Карманный справочник: сравнение синтаксиса MS SQL Server и PostgreSQL

Приветствую, уважаемые хабражители!

Я занимаюсь переводом кода из MS SQL Server в PostgreSQL с начала 2019 года и сегодня  продолжу сравнение этих СУБД.

В [прошлой публикации](https://habr.com/ru/post/457602/) мы рассматривали отличия в быстродействии MS SQL Server и PostgreSQL для «1C».

В Ozon есть решения и на MS SQL Server, и на PostgreSQL: первая используется в логистике и системах внутренних сервисов, вторая — в mission critical-подсистемах, от которых напрямую зависит бизнес компании (склад, корзина, оплата картами, платежи, информация о товарах на сайте и др.).

Периодически появляются задачи перевода решений из огромных монолитных баз из MS SQL Server в PostgreSQL. Поэтому давайте сравним основные конструкции синтаксиса этих СУБД для правильного чтения кода, а также для того, чтобы быстро изменять код из MS SQL Server для PostgreSQL и наоборот.

Начнём с сопоставления типов.

Сопоставление типов

|  |  |
| --- | --- |
| MS SQL | PostgreSQL |
| BIGINT | BIGINT, INT8 |
| BINARY(n) | BYTEA |
| VARBINARY(n) | BYTEA |
| VARBINARY(max) | BYTEA |
| ROWVERSION | BYTEA |
| IMAGE | BYTEA |
| FIELDHIERARCHYID | BYTEA, LTREE ([расширение](https://www.postgresql.org/docs/11/ltree.html)) |
| BIT | BOOLEAN, BOOL |
| CHAR(n), n<=8000 | TEXT |
| NCHAR(n), n<=4000 | TEXT |
| VARCHAR(n), n<=8000 | TEXT |
| NVARCHAR(n), n<=4000 | TEXT |
| VARCHAR(max) | TEXT |
| NVARCHAR(max) | TEXT |
| TEXT | TEXT |
| NTEXT | TEXT |
| FLOAT(n) | DOUBLE PRECISION, FLOAT8 |
| SMALLMONEY | MONEY |
| MONEY | MONEY |
| INT, INTEGER | INT, INTEGER, INT4 |
| SMALLINT | SMALLINT, INT2 |
| NUMERIC(n,m) | NUMERIC(n,m) |
| DEC(n,m), DECIMAL(n,m) | DEC(n,m), DECIMAL(n,m) |
| TINYINT | SMALLINT, INT2 |
| REAL | REAL, FLOAT4 |
| UNIQUEIDENTIFIER | CHAR(16), UUID |
| DATE | DATE |
| TIME(n) | TIME(n) |
| DATETIME | TIMESTAMP(3) |
| DATETIME2(n) | TIMESTAMP(m) |
| DATETIMEOFFSET(n) | TIMESTAMP(n) WITH TIME ZONE, TIMESTAMPTZ |
| SMALLDATETIME | TIMESTAMP(0) |
| XML | XML |

*Примечание.* Типы CHAR и VARCHAR лучше не использовать. Причины подробно описаны [здесь](https://wiki.postgresql.org/wiki/Don%27t_Do_This).

Более подробно о типах данных:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/12/datatype)

Теперь перейдём к сопоставлению синтаксиса MS SQL Server и PostgreSQL.

Сопоставление синтаксиса MS SQL Server и PostgreSQL

**I. Регистрозависимое обращение к схемам, таблицам (представлениям) и их полям и другим объектам базы данных**

В MS SQL Server при обращениях к объектам можно использовать квадратные скобки (они обязательны, только если в названии объекта или его поля присутствуют недопустимые символы):

[schema]

[**table**]

[**view**]

[object]

[**table**].[field]

[**view**].[field]

[schema].[**table**]

[schema].[**view**]

[schema].[object]

[schema].[**table**].[field]

[schema].[**view**].[field]

В PostgreSQL для этого используются двойные кавычки (они обязательны, только если в названии объекта присутствуют заглавные буквы или есть недопустимые символы в названии объекта или его поля):

"schema"

"table"

"view"

"table"."field"

"view"."field"

"schema"."table"

"schema"."view"

"schema"."table"."field"

"schema"."view"."field"

**II. Выборка заданных N данных**

|  |  |
| --- | --- |
| В MS SQL Server используется [TOP](https://docs.microsoft.com/ru-ru/sql/t-sql/queries/top-transact-sql?view=sql-server-ver15): | В PostgreSQL используется [LIMIT](https://postgrespro.ru/docs/postgresql/14/queries-limit): |
| SELECT TOP(N) ...; | SELECT .... LIMIT N; |

**III. Постраничная загрузка данных (скользящее окно)**  
Задача: извлечь 100 строк начиная с 202-й строки включительно по возрастанию даты рождения:

|  |  |
| --- | --- |
| в [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/queries/select-order-by-clause-transact-sql?view=sql-server-ver15): | в [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-select): |
| SELECT \* FROM tbl ORDER BY BirthDate ASC OFFSET 201 ROW FETCH NEXT 100 ROWS ONLY; | select \* from tbl order by BirthDate asc [--offset 201 row fetch next 100 rows only;] LIMIT 100 OFFSET 200 |
| *Примечание.* Вместо row можно использовать rows в любом месте запроса, а вместо next можно использовать first в обеих СУБД. | |

**IV. Выборка первого непустого значения**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| [COALESCE](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/coalesce-transact-sql?view=sql-server-ver15) — рекомендуется  [ISNULL](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/isnull-transact-sql?view=sql-server-ver15) — не рекомендуется | [coalesce](https://postgrespro.ru/docs/postgresql/14/functions-conditional#FUNCTIONS-COALESCE-NVL-IFNULL) |

**V. Тернарный оператор IIF**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| [IIF](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/logical-functions-iif-transact-sql?view=sql-server-ver15) (<условие>,<выражение\_если\_условие\_истинно>,<выражение\_если\_условие\_ложно>)  или  [CASE](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/case-transact-sql?view=sql-server-ver15) WHEN <условие> THEN <выражение\_если\_условие\_истинно> ELSE <выражение\_если\_условие\_ложно> END | [case](https://postgrespro.ru/docs/postgresql/14/functions-conditional#FUNCTIONS-CASE) when <условие> then <выражение\_если\_условие\_истинно> else <выражение\_если\_условие\_ложно> end |

**VI. Создание псевдонима**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| AS — рекомендуется  = — не рекомендуется | as |

**VII. Выражения CASE**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| [CASE](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/case-transact-sql?view=sql-server-ver15)-WHEN-THEN-END  [CASE](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/case-transact-sql?view=sql-server-ver15)-WHEN-THEN-ELSE-END | [case](https://postgrespro.ru/docs/postgresql/14/functions-conditional#FUNCTIONS-CASE)-when-then-end  [case](https://postgrespro.ru/docs/postgresql/14/functions-conditional#FUNCTIONS-CASE)-when-then-else-end |

**VIII. Работа с переменными**

**Объявление переменной**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| DECLARE @val; | declare val; |
| *Примечание.* В MS SQL Server при объявлении переменных используется знак @ перед именем, а в PostgreSQL — нет. Также, помимо [PL/pgSQL](https://github.com/postgres/postgres/tree/c30f54ad732ca5c8762bb68bbe0f51de9137dd72/src/pl/plpgsql), в PostgreSQL можно встраивать и другие языки, такие как [PL/Python](https://github.com/postgres/postgres/tree/c30f54ad732ca5c8762bb68bbe0f51de9137dd72/src/pl/plpython) и [PL/Perl](https://github.com/postgres/postgres/tree/c30f54ad732ca5c8762bb68bbe0f51de9137dd72/src/pl/plperl). | |

**Присвоение переменной значения**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| SET @переменная = значение; | переменная := значение |
| *Примечание.* В PostgreSQL используется := для [PL/pgSQL](https://github.com/postgres/postgres/tree/c30f54ad732ca5c8762bb68bbe0f51de9137dd72/src/pl/plpgsql) и просто = для [PL/Python](https://github.com/postgres/postgres/tree/c30f54ad732ca5c8762bb68bbe0f51de9137dd72/src/pl/plpython) и [PL/Perl](https://github.com/postgres/postgres/tree/c30f54ad732ca5c8762bb68bbe0f51de9137dd72/src/pl/plperl). | |

**Вывод значения на консоль**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| print 'строка';  print @переменная;  RAISERROR(@переменная, 1, 1) WITH NOWAIT; | RAISE NOTICE '%', 'строка';  RAISE NOTICE '%', <переменная>; |

**IX. Управление выполнением кода**

**Выполнение скрипта**

В MS SQL Server:

**declare** @\_query int;

**set** @\_query=777;

**set** @query=1+8;

RAISERROR(@\_query, 1, 1) **WITH** NOWAIT; --PRINT @\_query;

В PostgreSQL:

Шаблон:

do $$

<объявление переменных>

**begin**

<код>

**end**;

$$;

Пример (вывод информации):

do $$

**declare** \_query int;

**begin**

\_query:=777;

\_query:=1+8;

RAISE NOTICE '%', \_query;

**end**;

$$;

Пример (передача значения клиенту):

do $$

**declare** \_query int;

**begin**

\_query:=777;

\_query:=1+8;

PERFORM

set\_config('my.\_query',

\_query::text, FALSE);

**end**;

$$;

**SELECT** current\_setting

('my.\_query');

Для PostgreSQL:

1. В DBeaver (бобре) нужно нажать CTRL+SHIFT+O при отсутствии окна вывода, а в pgAdmin вывод происходит автоматически.
2. В psql и так всё работает.

**Цикл WHILE**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| WHILE <условие\_при\_котором\_цикл\_работает>  BEGIN  ...  END | while <условие\_при\_котором\_цикл\_работает>  loop  ...  end loop |

**Логическое ветвление**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| IF-BEGIN-END  IF-BEGIN-END-ELSE-BEGIN-END | if-then-else-end if;  if-then-elseif-then-else-end if; |

Более подробно про управление выполнением кода:

1. [Управление выполнением кода в MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/language-elements/control-of-flow?view=sql-server-ver15)
2. [Управляющие структуры в PostgreSQL](https://postgrespro.ru/docs/postgresql/14/plpgsql-control-structures)

**X. Функции для работы со строками**

**Определение длины строки (количество символов в строке)**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| [LEN](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/len-transact-sql?view=sql-server-ver15) (<строка>) | [length](https://postgrespro.ru/docs/postgresql/14/functions-string) (<строка>) |
| *Примечание.* В MS SQL Server исключаются конечные пробелы. Если нужно учитывать и их, то необходимо воспользоваться функцией [DATALENGTH](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/datalength-transact-sql?view=sql-server-ver15) (<строка>), которая возвращает суммарное количество байтов в символах строки. | |

**Возвращение символа по его коду:**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| char(n) | chr(n) |

**Конкатенация строк**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| + | || |

**Нахождение позиции вхождения подстроки**

|  |  |
| --- | --- |
| В MS SQL Server: | В PostgreSQL: |
| CHARINDEX(<что\_ищем>, <где\_ищем>,<с\_какой\_позиции\_ищем\_начиная\_с\_1>) | strpos(<где\_ищем>, <что\_ищем>)  strpos(substring(<где\_ищем>, <с\_какой\_позиции\_ищем\_начиная\_с\_1>, length(<где\_ищем>)- <с\_какой\_позиции\_ищем\_начиная\_с\_1>+1), <что\_ищем>) |
| *Примечание.* Точного соответствия не будет, если производить поиск не с начала строки. | |

**Регистронезависимое сравнение и поиск данных**

|  |  |
| --- | --- |
| В MS SQL Server: | В PostgreSQL: |
| 1. LIKE  2. a = b  3. <>  4. a in (b1, ...) | 1. ilike  2. lower(a) = lower(b) или upper(a)=upper(b)  3. lower(a) <> lower(b) или upper(a)<>upper(b)  4. lower(a) in (lower(b1), ...) или upper(a) in (upper(b1), ...) |

*Примечание.* В PostgreSQL рекомендуется произвести оптимизацию через создание функционального индекса:

**create** [concurrently] index idx\_lower\_<field>

**on** <schema>.<**table**> (lower(<field>));

--После создания concurrently-индекса,

--его необходимо проверить на наличие битых индексов следующим запросом:

**SELECT** indexrelid::regclass **FROM** pg\_index **where** **not** indisvalid;

--Далее для обновления статистики по нужной таблице

--необходимо выполнить команду ANALYZE:

ANALYZE <**table**>;

Более подробно про команду [ANALYZE](https://postgrespro.ru/docs/postgresql/14/sql-analyze).

**Слияние строк по запросу в одну строку по заданному разделителю**

В MS SQL Server можно использовать функцию [STUFF](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/stuff-transact-sql?view=sql-server-ver15) следующим образом:

STUFF(( **SELECT** **DISTINCT**

', ' + **CONVERT**(varchar, tbl.<поле>)

**FROM**

<схема>.<таблица> tbl

[**WHERE**

<условия>]

**FOR** XML PATH(''))

, 1

, 1

, '') **AS** STUFF\_tbl;

Также начиная с версии 2017 доступна функция [STRING\_AGG](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/string-agg-transact-sql?view=sql-server-ver15).

В PostgreSQL для этого можно использовать функцию [string\_agg](https://www.postgresqltutorial.com/postgresql-aggregate-functions/postgresql-string_agg-function/) таким образом:

string\_agg((**SELECT**

**distinct** ', ' ||

cast(tbl.<поле> **as** VARCHAR) **FROM**

<схема>.<таблица> tbl,

[**WHERE**

<условия>]

), 1, 1,

'') **AS**

string\_agg\_field;

Более подробно про функции для работы со строками:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/string-functions-transact-sql?view=sql-server-ver15)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/functions-string)

**XI. Функции для работы с датой и временем**

**Получение текущей даты и времени (локальное время)**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| GetDate()  SysDateTime()  current\_timestamp | clock\_timestamp  now() |

**Получение текущей даты**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| CAST(GetDate() as DATE) | current\_date |

Пример преобразования формата даты и времени из строки public\_date:

В MS SQL Server:

FORMAT(public\_date, 'dd.MM.yyyy HH:mm:ss', 'ru-RU') — предпочтительный способ

**convert**(varchar(32),**convert**(datetime,public\_date,104),120)

В PostgreSQL:

to\_char(to\_timestamp(public\_date, 'dd.MM.yyyy hh24.mi'), 'yyyy-mm-dd hh24:mi:ss')

**Приращение даты/времени**

|  |  |
| --- | --- |
| В MS SQL Server: | В PostgreSQL: |
| DateAdd(datepart, count, dt); | dt + (count \* interval '1 datepart'); или dt + interval 'count datepart'; |
| Более подробно про функции для работы с датой и временем:  1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/date-and-time-data-types-and-functions-transact-sql?view=sql-server-ver15)  2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/functions-datetime) | |

**XII. Получение количества строк, затронутых при выполнении последней команды**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| [@@ROWCOUNT](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/rowcount-transact-sql?view=sql-server-ver15) | [get diagnostics](https://postgrespro.ru/docs/postgresql/14/plpgsql-statements) <переменная>:=row\_count; |

**XIII. Выполнение динамического SQL-кода**

|  |  |
| --- | --- |
| MS SQL Server | PostgreSQL |
| execute [sp\_executesql](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-stored-procedures/sp-executesql-transact-sql?view=sql-server-ver15) @sql; | [execute](https://postgrespro.ru/docs/postgresql/14/sql-execute) \_sql; |

**XIV. Проверка и приведение типов**

**Проверка строки на то, что она является числом**

В MS SQL Server:

встроенная функция [isnumeric](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/isnumeric-transact-sql?view=sql-server-ver15)(val)

В PostgreSQL:

**CREATE** **OR** REPLACE

**FUNCTION**

dbo.isnumeric(\_input varchar(255) **DEFAULT** **NULL**::varchar(255))

**RETURNS** bit

**LANGUAGE** plpgsql

**AS** $**function**$

/\*

Проверяет,

является ли входная строка

числом

\*/

**declare** \_result bit;

**begin**

**begin**

perform

\_input::numeric;

\_result:=1::bit;

exception

**when** others **THEN**

\_result:=0::bit;

**end**;

**return**

\_result;

**end**;

$**function**$

;

**Безопасное приведение типа**

В MS SQL Server:

try\_cast(val **as** <type>)

*Примечание.* try\_cast в MS SQL Server возвращает NULL, если значение невозможно привести к заданному типу, в других случаях — работает как оператор CAST.

В PostgreSQL есть два способа:

1) через обработку ошибок:

**declare** \_result оператор CAST <type>;

...

**BEGIN**

\_result :=

cast(val **as** <type>);

exception

**when** others **then**

\_result :=**null**;

**end**;

2) через реализацию функции:

**CREATE** **OR** REPLACE

**FUNCTION**

dbo.try\_cast(**value** character varying, typename CHARACTER varying)

**returns** text

**LANGUAGE** plpgsql

**AS** $**function**$

**declare** \_sql\_command text;

**DECLARE** \_result text;

**begin**

\_result=**value**;

\_sql\_command :=

'select

cast('||''''||

**value**||''''||' as '||

typename||');';

**BEGIN**

**execute** \_sql\_command;

exception

**when** others **then**

\_result :=**null**;

**end**;

**return**

\_result;

**end**;

$**function**$

;

Функция в итоге не возвращает преобразованное в заданный тип значение.  
Функция на вход принимает текст и возвращает текст.  
Если значение невозможно привести к заданному типу, то возвращается NULL.

Пример использования (чтобы было как в MS SQL Server):

cast(dbo.try\_cast(val::text, '<type>') **as** <type>)

**XV. DML-команды**

**Обновление данных**

Пример в MS SQL Server:

Обновление поля Name в таблице Production.ScrapReason для тех строк, для которых есть соответствующие записи в таблице Production.WorkOrder по равенству ScrapReasonID и у которых значение ScrappedQty больше 300:

UPDATE

sr

**SET**

sr.Name = 'Name'

OUTPUT

deleted.\*

, inserted.\*

**FROM** Production.ScrapReas sr

**JOIN** Production.WorkOrder wo **ON** (sr.ScrapReasonID = wo.ScrapReasonID)

**AND** (wo.ScrappedQty > 300);

Ключевое слово OUTPUT позволяет получить данные об обновлении.

Пример в PostgreSQL:

Обновление поля Name в таблице production.scrapreason для тех строк, для которых есть соответствующие записи в таблице production.workorder по равенству scrapreasonid и у которых значение scrappedqty больше 300:

update

production.scrapreason **as** sr

**set** sr.Name = 'Name'

**from** production.workorder **as** wo

**where** (sr.scrapreasoid = wo.scrapreasonoid)

**and** (wo.scrappedqty > 300)

returning \*;

Ключевое слово returning позволяет получить данные об обновлении.

Более подробно о команде UPDATE:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/queries/update-transact-sql?view=sql-server-ver15)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-update)

**Удаление данных**

Пример в MS SQL Server:

Удаление из таблицы Sales.SalesPersonQuotaHistory тех записей, для которых есть соответствующие записи в таблице Sales.SalesPerson по равенству BusinessEntityID и у которых значение SalesYTD больше 2500000.00:

**DELETE** **FROM**

spqh

OUTPUT deleted.\*

**FROM** Sales.SalesPersonQuotaHistory spqh

**INNER** **JOIN** Sales.SalesPerson sp **ON** (spqh.BusinessEntityID = sp.BusinessEntityID)

**WHERE**

(sp.SalesYTD > 2500000.00);

Ключевое слово OUTPUT позволяет получить данные об удалении.

Пример в PostgreSQL:

Удаление из таблицы sales.salespersonquotahistory тех записей, для которых есть соответствующие записи в таблице sales.salesperson по равенству businessentitid и у которых значение salesytd больше 2500000.00:

**delete** **from**

sales.salespersonquotahistory **AS** spqh

**using**

sales.salesperson **AS** sp

**where**

(spqh.businessentityid = sp.businessentitid)

**and** (sp.salesytd > 2500000.00)

returning \*;

Ключевое слово returning позволяет получить данные об удалении.

Более подробно о команде DELETE:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/delete-transact-sql?view=sql-server-ver15)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-delete)

**Получение изменённых записей**

|  |  |
| --- | --- |
| В MS SQL Server: | В PostgreSQL: |
| insert/update/delete таблица Output deleted/inserted.<столбец> into [@/#] <таблица> Values|From <запрос> | insert/update/delete таблица values()|from <запрос>|using <запрос> returning \*, столбец/столбцы |
| В update есть доступ только к inserted.  *Примечание.* В PostgreSQL не нужна промежуточная таблица для получения изменённых записей.  Более подробно:  1. [OUTPUT в MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/queries/output-clause-transact-sql?view=sql-server-ver15)  2. [returning в PostgreSQL](https://postgrespro.ru/docs/postgresql/14/dml-returning) | |

**Удаление дубликатов (дублирующих строк):**

В MS SQL Server:

**with** dbl\_in\_stage **as** (

**select**

row\_number() **over**

(**partition** **by**

<field\_1>, ...,

<field\_N> **order** **by**

1) **as** rn

**from**

<схема>.<таблица>

**as** stg

)

**delete** **from**

dbl\_in\_stage **where**

rn > 1;

В PostgreSQL:

**with** x **as** (

**select** a, ctid, row\_number() **over**(**partition** **by** a **order** **by** ctid) rn

**from** t

)

**delete** **from** t

**using** x

**where** t.a = x.a

**and** t.ctid = x.ctid

**and** x.rn > 1;

или более сложный вариант:

**delete** **from** <схема>.<таблица>

**where** ctid=**any**(

**array**(**select** unnest(ctids[2:])

**from** (

**select** array\_agg(

ctid **order** **by** string\_to\_array(

regexp\_replace(ctid::text, E'\\(|\\)','','g'),',')::bigint[]) ctids

**FROM** <схема>.<таблица> **as** T **group** **by** T::text) **as** T)::tid[]);

*Примечание.* Оптимальный вариант — внести в таблицу уникальный ключ, так как работа с метаданными увеличивает нагрузку на систему.

При наличии уникального ключа удалять дубликаты в PostgreSQL можно следующим образом:

**delete** **from** <схема>.<таблица> **where** <уникальный ключ> **in**

(**select** <уникальный ключ>

**from** (

**select** \*, row\_number() **over** (**partition** **by** <field\_1>, ..., <field\_N> **order** **by** 1) **as** rn

**from** <схема>.<таблица>

) **as** tbl **where** rn > 1);

**XVI. DDL-команды для работы с таблицами**

**Удаление таблицы с предварительной проверкой**

В MS SQL Server:

Для основной таблицы:

**DROP** **TABLE** IF **EXISTS** <schema>.<**table**>;

Для локальной временной таблицы:

IF **EXISTS**(**SELECT** [name] **FROM** tempdb.sys.tables **WHERE** [name] **like** '#<table>%')

**BEGIN**

**DROP** **TABLE** #<**table**>;

**END**;

Для глобальной временной таблицы:

IF **EXISTS**(**SELECT** [name] **FROM** tempdb.sys.tables **WHERE** [name] **like** '##<table>%')

**BEGIN**

**DROP** **TABLE** ##<**table**>;

**END**;

Здесь:

1. #<table> — локальная временная таблица, которая видна только в текущей сессии
2. ##<table> — глобальная временная таблица, которая видна всем пока она существует

Все временные таблицы живут, либо пока активна сессия, во время которой они были созданы, либо пока их явно не удалят.

В PostgreSQL:

Для основной таблицы:

**drop** **table** if **exists**

<schema>.<**table**>;

Для временной таблицы:

**drop** **table** if **exists** <**table**>;

Более детально про удаление таблиц:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/drop-table-transact-sql?view=sql-server-ver15)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-droptable)

**Создание таблицы через выборку**

В MS SQL Server:

Для основной таблицы:

**select** ...

**into** <**table**>

**from** …

Для временной таблицы:

**select** ...

**into** #<**table**>

**from** …

В PostgreSQL:

Для основной таблицы:

**create** **table** <**table**> **as**

**select** ...

Для временной таблицы:

**create** temp **table** <**table**> **as**

**select** …

Более детально про создание таблиц через выборку:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/queries/select-into-clause-transact-sql?view=sql-server-ver15)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-createtableas)

**Создание/изменение и удаление значения по умолчанию для колонки таблицы**

В MS SQL Server:

Добавление:

**ALTER** **TABLE**

<схема>.<таблица>

**ADD** **CONSTRAINT**

<название\_правила>

**DEFAULT**

<значение\_по\_умолчанию> **FOR** <поле>;

Выборка всех значений по умолчанию:

**SELECT** SCHEMA\_NAME(t.[schema\_id]) **AS** sch

, t.name **AS** tbl

, col.name **AS** colname

, dc.definition **AS** def

**FROM** sys.default\_constraints dc

**INNER** **JOIN** sys.columns col **ON** dc.parent\_object\_id = col.[object\_id]

**INNER** **JOIN** sys.tables t **ON** t.[object\_id] = col.[object\_id];

Удаление:

**DROP** **DEFAULT** IF **EXISTS** <название\_правила>;

Изменение происходит через удаление и добавление.

В PostgreSQL:

Создание и изменение:

**alter** **table**

<схема>.<таблица>

**alter** **column** <поле>

**set** **default**

<значение\_по\_умолчанию>;

Выборка всех значений по умолчанию:

**select**

col.table\_schema,

col.table\_name,

col.column\_name,

col.column\_default

**from**

information\_schema.columns **as** col;

Удаление:

**alter** **table** <схема>.<таблица>

**alter** **column** <поле> **drop** **default**;

**Изменение типа колонки таблицы**

|  |  |
| --- | --- |
| В MS SQL Server: | В PostgreSQL: |
| ALTER TABLE <схема>.<таблица> ALTER COLUMN <поле> <новый\_тип> [NULL|NOT NULL]; | alter table <схема>.<таблица> alter column <поле> type <новый\_тип>; |
| *Примечание.* Если у изменяемого столбца есть какие-либо констрейнты или значения по умолчанию, то сначала нужно их сохранить, а затем удалить. Вернуть их можно будет после изменения типа. | |

**Перенос автоинкрементных полей**

В MS SQL Server делаем запрос вида:

**SELECT**

'do $$

declare

start\_with\_val

bigint;

declare

sql\_statement

varchar;

begin

start\_with\_val :=

coalesce((select

max(' + c.[name] + ') from

' + s.[name] + '.' + o.[name] + '),0)+1;

sql\_statement :=

''alter table

' + s.[name] + '.' + o.[name] + ' alter ' + c.[name] + '

add generated by

default as identity

(start with ''

||

cast(start\_with\_val

as varchar)||'');'';

execute

sql\_statement;

end;

$$;' **AS** plsql\_statement

--select distinct s.name

**FROM**

sys.all\_columns c

**INNER** **JOIN** sys.all\_objects o **ON** o.[object\_id] = c.[object\_id]

**INNER** **JOIN** sys.schemas s **ON** s.[schema\_id] = o.[schema\_id]

**WHERE**

is\_identity <> 0

**AND** SCHEMA\_NAME(o.[schema\_id]) <> 'sys'

**AND** o.[type] = 'U';

Пример:

do $$

**declare** start\_with\_val bigint;

**declare** sql\_statement varchar;

**begin**

start\_with\_val := coalesce((**select** max(ID) **from** dbo.ExchangeQueue),0)+1;

sql\_statement := 'alter table dbo.ExchangeQueue alter ID add generated by default as identity (start with ' || cast(start\_with\_val **as** varchar)||');';

**EXECUTE** sql\_statement;

**end**;

$$;

Полученные скрипты применяем на стороне PostgreSQL.

**Создание автоинкрементных полей**

В MS SQL Server:

**ALTER** **TABLE** [схема].[таблица]

**ADD**

<**IDENTITY**-поле> bigint **IDENTITY**(1, 1) **NOT** **NULL**;

В PostgreSQL:

do $$

**DECLARE** start\_with\_val bigint;

**DECLARE** sql\_statement varchar;

**BEGIN** start\_with\_val := coalesce((**select** max(<**IDENTITY**-поле>) **from** <схема>.<таблица>),0)+1;

sql\_statement := 'alter table <схема>.<таблица> alter <IDENTITY-поле> add generated by default as identity (start with ' || cast(start\_with\_val **as** varchar)||');';

**EXECUTE** sql\_statement;

**END**;

$$;

|  |  |
| --- | --- |
| Более детально про создание таблиц: | |
| [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver15) | [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-createtable) |
| Более детально про изменение таблиц: | |
| [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver15) | [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-altertable) |

**XVII. Создание и изменение представления**

|  |  |
| --- | --- |
| В MS SQL Server: | В PostgreSQL: |
| CREATE OR ALTER VIEW  [схема].[название\_представления]  AS … | create or replace view  <схема>.<название\_представления>  as … |
| *Примечание.* В PostgreSQL лучше сначала удалять представление, а потом заново его создавать, если набор полей меняется или меняются названия выходных полей, иначе можно получить ошибку при обращении к изменённому представлению. | |
| Более подробно про создание и изменение представлений: | |
| [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-view-transact-sql?view=sql-server-ver15) | [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-createview) |

**XVIII. Построчная обработка строк в наборе**

В MS SQL Server:

--объявление переменных @field\_1, ...@field\_N

**DECLARE** <курсор>

**CURSOR** **LOCAL** **FOR**

<**SELECT**>;

**OPEN** <курсор>;

**FETCH** NEXT **FROM**

<курсор> **INTO**

@field\_1, ...@field\_N;

WHILE

(@@FETCH\_STATUS = 0)

**BEGIN**

--оперируем значениями переменных @field\_1, ...@field\_N

...

**FETCH** NEXT

**FROM** <курсор> **INTO** @field\_1, ...@field\_N;

**END**

**CLOSE** <курсор>;

**DEALLOCATE** <курсор>;

В PostgreSQL:

do $$

**declare** \_val

record;

**begin**

**drop** **table** if

**exists** \_tmp\_tbl;

**create** temp

**table** \_tmp\_tbl **as**

<**select**>

**for** \_val **in**

(**select** field\_1, ..., field\_n from\_tmp\_tbl)

loop

--можно обратиться к любому выбранному ранее полю через \_val.<поле>. Например, \_val.<field\_1>

**end** loop;

**end**

$$

**XIX. Системные информационные функции безопасности**

**Текущий пользователь**

В MS SQL Server используется функция [CURRENT\_USER](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/current-user-transact-sql?view=sql-server-ver15)().

В PostgreSQL:

1. session\_user — под каким пользователем открыта сессия
2. current\_user (или просто user) — под каким контекстом (ролью) идёт выполнение (session\_user переключается для выполнения — здесь важно, под каким правом делается переключение)

**Получение имени экземпляра и IP-адреса сервера СУБД**

В MS SQL Server:

Получить информацию об IP-адресе сервера СУБД:

**SELECT**

CONNECTIONPROPERTY('

net\_transport') **AS** net\_transport

, CONNECTIONPROPERTY('

protocol\_type') **AS** protocol\_type

, CONNECTIONPROPERTY('

auth\_scheme') **AS** auth\_scheme

, CONNECTIONPROPERTY('

local\_net\_address') **AS** local\_net\_address

, CONNECTIONPROPERTY('

local\_tcp\_port') **AS** local\_tcp\_port

, CONNECTIONPROPERTY('

client\_net\_address') **AS** client\_net\_address;

Получить название экземпляра СУБД:

**SELECT** @@SERVERNAME;

В PostgreSQL:

Получить IP-адрес сервера СУБД:

do $$

**declare**

title varchar(100) :=host(inet\_server\_addr());

**begin**

raise notice '%',

title;

**end**; $$;

Получение названия экземпляра СУБД пока не реализовано.

Более подробно про системные информационные функции безопасности:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/functions/security-functions-transact-sql?view=sql-server-ver15)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/functions-info)

**XX. Определение и вызов хранимой процедуры**

**Определение хранимой процедуры**

В [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-ver15):

**CREATE** **OR** **ALTER** **PROCEDURE**

[схема].[назание\_процедуры]

<переменная\_1> <тип\_1>[=<значение\_по\_умолчанию\_1>],

...

**AS**

**BEGIN**

...

**END**

В [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-createprocedure):

**CREATE** **OR** REPLACE **PROCEDURE**

<схема>.<название\_процедуры>

(

[**INOUT**] <переменная\_1> <тип\_1>[=<значение\_по\_умолчанию1>],

...

)

**LANGUAGE** plpgsql

**AS** $body$

[<Объявление переменных>]

**BEGIN**

...

**END**;

$body$

;

**Вызов хранимой процедуры**

В MS SQL Server:

**EXEC** <схема>.<процедура>

<переменная\_1>=<значение\_1>, ..., <переменная\_OUTPUT> **OUT**[PUT];

В PostgreSQL:

**call** <схема>.<процедура> (

<переменная\_1>=<значение\_1>, ..., <переменная\_OUTPUT>);

**XXI. Создание скалярной функции**

В [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/t-sql/statements/create-function-transact-sql?view=sql-server-ver15):

**CREATE** **OR** **ALTER**

**FUNCTION** [схема].[название\_функции]

(<параметр\_1> <тип\_1>[=<значение\_по\_умолчанию\_1>], ...)

**RETURNS** <возвращаемый\_тип> **AS**

**BEGIN**

... **RETURN** ...

**END**

В [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/sql-createfunction):

**CREATE** **OR** REPLACE **FUNCTION**

<схема>.<название\_функции>

(<параметр\_1> <тип\_1>[=<значение\_по\_умолчанию\_1>], ...)

**RETURNS** <возвращаемый\_тип>

**LANGUAGE** plpgsql

**AS** $body$

[<Объявление переменных>]

**begin**

... **return** (

**select** ...

);

**end**;

$body$

;

**XXII. Передача табличного значения (вывод таблицы)**

В MS SQL Server:

**CREATE** **OR** **ALTER**

**PROCEDURE**

[схема].[название\_хранимой\_процедуры]

<параметр\_1> <тип\_1>,

...,

<параметр\_N> <тип\_N>

**AS**

**BEGIN**

...

**SELECT** ...

**END**

В PostgreSQL:

**create** **or** replace **function**

<схема>.<название\_функции>

(<параметр\_1> <тип\_1>, ..., <параметр\_N> <тип\_N>)

**return** **table** (<поле\_1> <тип\_1>, ..., <поле\_N> <тип\_N>)

**language** 'plpgsql'

**as** $body$

[<Объявление переменных>]

**begin**

**return** query (**select** ....);

**end**;

$body$;

**XXIII. DML-триггеры**

Пример в MS SQL Server:

**CREATE** **TRIGGER** [info].[tr\_isupoll\_question\_text\_last\_update\_trigger]

**ON** [info].[isupoll\_question\_text]

**FOR** UPDATE

**AS**

UPDATE

info.isupoll\_question\_text

**SET**

last\_update\_date = GETDATE()

, last\_update\_user = SUSER\_NAME()

**FROM**

info.isupoll\_question\_text ds

**INNER** **JOIN** INSERTED i **ON**

ds.isupoll\_question\_text\_id = i.isupoll\_question\_text\_id;

Здесь создаётся триггер tr\_isupoll\_question\_text\_last\_update\_trigger для таблицы  info.isupoll\_question\_text после обновления данных, который для обновляемых строк проставляет текущие дату, время и пользователя соответственно.

**DROP** **TRIGGER** IF **EXISTS** [tr\_isupoll\_question\_text\_last\_update\_trigger]

**on** [info].[isupoll\_question\_text];

Здесь удаляется триггер tr\_isupoll\_question\_text\_last\_update\_trigger для таблицы info.isupoll\_question\_text

Пример в PostgreSQL:

**CREATE** **OR** REPLACE

**FUNCTION** dbo.update\_mod()

**RETURNS** **trigger**

**LANGUAGE** plpgsql

**AS** $**function**$

**begin**

new.last\_update\_date=now();

new.last\_update\_user=session\_user;

**return** **new**;

**end**;

$**function**$

;

Здесь создаётся функция dbo.update\_mod(), которая заполняет два поля текущими датой, временем и пользователем соответственно.

**create** **trigger** tr\_isupoll\_question\_text\_last\_update\_trigger before update

**on** info.isupoll\_question\_text **for** **each** row **execute** **function** dbo.update\_mod();

Здесь создаётся триггер tr\_isupoll\_question\_text\_last\_update\_trigger для таблицы  info.isupoll\_question\_text до обновления данных, который для каждой строки вызывает выполнение функции dbo.update\_mod().

**drop** **trigger** if **exists** tr\_isupoll\_question\_text\_last\_update\_trigger

**on** info.isupoll\_question\_text;

Здесь удаляется триггер tr\_isupoll\_question\_text\_last\_update\_trigger для таблицы info.isupoll\_question\_text.

**Важно!** В триггере используйте ключевое слово **before,**когда хотите нашкодничать в той же таблице, для которой создаётся триггер, и **after**— для логирования в другую таблицу.

Более подробно про DML-триггеры:

1. [MS SQL Server](https://docs.microsoft.com/ru-ru/sql/relational-databases/triggers/dml-triggers?view=sql-server-2017)
2. [PostgreSQL](https://postgrespro.ru/docs/postgresql/14/triggers)

И в качестве бонуса кратко рассмотрим сопоставление основных системных представлений и приведём ссылки для мониторинга.

Немного о сопоставлении системных представлений и мониторинге

Сопоставление системных представлений

|  |  |  |
| --- | --- | --- |
| **MS SQL Server** | **PostgreSQL** | **Описание** |
| [Представления схемы системных сведений](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-information-schema-views/system-information-schema-views-transact-sql?view=sql-server-ver15) | [System Views](https://www.postgresql.org/docs/14/views-overview.html) |  |
| [sys.dm\_exec\_query\_stats](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-query-stats-transact-sql?view=sql-server-ver15) | [pg\_stat\_statements](https://postgrespro.ru/docs/postgresql/14/pgstatstatements) | Предоставляет статистику по выполненным запросам.  В MS SQL Server содержит только то, что в кеше, а в PostgreSQL — всю статистику.  Для PostgreSQL:  CREATE EXTENSION pg\_stat\_statements;  на каждую БД. |
| [sys.dm\_exec\_function\_stats](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-function-stats-transact-sql?view=sql-server-ver15) | [pg\_stat\_user\_functions](https://postgrespro.ru/docs/postgresql/14/monitoring-stats#MONITORING-PG-STAT-USER-FUNCTIONS-VIEW) | Предоставляет статистику по вызовам пользовательских функций. |
| [sys.dm\_db\_index\_usage\_stats](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-dynamic-management-views/sys-dm-db-index-usage-stats-transact-sql?view=sql-server-ver15) | [pg\_stat\_all\_indexes](https://postgrespro.ru/docs/postgresql/14/monitoring-stats)  [pg\_stat\_user\_indexes](https://postgrespro.ru/docs/postgresql/14/monitoring-stats) | Предоставляет статистику по использованию всех пользовательских индексов. |
| [sys.master\_files](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-catalog-views/sys-master-files-transact-sql?view=sql-server-ver15)  [sys.fn\_virtualfilestats (NULL, NULL)](https://docs.microsoft.com/ru-ru/sql/relational-databases/system-functions/sys-fn-virtualfilestats-transact-sql?view=sql-server-ver15) | [pg\_stat\_database](https://postgrespro.ru/docs/postgresql/14/monitoring-stats#MONITORING-PG-STAT-DATABASE-VIEW) | Предоставляет статистические данные по каждой БД. |