DIGITAL DESIGN

Миграция Системы документационного управления «Приоритет» с MS SQL на Postgres

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 — специализированной версия платформы, оптимизированной для разработки решений с высокой нагрузочной способностью и масштабированием.
- 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:

Министерство экономического развития РФ

МИНПРОМТОРГ РОССИИ

Агентство по страхованию вкладов

СБЕРБАНК

Минсельхоз России

Росрыболовство

Росграница

Росавиация

Рослесхоз

Минспорта России

грузовая компания

УНИВЕРСАЛЬНАЯ ЭЛЕКТРОННАЯ КАРТА

ФЕДЕРАЛЬНАЯ СЛУЖБА ПО АККРЕДИТАЦИИ РОСАККРЕДИТАЦИЯ

РОСКОСМОС

Росморпорт

NOVARTIS

МинКрыма России

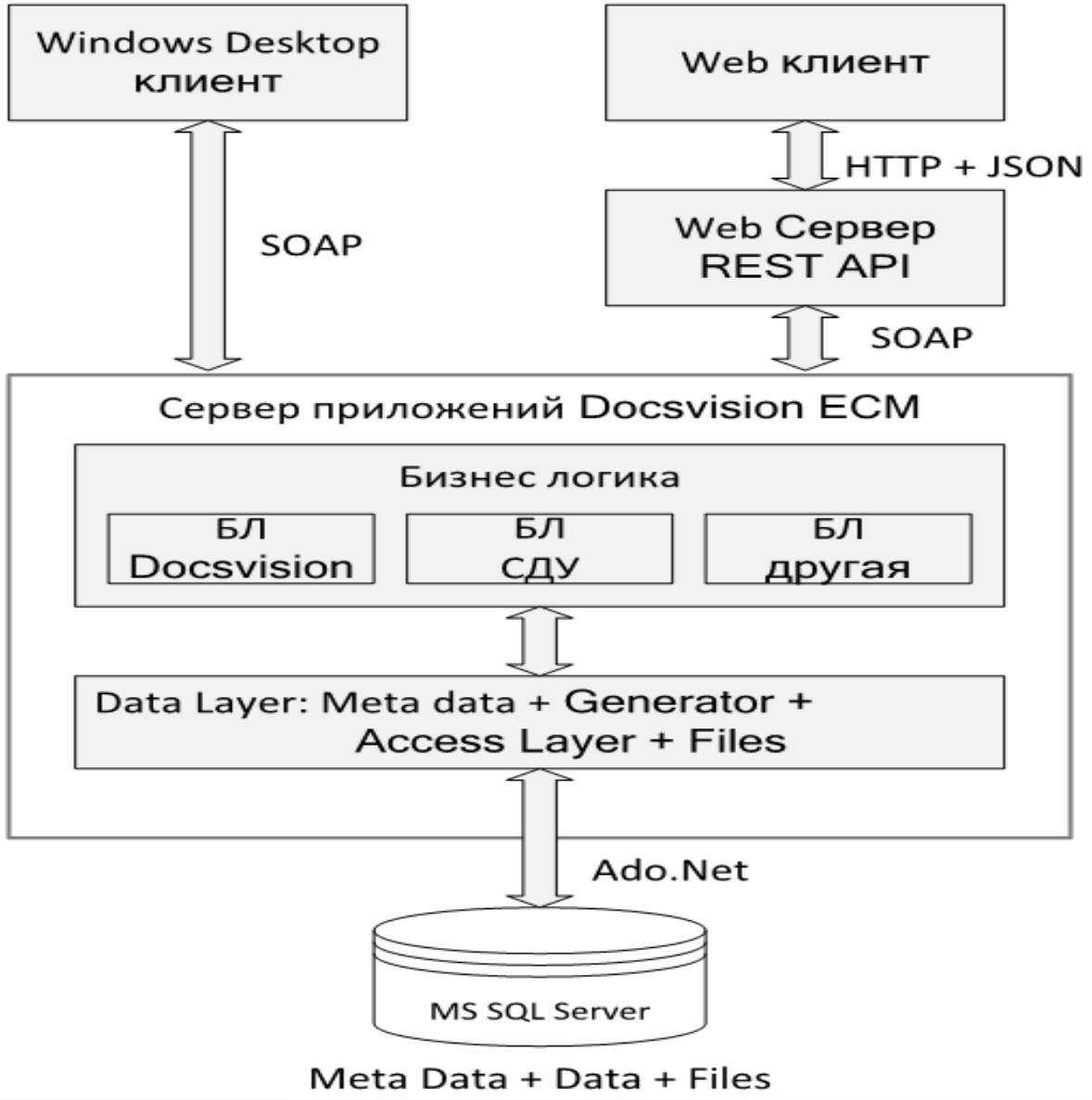
ССК Сибирская Сервисная Компания

ПОЧТА РОССИИ

АИЖК

Платформа DV

Архитектура платформы:



Состав

- Толстый клиент + тонкий клиент
- Сервер приложений:
  I. Интерфейсы для доступа к данным
  II. Скрывает логику доступа к бизнес-объектам
  III. Бизнес-логика решений
  IV. Data Access Layer
  V. Генерация запросов для поиска и объектов для отображения данных
- СУБД MS SQL

База Данных

Состав

- **Имеет набор «статичных» объектов**:
  - Таблиц ~ 100
  - Бизнес объектов (функций/хранимых процедур) ~ 200 (25000 чистого sql строк кода)
- **Расширяется при установке в платформу бизнес-решений (СДУ Приоритет)**:
  - Новые таблицы
  - Связанные авто генерируемые объекты для получения/изменения данных
  - Статичные хранимые процедуры для получения данных отчетов
- **Автоматически создаваемые в процессе настройки системы и работы пользователя объекты (таблицы/ хранимые процедуры /представления)**
  Механизмы:
  - Поисков (не sql) – динамический скрипт или автоматически создаваемая процедура
  - Представлений (не sql) – динамический скрипт или набор автоматически создаваемых процедур

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

Источник sql кода

- **xml описания таблиц – полное описание объектов с типизацией и составом**
  - таблицы
  - ограничения
  - индексы
- **xml описания статических бизнес объектов + связанный sql код**:
  - заголовки
  - параметры
  - возвращаемый результат
- **прототипы(шаблоны) для хранимых процедур доступа к генерируемым объектам при загрузке решений**
- **прототипы(шаблоны) для механизмов генераторов**
  - « Поиски»
  - «Представления»
- **Расширения со статичным sql кодом**

Установка/обновление БД

- **Генерация скрипта на основании описаний**
  - описаний системных объектов
  - загруженных решений
  - настроенных пользователями расширений решений
- **Скрипт должен учитывать изменение объектов**:
  - Смену типизации полей
  - Наличие и изменение появление индексов, ограничений, связей между объектами
- **Удаление устаревших авто-созданных при работе пользователя объектов**

Количество строк генерируемого кода при установке обновлении БД
~ 1 500 000

Специфика трафика и работы с БД

- **На уровне БД реализовано 99,99% логики получения подготовленных данных** для выборки на клиент (сервер приложения)
- **Бизнес-логика обработки данных реализована на уровне сервера приложений**
- **Преимущественно использование хранимых процедур**

изменение данных и получение данных о бизнес объектах только через хранимые процедуры.

- **Наличие сложной логики по получению данных с ветвлениями внутри sql-кода**
  - В хранимых процедурах
  - В динамических запросах
- **Активное использование временных таблиц и табличных переменных**

Специфика трафика и работы с БД

- **Возврат множественных наборов в одиночных вызовах**
  - Хранимых процедур
  - Динамических скриптах
- **Использование БД как место хранения «временных» наборов между обращениями клиента для поисков и представлений**
- **Множество сценариев последовательного вызова хранимых процедур**
  Используется промежуточное сохранение результатов во временные таблицы с обработкой их результативного наполнения на уровне сервера приложений и дальнейшего использования данных временных таблиц следующими хранимыми процедурами
- **Передача коллекций через xml**
- **Наличие триггеров для денормализации данных на уровне БД**
- **Существование внешней дискретной безопасности по объектам/части объектов**

Задачи

Основные задачи

- **Перевести на СПО**
- **Бизнес-логика не должна быть затронута**
- **Полная совместимость с текущими решениями**
- **Поддержать на уровне платформы несколько СУБД в одной сборке без создания отдельного решения.**
- **Единый код для Windows и Linux для серверной части**
- **Свободные комплектации ОС + СУБД**
- **Дополнительные особенности** – только стандартные сборки СПО

Первые мысли о переходе

УЖАС!!!

1 000 000 человеко-лет всё переделать

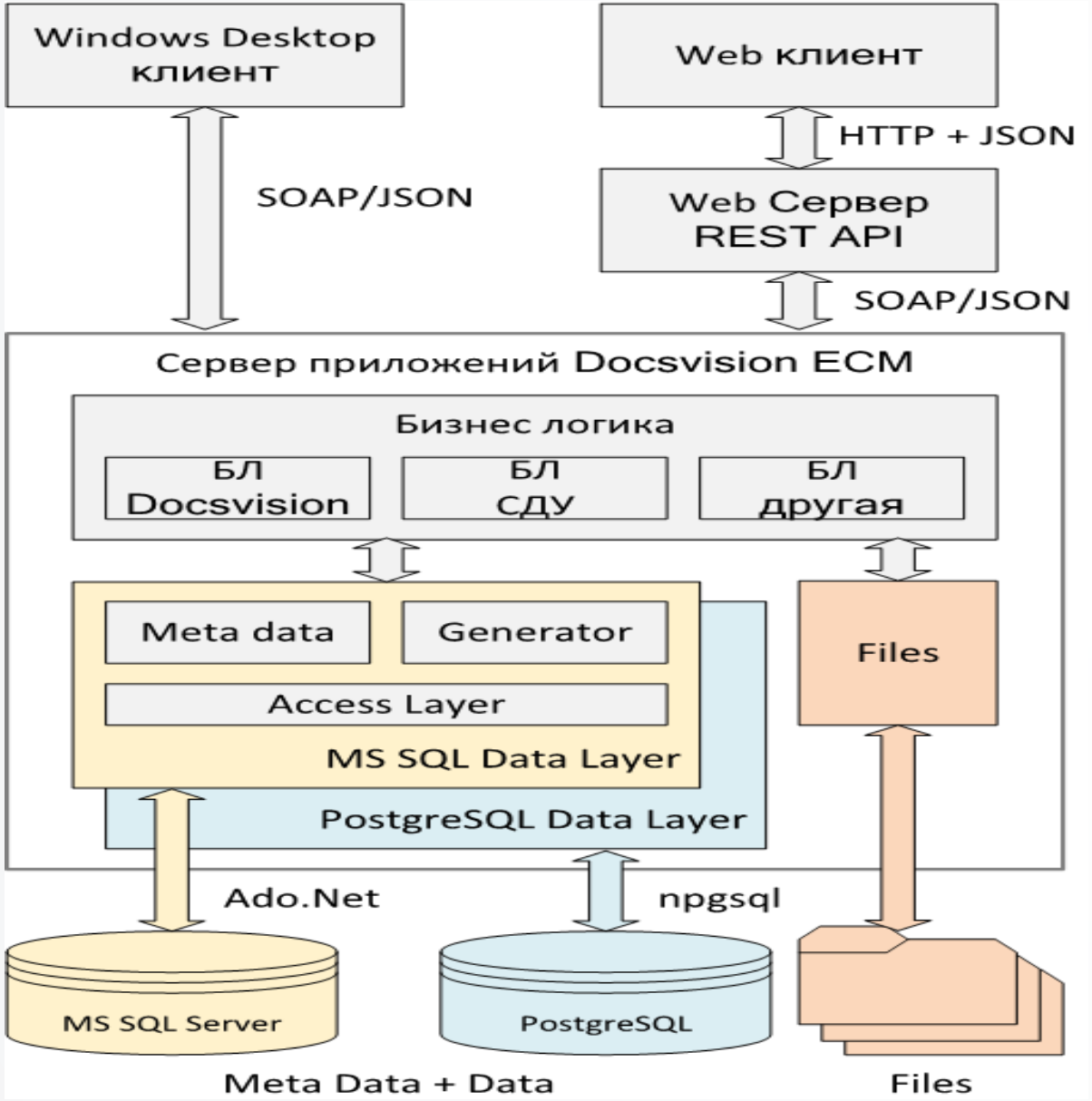
Мысли о переходе

- **Подумав и изучив**:
  - **Реализуемо относительно просто и в адекватные сроки и ресурсы**
  - **Выбран .net core + rest(json)**
  - **Выбран Postgres/Postgres Prof 9.5 и выше**

- **Переход**:
  - **Сроки ~ 10 ч.м.**
  - **Ресурсы**:
    - **PG** – 1,5 ч.г (код pgsql + доработка .net core кода)
    - **Linux** - переработка северной части около 9 ч.м. – в основном поддержка нового протокола
    - Не учтено много тестирования

Платформа DV после переработки

Архитектура. Что сделали.



- **Выделили и реализовали уровни:**
  - Общий уровень мета описаний с возможностью переопределения для конкретной СУБД
  - Генераторов
    - БД
    - Поиски
    - представления
  - Обращения к БД
  - Отдельного хранения файлов
- **Мигрировали(адаптация) кода с .Net 4.6.1 на .Net Core**
- **Развили тонкий клиент**

Переход на PG

О чем нужно подумать

- **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

Безопасность

MS SQL

- **Двух уровневая безопасность**: авторизация на уровне сервера и авторизация/аутентификация на уровне БД. (возможное упрощение Contained Databases);
- **Позволяет прозрачно работать с другим БД**
- **Плотная интеграция с AD**
- **Ограничения на уровне протоколов**

Postgres (более автономная работа БД)

- **Одноуровневая безопасность**. В общем случае пользователь + пароль
- **Для работы с другими БД надо использовать расширения** (dblink/postgres_fdw)
- **Есть псевдо SSPI авторизация**
- **Ограничения на уровне протоколов значительно шире и сильнее** (например, по IP для TCP IP)

Отличие организации реализации физического хранения

MS SQL - Хранение идёт от БД

- БД = коллекция файловых групп (минимум одна)
- Файловая – коллекция физических файлов (минимум один)
- Собственный файл(ы) журналов транзакций
- Физический бэкап идет от БД

Postgres

- Табличные пространства на уровне сервера, в которых располагаются БД(ых)
- WAL – один на всех
- Физический бэкап всех БД
- Существует логический бэкап

Отличие физического хранения данных

MS SQL

- Только последняя версия данных на странице
- Плотное расположение данных на странице
- Возможно работа со снимками данных при транзакциях

Postgres

- Версионированное хранение на уровне страниц данных
- Vacuum
- «Плотность» зависит от последовательности полей при создании таблицы

Системные моменты

MS SQL

- Набор обязательных, не удаляемых системных БД
- Автоматические настройки работы рассчитанные на максимальное использование ресурсов

Postgres

- Одна системная БД + БД чистый прототип
- Наличие различных автоматических настроек работы уровня системы, рассчитанных на минимальное использование ресурсов
- Значительно широкие настройки по использованию памяти

Код – Collation

Collation

- **Текстовые сравнения** (Кодировка, Сортировка, Сравнения)

- **Источник**:
  - зависит от локэйла сервера
  - Устанавливается для сервера при установке и потом не поменять
  - Можно изменить вниз по иерархии: для БД, для таблицы, для столбца

Основное отличие – **Case Insensitive/Sensitive**

Код – Collation - Case Insensitive/Sensitive

MS SQL	Postgres
▪ Зависит от *Collation* ▪ Влияет на: • Системную информацию (включая имена объектов) • Текстовые сравнения данных • Можно указать при сравнении другой *Collation*	▪ Изначально Case Sensitive ▪ Не влияет на наименование объектов ▪ Для Case Insensitive сравнений необходимо использовать специальные операторы/преобразования/типы: • ilike • Преобразования lower *(столбец)* = lower *(столбец/выражение)* • Расширение *citext*

Код – Collation - Case Insensitive сравнения и поиски

Способ решения	Преимущества	Недостатки
Столбец ilike *столбец/паттерн*	• Легко реализуется • Практически ничего не надо менять в коде.	• Нет возможности использовать индексы
Преобразования к регистру: lower/upper(*столбец*) =/like lower/upper(*столбец/выражение/паттерн*)	Возможен быстрый поиск используя индексы для выражений: *столбец = столбец/выражение* like *'текст%'*	• Необходимо везде учитывать в коде; • Специфика создания индексов `Create index` *индекс* `on` *таблица* (`lower/upper`(*столбец*) `text_pattern_ops`);
Использование расширения citext	• Легко реализуется; • Ничего в коде менять не требуется; • Возможен быстрый поиск используя индексы по условию *столбец = столбец/выражение*.	Индексы не работают на поисковых условиях *столбец* like *'текст%'* **Примечание – пока не работает ☺*

Ограничения по объектам БД: наименования, индексы

	Ms sql	Postgres

Наименования объектов (таблицы, столбцы, хранимые процедуры и т.д.)	• **Стандартно *схема.таблица;***
	• **Длинна 256 символов;**
	• **Регистрочувствительность в зависимости от collation БД;**
	• **Сложные наименования оборачиваются в []/”” при этом это не влияет на регистрочувствительность.**
		• **Стандартно *схема.таблица;***
		• **Длинна 64 (до 128 PG prof enterprize);**
		• **Сложные наименования оборачиваются в ”” - влияет на регистр чувствительность.**
		Решения:
		• **Выверка кода по:**
		  ○ **Использованию ””**
		  ○ **Приведению наименования к регистру;**
		  ○ **Уменьшению наименования авто генерируемых объектов.**
		• **Для сложных зависимых объектов (индексов) - формирование имени как стандартный префикс + hash от полного имени.**

Индексы	• **B-tree: кластерные и не кластерные** • Column store: только в последних стали авто обновляемые поэтому в OLTP малоиспользуемые; • Хэш для in memory.	• **B-tree**, хеш, gist, sp-gist, GIN и BRIN. ***Нет кластерных индексов*** ***Есть функциональные индексы***

Ограничения по объектам БД: типы данных и связанная функциональность

- **Практически полные аналоги**

- **Особенности работы и исключения:**
  - ***bit*** **MS SQL и** ***boolean*** **в PG;**
  - ***timestamp*** **- не путать с t*imestamp* в PG**
  - **Поля с авто счётчиками и** ***uniqueidentifier***
  - **Размышления по** ***char(n)/varchar(n/max)/text***
  - ***sql_variant*** **– свободный тип данных**
    - **разные подходы реализации хранения**
    - **необходимо учитывать передачу в и получения из базы**

Ограничения по объектам БД: bit и boolean

- **bit MS SQL и boolean в PG – не совместимы с точки зрения четкого приведения**

  ***0/1* не одно и тоже что *false/true*: при попытке сравнения boolean *столбца/переменной* с *0/1* получаем ошибку**

```
1::boolean = true
0::boolean = false
```

- **В PG тип удобнее, т.к. является истинным bool:**

  выражение = [not] [столбец/переменная]

  Например:

  where [not] t.столбец

Ограничения по объектам БД: – *timestamp*

timestamp **- в MS SQL тип поля**

- **соответствует единому на БД авто счётчику изменений в таблицах, где присутствует данный тип**
- **В таблице может быть только одно поле с таким типом**

Идея!!!

- ***timestamp*** **практически соответствует номеру транзакции вставки версии строки - xmin, НО xmin в обычном Postgres - 32**
- **Решено использовать поле с типизированным названием *SysRowTimestamp и* привязанным к нему sequence**

- **sequence**

```
create sequence public.dbts increment 1 minvalue 1 maxvalue 9223372036854775807 start 1
cache 1;
```

- **bigint default(sequence)**

```
SysRowTimestamp bigint not null default(nextval('public.dbts'))
```

- **триггер на изменение данных в таблице**

```
create function dvsys_trigger_before_set_srts() returns trigger as $$
begin
        NEW."SysRowTimestamp" = nextval(''public.dbts''); return NEW;
end;
$$ language      plpgsql;
create trigger "таблица_upd_ts" before update on "таблица " for each row execute
procedure dvsys_trigger_before_set_srts();
```

вместо: ***@@dbts***

```
create or replace function dvsys_dbts() returns bigint as $$
begin
        return (select last_value from dbts);
end;
$$ language plpgsql stable;
```

Ограничения по объектам БД: поля с авто счётчиками и uniqueidentifier

	Ms sql	Postgres
Автосчетчик	**smallint/int/**bigint **+ модификатор identity(*старт, шаг*)** **Последнее значение** `@@identity`/`scope_identity()` С SQL 2012 – объект **sequence**, который можно использовать в ограничении ***default*** (не используем т.к. должна быть совместимость с SQL 2008).	• **sequence** • **Типы данных с автоматически создаваемыми sequence: smallserial/serial/ bigserial**
Тип для GUID	**uniqueidentifier** `Newid()`/`default(newid())`/`default(new sequentialid())` **Последнее вставленное значение получить сложно: *output inserted.Поле into [@/#]таблица(столбец)***	• **uuid** `create extension "uuid-ossp";` • **uuid_generate_v1()(_v1mc()/_v3()/_v4()/_v5())**

PG – получение одного вставленного значения:
`insert into` таблица(…) … `returning` [столбец со значением по умолчанию], … `into` переменная …;

Ограничения по объектам БД: размышления по char(n)/varchar(n/max)/text - 1

Типы данных	Ms sql	Postgres
Char(n)	Текст + пробелы с учётом ограничения длинны *При конкатенации пробелы НЕ игнорируются*	Character Текст + пробелы с учётом ограничения длинны При конкатенации пробелы игнорируются
Varchar	Текст переменной длинны: *varchar(n)* с ограничением длинны *Varchar(max)* без ограничения	Character varying; Текст переменной длинны; varchar(n)- с ограничением длинны; Varchar - без ограничения.
Text	Устаревший тип Не поддерживает стандартные текстовые операторы С SQL 2005 *varchar(max)*	Текст без ограничений = *varchar*

Ограничения по объектам БД: размышления по char(n)/varchar(n/max)/text - 2

Специфика типа	MS SQL	Postgres
Ограничение по длине	**Имеет смысл:** • **Проверка длинны строк** • **В связи с особенностью хранения данных изменения данных по месту текущей строки**	**Имеет смысл:** • **Только если нужна проверка длинны строк на БД, что должно быть перекрыто бизнес логикой** **Плохо:** • **Доп. нагрузка на БД по проверке длины**
Юникод	**Определяется приставкой n для типа (*nchar, nvarchar*), N для констант**	**Определяется LC_CTYPE при создании.**
Выводы	**В postgres всегда использовать типы без ограничений:** • ***Text*** • ***Varchar/character varying*** • **Выбрать и везде использовать только один тип на всё решение(проблемы с типизацией при создании функций)** • **Подумать о *citext***	

Ограничения по объектам БД: text – конкатенация

MS SQL – «+»
Postgres – «||»

- **Функция оператора**

```
create or replace function text_add(leftarg text, rightarg text) returns text as
$$
begin
        return leftarg || rightarg;
end;
$$
language plpgsql immutable strict;
```

- **Создание оператора**

```
create operator + (leftarg = text, rightarg = text, procedure = text_add, commutator = +);
```

Ограничения по объектам БД: размышления по sql_variant

MS SQL - sql_variant:

- **Для хранения значения произвольного типа**
- **Индексируется**
- **Самостоятельно определяет тип при вставке/возврате/сравнениях**

Postgres – прямой замены нет

- **Что можно сделать**:
  - `create type sql_variant as (b boolean, i bigint, num numeric, fl float, dt timestamp, txt text, bt bytea);`
  - **Text + столбец для типизации**
  - **Расширение - собственный тип**
- **О чем подумать**:
  - **Как передать и выдавать** (*MapComposite* для пользовательских типов)
  - **Как искать и сравнивать**(функциональные индексы)

Hint и Блокировки

MS SQL

- **Hint:**
  - **Уровень доступа**
  - **Как работать запросу**
  - **Хранению плана выполнения**

Postgres

- **подсказок нет**
- **нет блокировок чтения – версионированное хранение данных**

Результат: просто всё удалить.

Отличия в работе временных объектов - принципы

MS SQL	Postgres

• **Табличные переменные *declare @ var_tbl table*(*col1 type, col2 type, …*)**
  ○ **Область видимости – текущий блок** кода/программный объект
  ○ **Время жизни – выполнение кода**
  ○ **Можно передавать как параметр**
• **Временные таблицы *create table # var_tbl* (*col1 type, col2 type, …*)**
  ○ **Область видимости – текущий блок кода/программный объект и вызовы вложенного кода**
  ○ **Время жизни – текущее соединение в рамках блока объявления кода**
• **Возможны идентичные по наименованию временные таблицы во вложенном коде** - можно обращаться к выше созданной временной таблице
• **Физически располагаются в отдельной БД**
• **Имеют независимую и упрощённую транзакционную нагрузку от основной БД**
	• **Временные таблицы *create temporary table var_tbl*(*col1 type, col2 type, …*)**
	• **Время жизни и область видимости – текущее соединение с момента создания**
	• **Надо учитывать наличие таблицы в нижележащем коде** – нельзя создать временную таблицу с таким же наименованием
	• **Физически располагаются в той же БД и аналогичны обычным таблицам**:
	  ○ Отдельная временная схема
	  ○ Есть оптимизации от Postgres Prof

Отличия в работе временных объектов - приёмы

- **Вместо простых табличных переменных – массивы**
- **Проанализировать код на создание аналогичных по наименованию временных таблиц во вложенных функциях**
- **Функция для удобства: проверка существования, удаление**
- **nologged таблицы для временного хранения данных**

Функция для удобства:

- ***проверка существования***

```
create or replace function public."help_is_temp_table_exists"(val_Name varchar(128)) returns boolean as
$$
declare val_int integer;
begin
        if exists (select * from pg_catalog.pg_class r
                                        where r.relnamespace = pg_my_temp_schema() and r.oid =
val_Name::regclass::oid and r.relkind = 'r') then
                return true;
        end if;
        return false;
        exception when others then
                return false;
end;
$$ language plpgsql stable;
```

- ***удаление***

```
create or replace function public."help_drop_temp_table"(val_Name varchar(128)) returns boolean as
$$
begin
        if public."help_pg_is_temp_table_exists"(val_Name) then
                execute format('drop table %s cascade', val_Name);
                return true;
        end if;
        return false;
end;
$$ language plpgsql volatile;
```

Обращение к другим БД

MS SQL server

- **Прозрачное обращение к соседним БД**
- **Database.[Shema].Object**
- **Права через логин уровня сервера**

Postgres

- **Расширения "dblink" – при использовании постоянно указывать параметры соединения (аналог openrowset)**
- **Расширение "postgres_fdw" – создание таблиц обёрток foreign table для обращения к внешним таблицам, но есть проблемы при наличии default значений и счетчиков в базовых таблицах**

Что сделали:

- **Пока исключили работу с несколькими БД**

Отсутствие встроенного авто-обработчика (pg agent)

MS SQL

- Встроенный сервис SQL server agent
- Объекты автоматизации в БД msdb
- API из системных процедур/функций/представлений

Postgres – нет встроенного сервиса, ждем от pg prof. Пока используем pg_agent

- Надо отдельно ставить сервис и скрипт
- Появляется схема pgagent со служебными таблицами аналогичными по job-ам в sql
- Расписание дискретно относительно запуска в течении дня

Что сделали для работы из контекста рабочей бд:

- Создаём foreign table для всех таблиц pg_agent
- Сделали вспомогательные функции (создать job, добавить расписание, добавить шаг и т.д.)
- Для произвольного запуска (например каждый n-ть секунд) создаем триггер определяющий время следующего запуска

Отличия в языке

Много различных тонкостей, но основное по базовым моментам:

SELECT – практически полностью идентично, отличие: LIMIT *количество* вместо TOP *количество/проценты*

CTE:

- Для рекурсивности надо указывать *recursion (with recursion CTE()…)*
- Для исключения зацикливания при итерации - есть *union*

Aliasing –более жёсткие требования использования:

- Обязательно во вложенных подзапросах, if exists () если несколько условий, в возвратах наборов в функциях.

Update

Update **таблица** *[alias]* – ***указание таблицы обязательно***

Set **столбец** = *выражение/столбец* - ***столбец обязательно без [alias]***

[From запрос] [where набор условий] ***- нет обновления top записей***

Delete

Delete from **таблица** *[alias]*

*[**using** запрос] [where набор условий]* ***- нет удаления top записей***

Отличия в языке

Получение изменённых записей

MS SQL

insert/update/delete таблица
Output deleted/inserted.столбец
Into [@/#]таблица
Values|From запрос

Postgres – *удобнее – не нужна промежуточная таблица*

insert/update /delete таблица
values()|from запрос|using запрос
returning *, столбец/столбцы
в *update* есть доступ только к *inserted*

MERGE – нет, но можно

with CTE(...)
as
(*insert/update/delete ...*
returning [столбец |столбцы]
)
insert/update/delete ...
*from [запрос с использованием **CTE**];*

Отличия в языке: Присваивание значений переменных

- **Переменная = константа/выражение/(*select* столбец … *limit 1*)/скалярная функция**
- ***Select* столбец1, столбец2 … *limit 1 into* переменная1, переменная1**

  Аналог в MS SQL

  Select @переменная1 = столбец1, @переменная1 = столбец2 *from*…

  Отличия:

  - **Если результата не будет, то значение переменных будет *null*** *(в MS SQL* –не измениться*)*
  - **Если запрос вернёт несколько строк – то будет ошибка => *limit 1***
  - **Нет поддержки конструкции конкатенации строкового столбца**

    Select @переменная = @переменная1+ столбец *from.*

  Вместо этого:

  - Массив привести к тексту

  ```
  [Переменная] = (select array(select столбец from …)::text);
  ```

  - Через цикл

Пакетные скрипты и особенности динамика

Пакетный скрипт – скрипт, состоящий из нескольких операторов.

- **Может вернуть результат только одного(последнего)** ***select***
- **Можно передать параметры из прикладного кода по имени [:параметр] номеру $N**
- **Не поддерживаются элементы логики –** ***переменные, if, while и т.д.***
  - **Нужно использовать блок анонимной функции.**

```
do language plpgsql $$
declare переменная тип [= значение];
begin
        if … then
        end if;
end;
$$;
```

- **Можно использовать в последовательности операторов в пакете**
- **Невозможно передать параметры из кода (.Net) – необходимо явно подставлять значения параметров в текст запроса**
- **Внутри блока можно использовать динамические запросы**

Отличия в работе хранимых процедур

MS SQL

- **Процедура - поименованный блок t-sql кода**: может принимать и возвращать параметры; имеет результат выполнения параметры; может дополнительно возвращать один или несколько наборов данных; исполняется самостоятельно через *exec[ute]*
- **Функции - блок t-sql кода, не изменяющий данных таблиц**: может принимать параметры; возвращать один результат, одно скалярное или одно табличное значение, и не поддерживают out параметры. Используется как часть запроса
- **Имя однозначно определяется по имени процедуры/функции**
- **Свободное объявление переменных**
- **Процедура/функция – просто набор команд работающих вне рамок транзакции** (пока явно этого не сделать)

Postgres

- **Имя определяется по имени функции + типизация параметров**
- **Поддерживаются только функции**, которые могут принимать параметры и возвращать только что-то одно
- **Нет ограничений по использованию операторов модификации данных**
- **Объявление переменных только в начале функции**
- **Работа функции является полностью транзакционной**

Отличия в работе хранимых процедур – возврат данных

```
create or replace function функция(параметр1 тип,
параметр2 тип, ...) returns [тип] as
$$
declare
begin
     [тело]
     return [тип];
end;
$$ language plpgsql volatile;
```

Результат	Конструкциия
Одно значение определённого типа	• В объявлении `returns` *`тип`* • В теле `return` *`переменная/выражение`*; • После `return` прекращается работы функции
Возврат набора переменных	• В объявлении `returns` *`record`* или *`параметр2`* `out` *`тип (тогда returns можно опустить)`* • В теле `return` [*`переменная типа record`*]; или *`параметр2 = …;`* [`return;`]
Возврат таблицы/столбца /набора значений как столбец	• В объявлении `returns` *`table(columns)`* или *`setof тип`* • В теле `return query` *`запрос соответствующий table`*; или `return next` *`переменная типа`*; • Работа функции прекращается только по явному `return;` или по окончанию функции • Просто `return;` - пустой табличный результат описанного типа • Удобно если `union all` – нет необходимости формировать весь запрос

Возврат таблицы/столбца /набора значений как столбец	• В объявлении `returns` *`setoff`* `refcursor`
	• В теле:
	`Declare переменная1 refcursor; переменная2 refcursor;`
	
	`Open переменная for select …;`
	`Return next переменная;`
	
	`Open переменная for execute format('select …');`
	`Return next переменная;`
	• дополнительные ограничения:
	○ Вызов функции и получение данных из *refcursor* должно происходить в единой транзакции
	○ Если *refcursor* ссылается на временную таблицу, то она должна существовать

`raise noitice` *'Return value: %s', код возврата;* - как возможность вернуть что то стандартизированное

Функция удаления процедуры

```
help_drop_proc(val_Name varchar(128)) returns integer as
$$
declare val_func_name varchar(128);
           val_shema_name varchar(128);
           val_fname text;
           val_pos integer;
           val_del_count integer;
begin
           val_shema_name = '';
           val_Name = replace(val_Name, '"', '');
           if val_Name ilike '%.%' then
                      val_pos = position('.' in val_Name);
                      if val_pos > 1 then
                                 val_shema_name = substring(val_Name, 1, val_pos-1);
                      end if;
                      if val_pos < char_length(val_Name) then
                                 val_func_name = substring(val_Name, val_pos+1, char_length(val_Name) - val_pos);
                      end if;
           else
                      val_func_name = val_Name;
           end if;
           val_del_count = 0;
           for val_fname in select oid::regprocedure from pg_proc where proname = val_func_name and (val_shema_name = '' or pronamespace =
(select oid from pg_namespace where nspname = val_shema_name limit 1)) loop
                      val_del_count = val_del_count + 1;
                      execute format('drop function %s cascade', val_fname);
           end loop;
           return val_del_count;
end;
$$
language plpgsql volatile;
```

Отличия в работе транзакций при ошибках

MS SQL

- **Ошибка в рамках транзакции не влияет на ее работу**, может перевести транзакцию в состояние невозможности COMMIT, либо разорвать соединение, что вызовет откат транзакции
- **Для обработки ошибки - нужно их явно обрабатывать**: откатывать/применять транзакцию, игнорировать ошибку и т.д.:
  - Анализ `if @@error > 0 begin . . . end`
  - `begin try` обёрнутый скрипт `end try begin catch` реакция на ошибку `end catch`

Postgres

- **Ошибка в рамках транзакции откатывает её работу**
- **Блок с обработки ошибок**:

```
begin
    код
    . . .
    exception
      then [OTHERS] then
    код
    . . .
end
```

Состояние сейчас

- **Происходит пилотное внедрение новой версии системы в Министерство промышленности и торговли**
- **Будущие участники : Минкомсвязи, ФНС, Пенсионный фонд РФ и Фонд социального страхования РФ**
- **Перевод на PG дополнительных сервисов СДУ «Приоритет»**

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