладов

СБЕРБАНК

NOVARTIS

АИЖК

Минсельхоз России

Росрыболовство

Росграница

Росавиация

Рослесхоз

Минспорта России

Росморпорт

РОСКОСМОС

РОСАККРЕДИТАЦИЯ

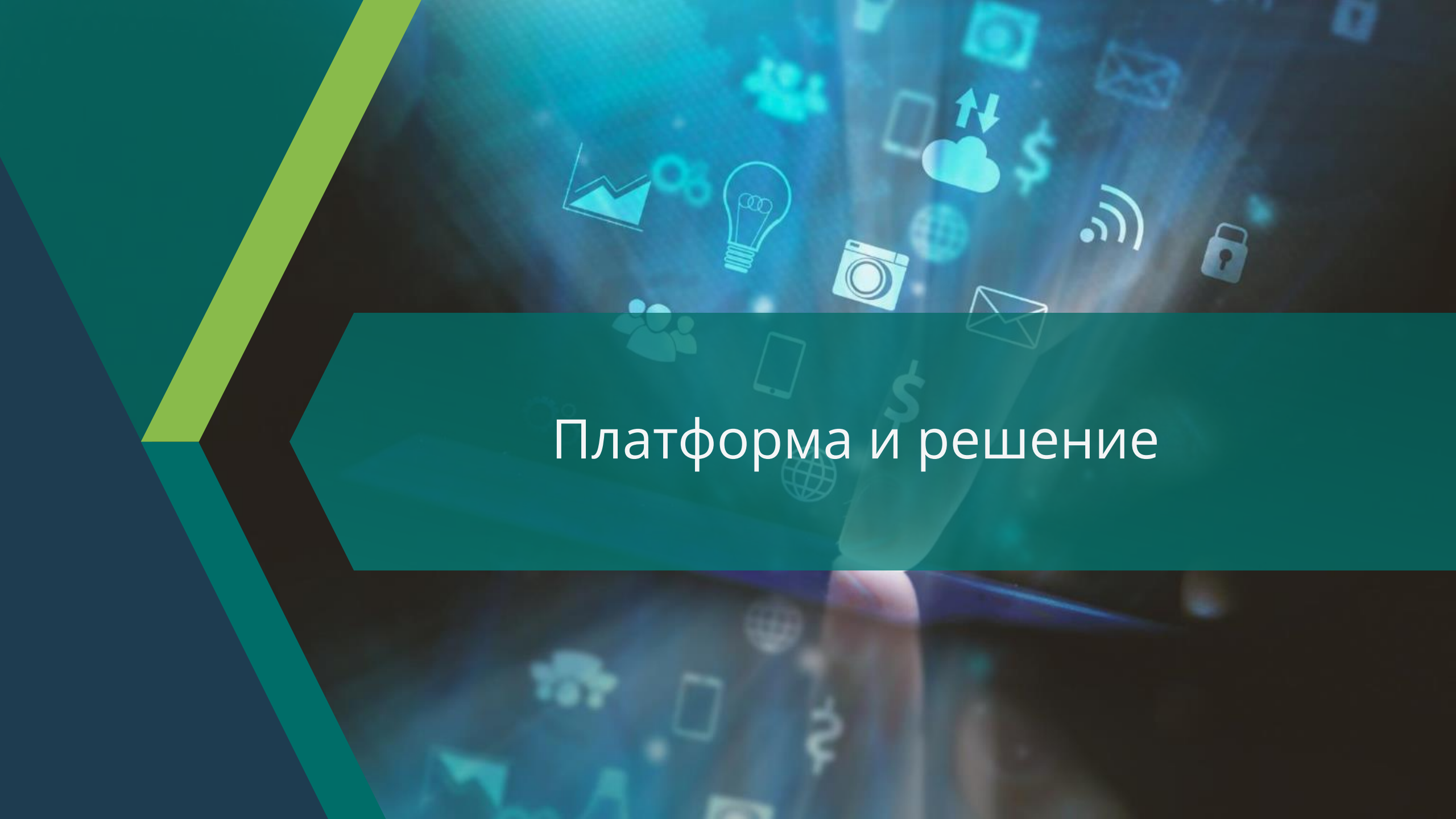
грузовая компания

МинКрыма России

ССК Сибирская Сервисная Компания

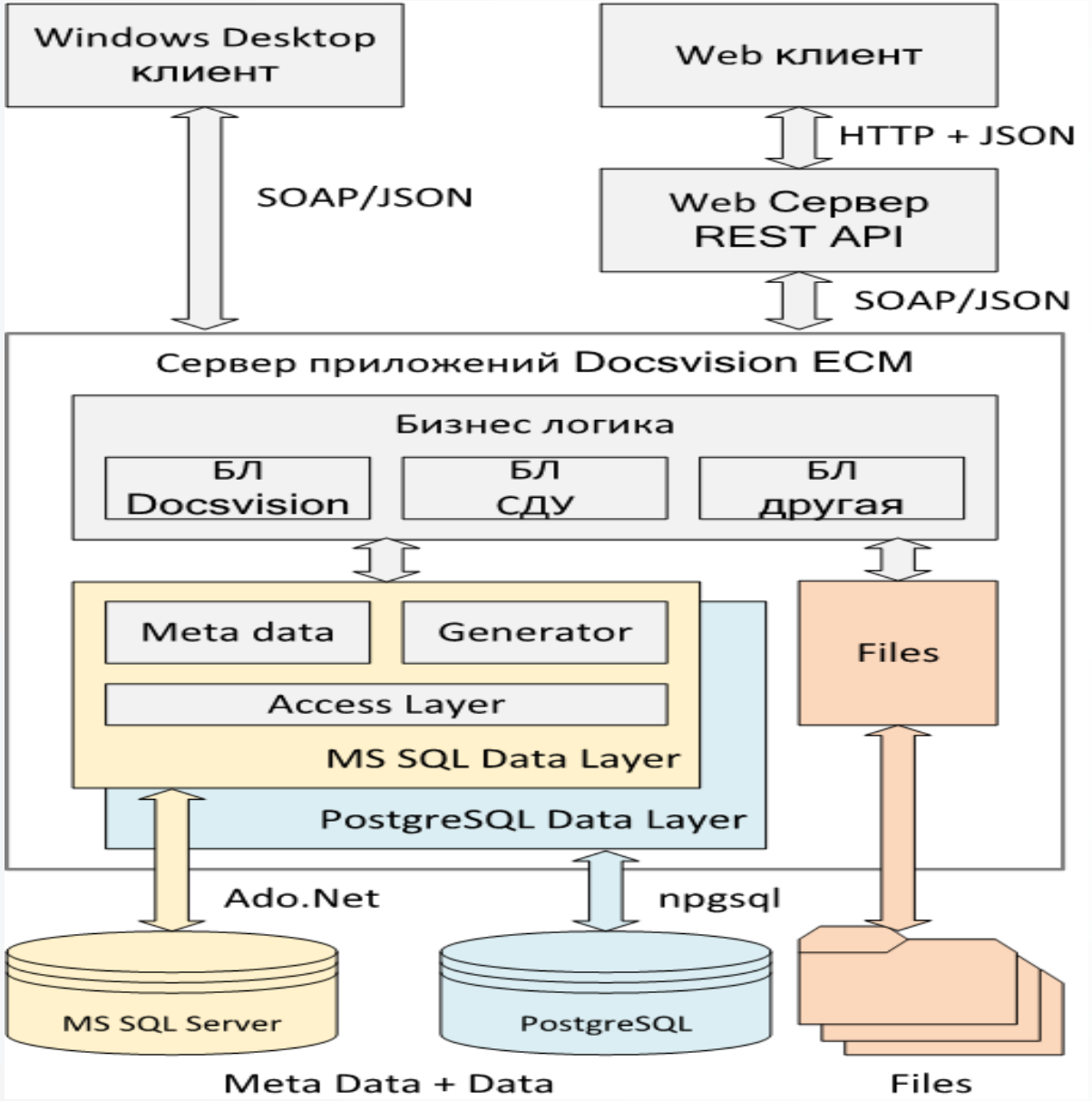
ПОЧТА РОССИИ

УНИВЕРСАЛЬНАЯ ЭЛЕКТРОННАЯ КАРТА



Платформа и решение

Архитектура.



- **Выделили и реализовали уровни:**
  - Общий уровень мета описаний с возможностью переопределения для конкретной СУБД
  - Генераторов
    - БД
    - Поиски
    - представления
  - Обращения к БД
  - Отдельного хранения файлов
- **Серве прилождений на .Net Core**
- **Развитый тонкий клиент**

Специфика БД

- **Набор «статичных» таблиц** *(> 100)* **и функций/процедур** *(> 200 ~ 25000 строк plpgsql кода)*
- **Расширение набора объектов при установке решений**
- **Авто создаваемые объекты при настройке системы и онлайн работе пользователей**

Ещё около 20000 строк кода чистого plpgsql + 10000 по генераторам .Net

- **XML мета-описания статических и динамических объектов, статический код отчетов**
- **«Умное» обновление БД** *(авто код > 1500000 строк)*
- **На уровне БД реализовано 99,99% логики получения подготовленных данных (сложные функции в БД и динамик plpgsql)**

Специфика трафика и работы с БД

- **Возврат множественных разнотипных наборов из функций**
- **«Временные» наборы в БД**
- **Вызов множества хранимых процедур в рамках одного сценария в транзакции и без**
- **Передача коллекций из клиентского кода через xml и сессионные временные таблицы**
- **Триггеры денормализации**
- **Внешняя дискретная безопасности по объектам/части объектов**
- **Активное использование временных таблиц**

Код с T-SQL на plpgsql

Что рассмотрено ранее https://youtu.be/v6_4Szr8t14

- **Уровни системы**:
  - Безопасность
  - Отличие реализации физического хранения
  - Системные моменты

- **Влияющие на код**
  - Collation
  - Ограничений объектам БД (типы данных, наименования, индексы)
  - Блокировки и hint
  - Отличия в работе временных объектов
  - Обращение к другим БД
  - Отсутствие встроенного авто-обработчика (pg agent)
  - Отличия в языке
  - Пакетные скрипты и особенности динамика
  - Отличия в работе хранимых процедур
  - Отличия в работе транзакций
  - Триггеры

Перенос данных в PG

Перенос размышления

- Скорость переноса
- Объем данных
- Файлы/блобы в БД
- Состав данных – что непосредственно переносим
- Наличие несовместимых/кастомных типов
- Необходимость дозагрузки
- Наличие дополнительного физического пространства
- Сетевой доступ/скорость связи

Способы переноса

- Различные утилиты
  https://wiki.postgresql.org/wiki/Converting_from_other_Databases_to_PostgreSQL#Microsoft_SQL_Server
  https://wiki.postgresql.org/wiki/Converting_from_other_Databases_to_PostgreSQL
- SSIS в MS SQL
- Перелить построчно через кастомное приложение
- Репликация из MS SQL в PG (https://www.hagander.net/blog/replicating-from-ms-sql-server-to-postgresql-103/)
- Экспорт в xml(подобное) вставка и разбор на уровне plpgsql
- Сгенерировать скрипт вставки (insert into .... values(...))
- Создать файл логического бэкапа
- **bcp экспорт- COPY вставка**

bcp экспорт- COPY вставка

- Плоский файл с разделителями
- Формат д.б. совместимый с PG COPY
- Загрузка в последовательности ссылочной целостности
- Предварительная проверка структуры

Что делаем – генератор:

- bat файла выгрузки из MS SQL
  - Обход требуемых таблиц в последовательности наличия связей
  - bcp – select
  - Select требуемых полей в текстовом виде совместимом с PG copy
- Скрипта загрузки в PG через COPY
  - Проверка наличия таблиц
  - Соответственно последовательности выгрузки

Генератор полей select – посмотрим код скрипта

```
select @export_from_ms_str = isnull(@export_from_ms_str + N', ', N'') +
+ case          when col.system_type_id = 189 then N'1'
when col.system_type_id = 104 then N'case t.[' + col.name + N']  when 1 then ''t'' when 0 then ''f'' else ''\N'' end'
when col.system_type_id in (58, 61) or col.system_type_id between 40 and 43 then N'case when t.[' + col.name + N'] is null then ''\N'' else convert(nvarchar(max), t.[' + col.name + N'], 121) end'
when col.system_type_id = 36 then N'case when t.[' + col.name + N'] is null then ''\N'' else lower(convert(nvarchar(40), t.[' + col.name + N'])) end'
when tp.name = 'text' then N'replace(replace(replace(replace(replace(convert(varchar(max), t.[' + col.name + N']), ''\'', ''\\''), nchar(10), ''\n''), nchar(13), ''\r''), nchar(9), ''\t''), nchar(11), ''\v'')'
when tp.name = 'ntext' then N'replace(replace(replace(replace(replace(convert(nvarchar(max), t.[' + col.name + N']), ''\'', ''\\''), nchar(10), ''\n''), nchar(13), ''\r''), nchar(9), ''\t''), nchar(11), ''\v'')'
when tp.name like '%text%' or tp.name like '%char%' or tp.name in ('sysname') then N'replace(replace(replace(replace(replace(t.[' + col.name + N'], ''\'', ''\\''), nchar(10), ''\n''), nchar(13), ''\r''), nchar(9), ''\t''), nchar(11), ''\v'')'
when tp.name in ('xml') then N'replace(replace(replace(replace(replace(convert(nvarchar(max), t.[' + col.name + N']), ''\'', ''\\''), nchar(10), ''\n''), nchar(13), ''\r''), nchar(9), ''\t''), nchar(11), ''\v'')'
when tp.name = 'image' then N'''\\'' + stuff(convert(nvarchar(max), convert(varbinary(max), t.[' + col.name + N']), 1), 1, 1, '''')'
when tp.name like '%binary' then N'''\\'' + stuff(convert(nvarchar(max), t.[' + col.name + N'], 1), 1, 1, '''')'
when col.system_type_id = 98 /*sql_variant*/ then N'
case          when sql_variant_property(t.[' + col.name + N'], ''BaseType'') is null then ''(0,,,,,,,)''
when sql_variant_property(t.[' + col.name + N'], ''BaseType'') = ''bit'' then N''(1,'' + case t.[' + col.name + N'] when 1 then ''t'' else ''f'' end + N'',,,,,,)''
when sql_variant_property(t.[' + col.name + N'], ''BaseType'') in (''bigint'', ''int'', ''smallint'', ''tinyint'') then N''(2,,'' + convert(nvarchar(40), t.[' + col.name + N']) + N'',,,,,)''
....'
else N'case when t.[' + col.name + N'] is null then ''\N'' else convert(nvarchar(max), t.[' + col.name + N']) end'
end + N' [' + col.name + N']' + nchar(13)
, @import_in_pg_str = isnull(@import_in_pg_str + N', ', N'') + N'''' + col.name +  N''''
from          sys.columns col
inner join sys.types tp
on col.system_type_id = tp.system_type_id
and col.user_type_id = tp.user_type_id
where col.[object_id] = @object_id
order by case when tp.name = 'image' or tp.name like '%binary' then 1 else 0 end asc, col.column_id asc;
```

Генератор select

- **timestamp - 1**
- **bit – 1/0 в t/f**
- **datetime типы – текстовое представление в 121 формат**
- **xml – в текст**
- **Во всех текстовых типах замена служебных символов**

  \ -> \\, [n]char(10) -> \n, [n]char(13) -> \r, [n]char(9) -> \t, [n]char(11) -> \v
- **null значения - \N**
- **binary типы и image – в текст и первый символ заменен на **
- **sql_variant – анализ свойства BaseType** (sql_variant_property(*столбец*, 'BaseType')) **и формирование текстового представления структуры**
- **Остальное как есть**

Проблемы при работе bcp и copy

- **Длинна команды bcp -> сделали набор представлений и экспорт из представления**

bcp "select * from ПРЕДСТАВЛЕНИЕ_ДЛЯ ВЫГРУЗКИ" queryout ПУТЬ_К_ФАЙЛУ.csv -S ИСХОДНЫЙ_SQL_SERVER -database_name ИСХОДНАЯ_БД -w –T

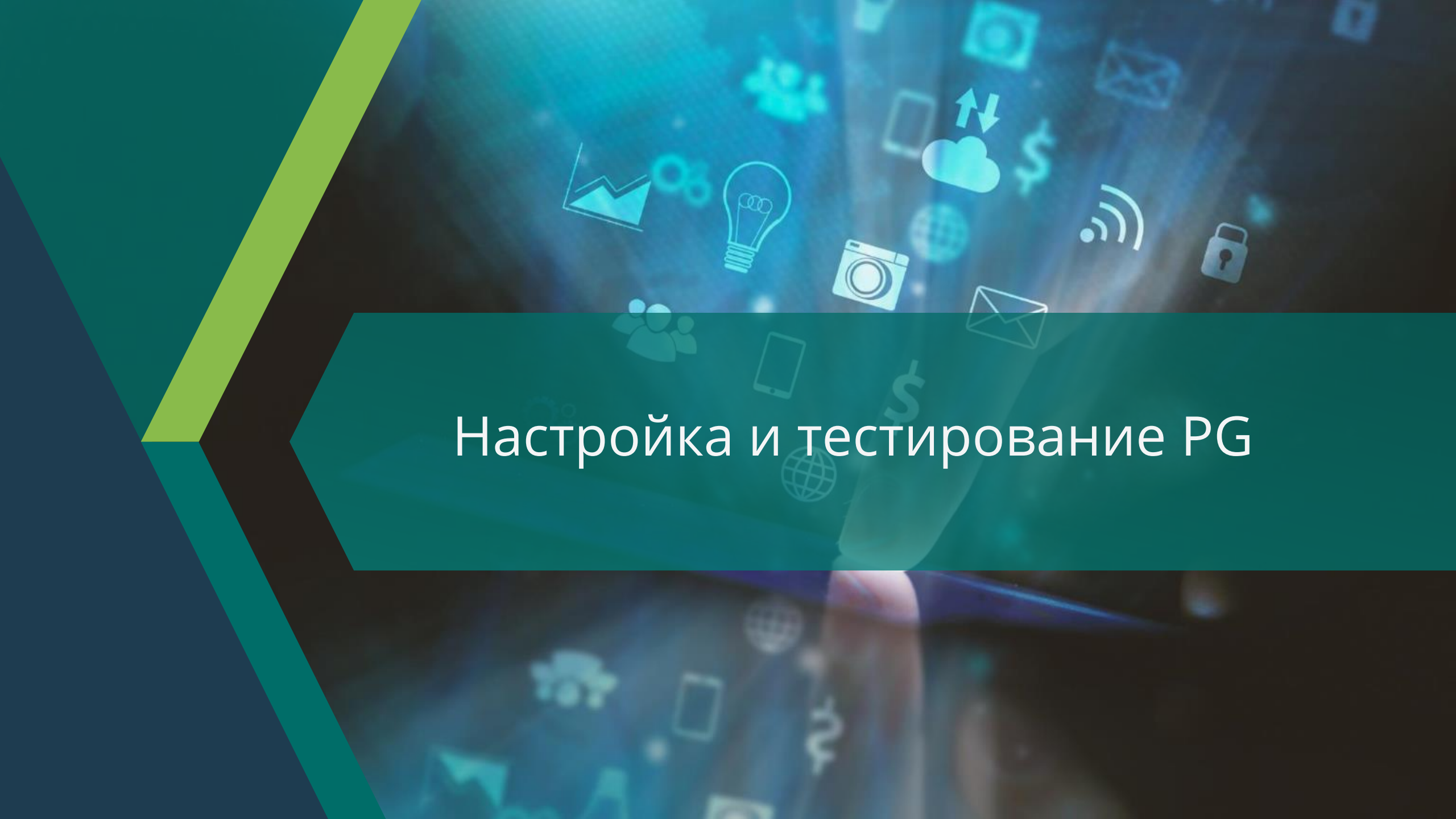
- **Кодировка файла для загрузки текста**

```
set client_encoding to 'win';
do language plpgsql
$$
begin
        if exists(select * from pg_class where relname = ‘таблица' and relkind = 'r') then
                truncate table public."таблица" ("Столбец1", … , "СтолбецТ") restart identity cascade;
                copy from ‘ПУТЬ_К ФАЙЛУ_таблица';
        end if;
end;
$$;
```

- **Пустые строки при работе bcp -** \x00

Предварительная обработка выгруженных файлов: конвертировать в utf или по быстрому(неявно и не совсем правильно) .ps1 скриптом:

```
function convertFile($filePath){(get-content $filePath) -replace '\x00','' | set-content $filePath}
convertFile(“путь_к_файлу.csv")
```



Настройка и тестирование PG

PG требует значительно более тонкой настройки

MS SQL	Postgres
Пытается забрать максимально все ресурсы => сразу более менее работает на всех ресурсах соответственно тому что дано	**должен запускаться на кофемолке ☹ => настройки для кофемолки, а не мощного сервера**

- **Что надо настроить:**
  - **Память (**shared_buffers, work_mem, maintenance_work_mem и т.д.**)**
  - **Параллелизм(**max_worker_processes, ax_parallel_workers_per_gather и т.д.**)**
  - **Настройки журнала транзакций –Wal+Checkpoint (**wal_writer_delay, wal_writer_flush_after и т.д.**)**
  - **Авто-вакуум**
  - **Доступ(pg_hba.conf)**
- **Основное – их много и могут сильно влиять**

PG - настройка

- **Cybertec PostgreSQL Configurator**
  http://pgconfigurator.cybertec.at/

- **Рекомендация настроек по результатам живого трафика** (+ плавно к тестированию производительности)

  mamonsu tune— агент мониторинга для сбора метрики оптимизации (https://postgrespro.ru/docs/postgrespro/11/mamonsu)

PG – сбор данных о работе системы

- **Mamonsu** — агент мониторинга для сбора метрики оптимизации https://postgrespro.ru/docs/postgrespro/11/mamonsu
- **Настройка PG – журналирование длительных запросов**
  - **#log_min_duration_statement**
  - **Красиво посмотреть и проанализировать журнал http://pgcookbook.ru/article/pgbadger.html**
- **Модуль auto_explain - план запроса(https://postgrespro.ru/docs/postgrespro/11/auto-explain)**
- **pg_stat_statements – статистика по запросам https://postgrespro.ru/docs/postgrespro/11/pgstatstatements**
- **PostgreSQL monitoring for Zabbix(на основе PostgreSQL Statistics Collector https://postgrespro.ru/docs/postgrespro/11/monitoring-stats) - статистика по физическим объектам таблицам, индексам и т.д. и по текущим запросам http://cavaliercoder.com/libzbxpgsql**

Кратко как тестировали

- MS Visual Studio
  - MS Visual Studio
    - MS Visual Studio Test Controller - для запуска и настройки самих тестов
    - MS Visual Studio Test Agent – на каждой машине используемой для генерации нагрузки (подключаются на Test Controller)
  - Нагрузка:
    - тестами покрывающим все основные бизнес операции
    - определенное кол-во операций (=тестов) определенного типа в единицу времени
  - Тест удачен если: выполнился без ошибок в требуемое время
  - > 98% тестов удачны = держим нагрузку
- Оптимизируем:
  - то что начинает падать первым
  - траффик по БД
    - долгие запросы
    - относительно быстрые но если очень много то д.б. сверх быстрыми

Особенности кода

Выявленные проблемы 1

- **Индексы –** ***долгие запросы*** **=> добавить недостающие по результатам планов**
- **Intersect –** ***Не умеет понимать селективность наборов по отдельности и реально объединяет независимо сделанные наборы*** **=> переделка на inner join**
- **Join большой таблицы по полю с разной селективностью с малой таблицей с доп. фильтрацией по второй –** ***периодический сбой плана причем не повторяется в ручном запуске без нагрузки –*** **анализ и переписывание запроса**

Выявленные проблемы 2

- **MS Like <> PG Like, MS Like = PG Similar to**
- **Доработка sql_variant**
- **Очень сильные требования по правильной типизации и использованию алиасов**
- **Временные таблицы лучше чем решения с массивами. При этом:**
  - **не пересоздаем**

```
if not is_temp_table_exists ('имя_таблицы') then
        create temporary table имя_таблицы(...);
else
        truncate table имя_таблицы;
end if;
```

  - **Analyze** имя_таблицы; после гарантированных изменений данных

Дополнительно

- Передача коллекций из клиентского кода проще и быстрее

```
Select res.Item::нужный_тип from unnest(regexp_split_to_array(текстовая_коллекия_значений, 'разделитель')::text[]) res(Item)
```

- При возврате таблицы (return query) из функции:
  - Выход только return;
  - Return как возврат пустого набора
  - проще делать ветвление по логике и альтернатива union all делать
- Unlogged table – как кэши на уровне PG
- returning – аналог output into, но только новые значения

Немного про автоматизацию

- Для ванильного и других – pgAgent
  - Связан с БД
  - Как вариант – поставить в основную и ходить через dblink/foreign table
- Postgres Professional
  - create extension pgpro_scheduler;

  alter system set schedule.enabled = on;
  - Расписание на основе cron

```
select id from schedule.get_cron() where name = job_name limit 1 into
job_id;
if job_id >= 0 then perform schedule.drop_job(job_id); end if;
perform schedule.create_job(format('{"commands": "select %s();", "cron":
"0 2 * * *", "name": "%s"}', 'какая-то функция pg', '"', '\"'),
job_name)::jsonb);
```

sql_variant

MS SQL - sql_variant:

- Произвольный тип
- Индексируется
- Самостоятельно определяет тип при вставке/возврате/сравнениях
- При авто-сравнении с не приводимым типом – exception

Postgres – прямой замены нет

- Что можно сделать:
  - Text + столбец для типизации => проблемы неоднозначного поиска
  - Расширение - собственный тип => сложность создания/установки/поддержки
- create type sql_variant as (type smallint, bl boolean, i bigint, guid uuid, fl float, dc numeric, dt timestamp, txt text, img bytea);
  - На уровне клиентского кода - как передать и выдавать (MapComposite для пользовательских типов)
  - Как искать и сравнивать (функциональные индексы)
  - Сделать функции и операторы сравнения
  - Функция приведения к тестовому ВИДУ

Мы ждем вас!

Вопросы!!!

www.digdes.ru

info@digdes.com

Санкт-Петербург
наб. реки Смоленки, д. 33
телефон: +7 812 346 58 33

Москва
Варшавское шоссе, д. 36, стр. 8
телефон: +7 499 788 74 94