3

Секционирование таблиц

Ограничения PostgreSQL

- 32Тб - максимальный размер:
  - несекционированной таблицы
  - секции секционированной таблицы
  - TOAST-таблицы
- ключевые столбцы типа `int/serial` хранят до 2 млрд. положительных значений
- при размере строки больше 2032 байта её поля выносятся в TOAST
- число полей, которые могут храниться в TOAST - 4 млрд.
- ограничения на размеры таблиц, TOAST-таблиц, число вынесенных полей, могут достигаться независимо друг от друга

Ограничения PostgreSQL

Максимальный размер файлов с данными у таблицы - 32Тб для блока размером 8Кб. Строка хранится в блоке, если полная (с заголовком) длина строки в таблице меньше 2032 байта. Если длина строки больше, то её поля сжимаются и вытесняются в TOST-таблицу. В строках TOAST-таблицы, поле хранится порезанным на части (чанки). Размер чанка 1996 байт. Один чанк- одна строка TOAST-таблицы.

```
pg_controldata | grep TOAST
Maximum size of a TOAST chunk:        1996
```

В таблице остаётся 18 байт, в которых хранится указатель на первый чанк.

В TOAST-таблицу может быть вытеснено до 4млрд. полей (даже не строк). Если ограничение будет достигнуто, то оно ограничит число строк в таблице.

Также, размер самой TOAST-таблицы не может превышать 32Тб. 4 миллиарда полей займут 32Тб при размере поля ~8000 байт (в сжатом виде). Если размер поля 8Мб (или в строке 10 полей по 8000 байт), то в TOAST может быть вынесено 4 миллиона полей, что не так много. Это ограничит число строк в таблице - в таблицу нельзя вставить больше 4 миллионов строк. Если точнее, то не строк, а версий строк: если значения вынесенных полей обновляются, то каждая версия строки будет ссылаться на образ своего поля, вынесенного в TOAST. Старые версии полей будут храниться в TOAST до обработки автовакуумом.

При проектировании схем хранения стоит учитывать, что ограничения на размеры таблиц, TOAST-таблиц, число вынесенных полей, могут быть достигнуты независимо друг от друга.

Для обхода ограничений можно удалять старые данные, создавать новые таблицы, использовать секционированные таблицы. Для секционированных таблиц ограничения действуют на каждую секцию.

Число полей, вынесенных в TOAST

- Размер TOAST-таблицы и её название:

```
select reltoastrelid::regclass name, relpages from pg_class where
reltoastrelid::regclass::text like 'pg_toast%' and relname='pg_rewrite';
          name          | relpages
------------------------+----------
 pg_toast.pg_toast_2618 |       15
select count(distinct chunk_id) from pg_toast.pg_toast_2618;
    82
```

- Подсчет числа полей, вынесенных в TOAST-таблицы:

```
psql -qtAc "select 'select '''||a||''', count(distinct chunk_id) from
'||a||' having count(distinct chunk_id)>0;' from (select
reltoastrelid::regclass a from pg_class where
reltoastrelid::regclass::text like 'pg_toast%')" | psql -qt | uniq | sort
 pg_toast.pg_toast_1255 |     2
 pg_toast.pg_toast_2618 |    82
 pg_toast.pg_toast_2619 |     8
```


Число полей, вынесенных в TOAST

При администрировании, можно проверять: размеры таблиц не должны приближаться к 32Тб, число строк в таблицах, у которых тип уникального ключа `int` (а не `bigint`) к 2 млрд (обычно, заполняются последовательностями с положительными числами), `count(chunk_id)` TOAST-таблицы к 4млрд. Если такие таблицы есть, то, возможно, стоит удалить старые данные или запланировать их секционирование или сообщить разработчикам приложения, что возможно достижение ограничений PostgreSQL. Пример подсчёта:

```
psql -qtAc "select 'select '''||a||''', count(distinct chunk_id) from '||a||'
having count(distinct chunk_id)>0;' from (select reltoastrelid::regclass a
from pg_class where reltoastrelid::regclass::text like 'pg_toast%')" | psql -
qt | uniq | sort
```

Без указания `distinct` подсчитывается число чанков, а не полей, вынесенных в TOAST.

Формируемые командой запросы читают только листовые блоки TOAST-индексов:

```
explain (analyze) select count(distinct chunk_id) from pg_toast.pg_toast_2618
having count(distinct chunk_id)>0;
                        QUERY PLAN
-----------------------------------------

 Aggregate  (cost=13.03..13.04 rows=1 width=8) (actual time=2.365..2.381
rows=1.00 loops=1)
    Filter: (count(DISTINCT chunk_id) > 0)
    Buffers: shared hit=2
    ->  Index Only Scan using pg_toast_2618_index on pg_toast_2618
(cost=0.15..12.33 rows=279 width=4) (actual time=0.017..1.240 rows=279.00
loops=1)
          Estimated Fetched Rows: 279
          Heap Fetches: 0
          Index Searches: 1
          Buffers: shared hit=2
```

Для оценки числа полей (`chunk_id`) можно использовать размер TOAST-таблиц, но, при использовании сжатия полей, оценка грубая. Можно подсчитывать поля только у TOAST-таблиц большого размера (~100Гб).

Причины использования секционирования

- Обойти ограничения PostgreSQL
- Физическое разбиение данных на части для удобства администрирования
  - Перестройка индексов, cluster/vacuum full отдельной секции
  - Усечение секции быстрее поиска и удаления строк
  - Вывод старых секций, добавление новых
- Секционирование прозрачно для приложений
- Простые команды для администрирования

Причины использования секционирования

Секционирование - разбиение строк, логически относящихся к одной большой таблице, на более мелкие физические части, называемые секциями. Для приложения (на логическом уровне: для команд SQL `SELECT/INSERT/DELETE`) секционированная таблица неотличима от обычной таблицы. Прозрачность для приложений (**за исключением перемещения строки в другую секцию при `UPDATE`**) - основное преимущество секционирования, по сравнению с другими способами, например, созданием множества таблиц или материализованных представлений. Секционирование используется, чтобы:

1) обойти ограничения PostgreSQL по объему хранимых данных

2) упростить администрирование: команды DDL над меньшими (по размеру) объектами выполняются быстрее.

Секционирование есть в СУБД других производителей, логика и способы управления секционированными таблицами просты в изучении и использовании. Даже, если ограничения не достигаются, секционирование используют для упрощения администрирования хранения данных в таблицах. Секционирование можно рассматривать для таблиц, размер которых сравним с размером кэша буферов PostgreSQL.

Польза секционирования может быть, например, в быстром удалении большого числа строк. Если строки хранятся в таблице , то удаление строк долгое - обновление по одной индексной записи, построчное удаление строк в блоках таблиц. При использовании секций, удаление содержимого секции выполняется на уровне объекта, если удачно выбраны ключевые столбцы, по которым строки распределяются по секциям и тип секционирования.

При удачном распределении строк по секциям, планировщик может исключать из сканирования целые секции, что ускоряет выполнение запросов, а также использовать Seq Scan по строкам, хранящимся в секциях, вместо индексного поиска по большому числу блоков, где хранятся все строки таблицы.

Компактность хранения строк в секциях повышает вероятность того, что востребованные данные (секции таблицы и их индексы) будут загружены в буферный или страничный кэш.

Можно быстро добавить строки в секционированную таблицу: подготовив строки в обычной таблице и, прицепив эту таблицу, как секцию (`ATTACH PARTITION`).

Обычно, данные, через какое-то время, теряют актуальность и доступ к ним становится менее активным. Секции с наименее актуальными данными можно выводить из таблицы или переносить в табличные пространства с меньшей стоимостью хранения или даже перегружать в другие базы данных и, затем, использовать как внешние таблицы.

Виды секционирования

- Декларативное
  - Секции могут быть:
    - простыми
    - внешними (`CREATE FOREIGN TABLE секция PARTITION OF tab`)
    - секционированными таблицами (subpartitioning)
  - В декларации (определении) указывается:
    - метод секционирования
    - ключ секционирования
  - Секционированные таблицы физически не хранят строки
- Наследованием:
  - `CREATE TABLE наследник LIKE tab...;`
  - `ALTER TABLE tab ATTACH PARTITION наследник...;`

Виды секционирования

В 10 версии PostgreSQL появилось декларативное секционирование (partitioning).

Декларация секционирования - это указание:

1) **метода секционирования**;

2) **ключа секционирования** - названий столбцов или выражений, по значению которых будет определяться, в какой секции должна находиться строка.

Декларация указывается при создании секционированной таблицы.

Секционированная таблица физически не хранит строки, не имеет файлов, представляет собой определение (декларацию) таблицы - то, что можно указать в команде `CREATE/ALTER TABLE`. После создания определения секционированной таблицы, создаются таблицы-секции, в которых указывается, к какой секционированной таблице она относится. В определениях таблиц-секций указываются значения (для секционирования методом `RANGE` указываются диапазоны значений) ключа секционирования, в соответствии с которыми, строки будут помещаться в секцию.

Таблицы-секции могут быть:

1) **обычными** таблицами;

2) **внешними** (`FOREIGN`) таблицами (определениями таблиц в другой базе данных, доступ к которым идёт через FDW - Foreign Data Wrapper);

3) **секционированными** таблицами (определениями секционированных таблиц). При указании определений секционированных таблиц, получаются подсекции (subparttitions). Разбивать можно отдельные секции, создавать дерево с разным числом уровней, использовать разные методы секционирования.

До появления декларативного секционирования, для эмуляции секционирования, использовалось наследование (`CREATE TABLE .. LIKE`). Недостаток в том. что нужно вручную создавать триггеры на родительской таблице, чтобы вставляемые строки вставлялись в нужные дочерние таблицы. Для исключения секций из сканирования, нужно вручную создавать ограничения целостности `CHECK`. Наследование и его синтаксис специфичны для PostgreSQL и отсутствует в СУБД других производителей. Синтаксис и логика декларативного секционирования схожа с СУБД других производителей. Таблицы-наследники, в отличие от декларативного секционирования, могут иметь дополнительные столбцы, отсутствующие а родительской таблице. Однако, дополнительная гибкость увеличивает вероятность ошибок.

Ограничения секционирования

- Первичный, уникальные ключи секционированной таблицы:
  - должны включать в себя все столбцы ключа секционирования
  - могут включать только названия столбцов
- Триггеры `BEFORE ROW` на `INSERT` не могут изменить секцию, куда будет вставлена строка
- Таблица и все её секции должны быть либо постоянными объектами, либо временными
- Секции декларативно секционированных таблиц не могут наследовать другие таблицы

Ограничения секционирования

1) Первичный, уникальные ключи, ограничения `EXCLUDE` секционированной таблицы:
- **должны включать в себя все столбцы ключа секционирования**
- **не могут содержать выражения или вызовы функций**. Могут содержать только названия столбцов (`EXCLUDE` только сравнивать столбцы ключа секционирования на равенство, а не использовать другие операторы типа `&&`). Ограничения `EXCLUDE` поддерживаются на секционированных таблицах **с 17 версии**.

Эти ограничения - следствие того, что на каждую секцию создаётся свой уникальный индекс ("локальный"). "Глобальные" индексы на столбцы секционированной таблицы не создаются. Чтобы гарантировать уникальность нужно, чтобы первичный или уникальный ключ использовали предположение о том, что значения (или диапазоны значений) ключа секционирования не пересекаются. Сама структура секционирования должна гарантировать отсутствие дубликатов в разных секциях.

В форках PostgreSQL могут быть "глобальные индексы" на секционированные таблицы. В Oracle Database имеются глобальные индексы, но в PostgreSQL индексы имеют физически больший размер и **большой размер "глобальных индексов" нивелирует преимущества секционирования**. При миграции на PostgreSQL, это стоит учитывать.

2) Триггеры `BEFORE ROW` на `INSERT` не могут изменить секцию, куда будет вставлена строка. `BEFORE ROW` триггеры стали поддерживать секционированные таблицы **с 13 версии**.

3) Таблица и все её секции должны быть либо постоянными объектами, либо временными. Смешивать постоянные и временные объекты нельзя.

4) Секции декларативно секционированных таблиц не могут наследовать другие таблицы, технически у секций родителем является секционированная таблица, чьими секциями они являются.

5) Если у секционированной таблицы есть уникальные индексы, то создать внешнюю таблицу в виде её секции нельзя.

6) Команда UPDATE не может переместить строку из секции внешней таблицы в другую секцию.

7) Секционированные таблицы не могут быть нежурналируемыми (UNLOGGED). До **18 версии** использование ключевого слова UNLOGGED не выдавало ошибки, но игнорировалось.

8) вычисляемые столбцы (`GENERATED ALWAYS AS`) не могут входить в ключ секционирования.

Методы декларативного секционирования

- `BY RANGE` - может быть несколько ключевых столбцов
- `BY LIST` - один ключевой столбец, список значений для каждой секции

```
create table t (c1 int, c2 text) PARTITION BY LIST (c1);
create table t1 PARTITION OF t FOR VALUES IN (1,2,3);
create table t2 PARTITION OF t FOR VALUES IN (3,4,5);
ERROR:  partition "t2" would overlap partition "t1"
```

- `BY HASH`
  - число секций равно делителю
  - секции нужно создавать сразу
  - при создании каждой секции указывается модуль (число-делитель) и остаток от деления

```
create table t (c1 int, c2 boolean) PARTITION BY hash (c1, c2);
create table t1 PARTITION OF t FOR VALUES WITH (MODULUS 1, REMAINDER 0);
create table t2 PARTITION OF t FOR VALUES WITH (MODULUS 1, REMAINDER 1);
ERROR:  remainder for hash partition must be less than modulus
```


Методы декларативного секционирования

В декларативном секционировании есть три метода распределения строк по секциям:

1) `BY RANGE`. Выбирается столбец или несколько столбцов или выражения по ним, которые назначаются "ключом" секционирования. Для каждой секции задаётся диапазон значений каждого столбца. Диапазоны значений не должны пересекаться. При задании диапазона, меньшее значение включается в диапазон, а большая не включается. Типы данных ключа должны поддерживать класс операторов для **btree**. Чаще всего используются диапазоны по целым числам. Вместо диапазонов дат можно использовать столбец, хранящий года (2026, 2027). Не стоит плодить большое число мелких секций.

2) `BY LIST`. Ключевой столбец **один**:

```
create table t (c1 text, c2 text) PARTITION BY LIST (c1, c2);
ERROR:  cannot use "list" partition strategy with more than one column
```

Для ключевого столбца задаётся список значений для каждой секции. Ограничений по типам данных для ключевого столбца нет. Списки не должны **пересекаться**:

```
create table t (c1 int, c2 text) PARTITION BY LIST (c1);
create table t1 PARTITION OF t FOR VALUES IN (1,2,3);
create table t2 PARTITION OF t FOR VALUES IN (3,4,5);
ERROR:  partition "t2" would overlap partition "t1"
```

3) `BY HASH`. Появилось **в 11 версии**, используется реже. Имеет смысл, чтобы обойти ограничения PostgreSQL, повысить скорость вставки строк или для шардинга (внешние таблицы как секции). **Число секций равно делителю** и секции нужно создавать сразу. При создании каждой секции указывается модуль (число-делитель) и остаток от деления. В секцию попадут строки, для которых двоичное значение ключевых столбцов, делённое на модуль, равняется остатку от деления:

```
create table t (c1 int, c2 boolean) PARTITION BY hash (c1, c2);
create table t1 PARTITION OF t FOR VALUES WITH (MODULUS 1, REMAINDER 0);
create table t2 PARTITION OF t FOR VALUES WITH (MODULUS 1, REMAINDER 1);
ERROR:  remainder for hash partition must be less than modulus
```

Создание секционированных таблиц

- Сначала создается сама таблица, которая будет включать секции и/или подсекции:
- `CREATE [TEMP] TABLE имя (...) PARTITION BY {RANGE | LIST | HASH} ({ имя_столбца | (выражение)} [ класс_операторов ] [,...] )`
- Затем создаются таблицы, которые будут секциями:
  `CREATE [TEMP] TABLE имя_секции PARTITION OF имя {FOR VALUES границы | DEFAULT } [PARTITION BY ...]`
  - `границы` для списка: `IN (..,..[NULL],..)`
  - для диапазона: `FROM (.. | MINVALUE ) TO (.. | MAXVALUE)`
  - для хэш: `WITH (MODULUS число, REMAINDER число)`

Создание секционированных таблиц

Сначала создается сама таблица, которая будет включать секции и/или подсекции:

```
CREATE TABLE имя (...) PARTITION BY { RANGE | LIST | HASH }
  ({ имя_столбца | (выражение)} [ класс_операторов ] [,...] );
```

Обязательно указывается один из трёх методов секционирования, а за ним в круглых скобках список столбцов (или выражений), который будет ключом секционирования.

Для `LIST` - один столбец или выражение, для `RANGE` и `HASH` - от 1 до 32.

Затем создаются таблицы, которые будут секциями:

```
CREATE TABLE секция PARTITION OF имя {FOR VALUES границы | DEFAULT}
[PARTITION BY ...];
```

`границы` для `LIST` указываются как список: `IN (..,..[NULL],..)`

для `RANGE` как диапазон: `FROM (.. | MINVALUE ) TO (.. | MAXVALUE)`

для `HASH` как делитель и остаток от деления: `WITH (MODULUS число, REMAINDER число)`.

В значениях нельзя использовать подзапросы и функции, возвращающие множества.

Если задаётся `MINVALUE` или `MAXVALUE`, то же значение должно стоять в следующих за ним столбцах. Например: `(10, MINVALUE, 0)` - не верно; `(10, MINVALUE, MINVALUE)` верно.

Диапазон `FROM ('infinity') TO (MAXVALUE)` не пустой, в нём есть значение `'infinity'`.

Только для `LIST` в списке одной из секций можно указать `NULL`.

Единственную секцию `DEFAULT` можно (но необязательно) создать для таблиц, секционированных как `LIST` и `RANGE` (но не для `HASH`).

Изменение имён и типов столбцов в секционируемой таблице будет автоматически распространяться во все секции. Ограничения `CHECK` автоматически наследуются всеми секциями. Для отдельных секций можно задать дополнительные ограничения `CHECK`; дополнительные ограничения с теми же именами и условиями, как в родительской таблице, будут объединены с родительским ограничением. Для отдельных секций можно задать свои значения по умолчанию, только это бессмысленно: оно не будет действовать при добавлении строк через секционированную таблицу.

`TRUNCATE` каскадно распространяется на дочерние секции, но может выполняться и на отдельных секциях.

Для удаления и добавления секции командой `CREATE/DROP TABLE` требуется установить блокировку ACCESS EXCLUSIVE в родительской таблице. Для выполнения этих операций с более слабой блокировкой стоит использовать команду `ALTER TABLE ATTACH/DETACH PARTITION`.

Выражения в ключе секционирования

- Выражения бесполезны, так как таблица не сможет иметь первичный или уникальный ключи:

```
CREATE TABLE t (id int, d date) PARTITION BY range (extract(YEAR from d));
alter table t add constraint t_pk primary key (id);
ERROR:  unsupported PRIMARY KEY constraint with partition key definition
DETAIL:  PRIMARY KEY constraints cannot be used when partition keys include expressions.
```

- Вычисляемые столбцы, как хранимые, так и виртуальные не могут использоваться:

```
create table t (id serial,d date,y int GENERATED ALWAYS AS (extract(YEAR from d)) VIRTUAL, CONSTRAINT t_id PRIMARY KEY (id, y)) PARTITION BY RANGE (y);
ERROR:  cannot use generated column in partition key
```

  - автоинкрементальные (IDENTITY) могут использоваться
- Функции, используемые в выражениях должны иметь категорию изменчивости IMMUTABLE

Выражения в ключе секционирования

Функции, используемые в выражениях должны иметь категорию изменчивости **IMMUTABLE**.

Команда для поиска **IMMUTABLE** функций с аргументом типа **date**:

```
SELECT distinct proname, pg_get_function_arguments(oid) FROM pg_proc a WHERE provolatile = 'i' AND prokind = 'f' AND pg_get_function_arguments(oid) like '%date%' ORDER BY 1;
```

Функции to_char, to_number, to_date - STABLE, функция **extract** - **IMMUTABLE**.

Пример секционирования по выражениям:

```
CREATE TABLE t (id int, balance int) PARTITION BY range ((id*balance));
CREATE TABLE t (id int, d date) PARTITION BY range (extract(YEAR from d));
```

Ценность возможности секционирования по выражениям невысока, так как таблица не сможет иметь первичный или уникальный ключи:

```
alter table t add constraint t_pk primary key (id);
ERROR:  unsupported PRIMARY KEY constraint with partition key definition
DETAIL:  PRIMARY KEY constraints cannot be used when partition keys include expressions.
```

Вычисляемые столбцы, как хранимые, так и виртуальные не могут использоваться:

```
create table t (id serial, d date, y int GENERATED ALWAYS AS (extract(YEAR from d)) VIRTUAL, CONSTRAINT t_id PRIMARY KEY (id, y)) PARTITION BY RANGE (y);
ERROR:  cannot use generated column in partition key
```

Начиная с 17 версии могут использоваться автоинкрементальные столбцы GENERATED {ALWAYS | BY DEFAULT} AS **IDENTITY**.

Описание функции **extract**, которая возвращает значение типа numeric:
https://docs.tantorlabs.ru/tdb/ru/18_3/se/functions-datetime.html#FUNCTIONS-DATETIME-EXTRACT

Функцию date_part не рекомендуется использовать, так как она возвращает значение double precision, что может привести к потере точности, **extract** предпочтительнее.

Секционирование по диапазону (BY RANGE)

- Пример диапазона целых чисел с автоинкрементом:

```
CREATE TABLE t (id bigint GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
 balance int ) PARTITION BY RANGE (id);
CREATE TABLE t_1 PARTITION OF t FOR VALUES FROM (0) TO (10);
CREATE TABLE t_2 PARTITION OF t FOR VALUES FROM (10) TO (20);
CREATE TABLE t_3 PARTITION OF t FOR VALUES FROM (20) TO (MAXVALUE);
insert into t (balance) select generate_series(1,30);
select (select count(*) from t) t, (select count(*) from t_1) t_1,
(select count(*) from t_2) t_2, (select count(*) from t_3) t_3;
explain select * from t where id>10 and id<20;
 t  | t_1 | t_2 | t_3
----+-----+-----+-----
 30 |   9 |  10 |  11
```

- В плане исключены секции, осталась только t_2:

```
explain select * from t where id>15 and id<18;
Bitmap Heap Scan on t_2 t  (cost=4.25..14.89 rows=10 width=12)
  Recheck Cond: ((id > 15) AND (id < 18))
  Estimated Fetched Rows: 10
  ->  Bitmap Index Scan on t_2_pkey  (cost=0.00..4.25 rows=10 width=0)
        Index Cond: ((id > 15) AND (id < 18))
```


Секционирование по диапазону (BY RANGE)

Обычно, первичный ключ это столбец с типом `bigint`. В секционированной таблице первичный ключ должен включать в себя ключ секционирования. `IDENTITY` столбцы могут использоваться **с 17 версии**. Пример:

```
DROP TABLE IF EXISTS t;
CREATE TABLE t (id bigint GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
 balance int not null) PARTITION BY RANGE (id);
CREATE TABLE t_1 PARTITION OF t FOR VALUES FROM (0) TO (10);
CREATE TABLE t_2 PARTITION OF t FOR VALUES FROM (10) TO (20);
CREATE TABLE t_3 PARTITION OF t FOR VALUES FROM (20) TO (MAXVALUE);
insert into t (balance) select generate_series(1,30);
```

Диапазоны значений не должны пересекаться.

Нижняя граница включается в диапазон, верхняя граница не включается. **MINVALUE** и **MAXVALUE** - нижняя или верхняя граница отсутствует.

```
select (select count(*) from t) t, (select count(*) from t_1) t_1, (select
count(*) from t_2) t_2, (select count(*) from t_3) t_3;
 t  | t_1 | t_2 | t_3
----+-----+-----+-----
 30 |   9 |  10 |  11
```

Планировщик смог исключить секции из сканирования. Используется `Index Scan` по индексу `t_2_pkey` секции `t_2`:

```
explain select * from t where id>15 and id<18;
Bitmap Heap Scan on t_2 t  (cost=4.25..14.89 rows=10 width=12)
  Recheck Cond: ((id > 15) AND (id < 18))
  Estimated Fetched Rows: 10
  ->  Bitmap Index Scan on t_2_pkey  (cost=0.00..4.25 rows=10 width=0)
        Index Cond: ((id > 15) AND (id < 18))
```

```
 id | balance
----+---------
 16 |      16
```

Секционирование по диапазону или по хэшу с ключом в виде целочисленного столбца (без автоинкремента) используется в для `pgbench_accounts` в тестовой схеме утилиты `pgbench`:

pgbench -i -s 1 --partitions=8 --partition-method=range (или **hash**)

Секционирование по списку (BY LIST)

- Пример:

```
drop table if exists t;
CREATE TABLE t (id bigserial, city text, primary key (id, city))
PARTITION BY LIST (city);
create table t1 PARTITION OF t FOR VALUES IN ('MSK', NULL);
create table t2 PARTITION OF t FOR VALUES IN ('SPB','EKB');
insert into t (city) values ('KRS');
ERROR:  no partition of relation "t" found for row
DETAIL:  Partition key of the failing row contains (city) = (KRS).
create table t3 PARTITION OF t DEFAULT;
insert into t (city) values ('KRS') RETURNING *;
 id | city
----+------
  2 | KRS
```

- В плане исключены все секции, кроме t3:

```
analyze t;
explain select * from t where city ='KRS';
 Seq Scan on t3 t  (cost=0.00..1.01 rows=1 width=12)
   Filter: (city = 'KRS'::text)
```


Секционирование по списку (BY LIST)

Указываются не диапазоны, а конкретные значения. Типы столбцов для диапазонов и списков те же самый. Требование для типа- возможность создать индекс типа btree (тип должен иметь класс операторов для метода доступа btree). Список типов выдаст команда:

```
\dAc btree
  List of operator classes
  AM   | Input type | Storage type | Operator class | Default?
-------+------------+--------------+----------------+----------
 btree | anyarray   |              | array_ops      | yes
..
(44 rows)
```

Диапазоны удобно использовать для дат и чисел. Списки для текста. В одной из секций LIST можно указать значение NULL, но это необязательно. Также, можно добавить DEFAULT секцию, в которую будут вставляться значения, которые не соответствуют другим секциям. Если такой секции нет, то при вставке или обновлении поля, для значения которого нет подходящей секции, будет выдаваться ошибка, показанная на слайде.

```
analyze t;
explain select * from t where city ='KRS';
      QUERY PLAN
--------------------
 Seq Scan on t3 t  (cost=0.00..1.01 rows=1 width=12)
   Filter: (city = 'KRS'::text)
```

Запрос по столбцу ID не может исключить секции и сканирует их все:

```
explain select * from t where id=2;
 Append  (cost=0.00..1.03 rows=3 width=31)
   ->  Seq Scan on t2 t_1  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t1 t_2  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t3 t_3  (cost=0.00..1.01 rows=1 width=12)
         Filter: (id = 2)
```

Если много строк, то будет использоваться Index Scan, вместо Seq Scan.

Секционирование по хэшу (BY HASH)

- Пример:

```
drop table if exists t;
create table t (c1 int PRIMARY KEY, c2 int) PARTITION BY hash (c1 int4_ops);
create table t1  PARTITION OF t FOR VALUES WITH (MODULUS 4, REMAINDER 0);
create table t2  PARTITION OF t FOR VALUES WITH (MODULUS 4, REMAINDER 1);
create table t31 PARTITION OF t FOR VALUES WITH (MODULUS 8, REMAINDER 2);
create table t32 PARTITION OF t FOR VALUES WITH (MODULUS 8, REMAINDER 6);
create table t4  PARTITION OF t FOR VALUES WITH (MODULUS 4, REMAINDER 3);
insert into t select id, id from generate_series(1, 100000) id;
```

- Секции исключаются, только при использовании равенства:

```
explain select * from t where c1 = 5;
 Index Scan using t2_pkey on t2 t  (cost=0.29..8.30 rows=1 width=8)
   Index Cond: (c1 = 5)
   Estimated Fetched Rows: 1
explain select * from t where c1 > 48 and c1 <49;
   сканируются все секции
```


Секционирование по хэшу (BY HASH)

Метод HASH появился **в 11 версии**, используется реже LIST и RANGE. Используется, чтобы обойти ограничения PostgreSQL на размер таблиц; повысить скорость вставки строк; для шардинга (внешние таблицы как секции). **Число секций равно делителю** (MODULUS) и секции нужно создавать сразу. При создании каждой секции указывается модуль (число-делитель) и остаток от деления. В секцию попадут строки, для которых двоичное значение ключевых столбцов, делённое на модуль, равняется остатку от деления (REMAINDER)

При создании HASH секции нужно указать делитель и остаток от деления. Делитель - целое положительное число, остаток - неотрицательное число, меньшее, чем делитель. Число секций должно быть равно делителю. Также, секциям можно назначить и разные модули, с условием, что делители делят другие делители. Это позволяет увеличивать число секций, не производя перемещение всех строк всех секций.

Например, таблица из 4 секций (делитель 4), и нужно увеличить число секций до 8. Можно отсоединить одну из секций, создать две новые секции указав делителем 8, покрывающих ту же часть пространства ключа (одну секцию с остатком, равным остатку отсоединённой секции, а вторую с остатком, равным значению остатка плюс 4), и наполнить эти секции данными. Можно проделать то же самое с остальными секциями, пока они все не будут заменены. Хотя придётся перемещать большие объёмы данных, это может быть лучше, чем создавать новую секционированную таблицу и перемещать в нее все строки за один раз.

Секции исключаются только при использовании равенства, иначе сканируются все секции:

```
explain select * from t where c1 > 48 and c1 <50;
Append  (cost=0.29..41.56 rows=5 width=8)
 ->  Index Scan using t1_pkey on t1 t_1  (cost=0.29..8.31 rows=1 width=8)
       Index Cond: ((c1 > 48) AND (c1 < 50)) Estimated Fetched Rows: 1
 ->  Index Scan using t2_pkey on t2 t_2  (cost=0.29..8.31 rows=1 width=8)
       Index Cond: ((c1 > 48) AND (c1 < 50)) Estimated Fetched Rows: 1
 ->  Index Scan using t4_pkey on t4 t_3  (cost=0.29..8.31 rows=1 width=8)
       Index Cond: ((c1 > 48) AND (c1 < 50)) Estimated Fetched Rows: 1
 ->  Index Scan using t31_pkey on t31 t_4  (cost=0.29..8.30 rows=1 width=8)
       Index Cond: ((c1 > 48) AND (c1 < 50)) Estimated Fetched Rows: 1
 ->  Index Scan using t32_pkey on t32 t_5  (cost=0.29..8.30 rows=1 width=8)
       Index Cond: ((c1 > 48) AND (c1 < 50)) Estimated Fetched Rows: 1
```

Индексы по первичному ключу могут быть только типа btree.

Секционирование и производительность

- Могут сканироваться все секции:

```
explain select * from t where id=2;
 Append  (cost=0.00..1.03 rows=3 width=31)
   ->  Seq Scan on t2 t_1  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t1 t_2  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t3 t_3  (cost=0.00..1.01 rows=1 width=12)
         Filter: (id = 2)
```

- Добавление подсекций:

```
CREATE TABLE t (id bigserial, city text, primary key (id, city)) PARTITION BY
LIST (city);
create table t1 PARTITION OF t FOR VALUES IN ('MSK') PARTITION BY RANGE (id);
CREATE TABLE t11 PARTITION OF t1 FOR VALUES FROM (1) TO (10);
create table t2 PARTITION OF t FOR VALUES IN ('KRS') PARTITION BY RANGE (id);
CREATE TABLE t21 PARTITION OF t2 FOR VALUES FROM (1) TO (10);
explain select * from t where id=2;
 Append  (cost=0.00..0.01 rows=2 width=40)
   ->  Seq Scan on t21 t_1  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t11 t_2  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
```


Секционирование и производительность

Запрос по столбцу ID не может исключить секции и сканирует их все:

```
explain select * from t where id=2;
 Append  (cost=0.00..1.03 rows=3 width=31)
   ->  Seq Scan on t2 t_1  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t1 t_2  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t3 t_3  (cost=0.00..1.01 rows=1 width=12)
         Filter: (id = 2)
```

Метод доступа не играет роли - если много строк, то будет использоваться `Index Scan`, вместо `Seq Scan`.

Если планировщик не может исключить секции, сканируются все секции. Исключение секций не повышает производительность по сравнению с несекционированной таблицей, оно сглаживает последствия деградации. Секционирование используется для обхода лимитов на размер таблиц и улучшения управляемости - отцепление (`DETACH`) или усечение секций, перестройка более мелких индексов по секциям, вместо большого по всей таблице. Начав секционировать и увидев, что оно приводит к сканированию всех секций, возникает идея сгладить проблему - добавить уровень секционирования, но это даст эффекта - исключение секций нивелируется увеличением числа секций: `CREATE TABLE t (id bigserial, city text, primary key (id, city)) PARTITION BY LIST (city);`

```
create table t1 PARTITION OF t FOR VALUES IN ('MSK') PARTITION BY RANGE (id);
CREATE TABLE t11 PARTITION OF t1 FOR VALUES FROM (1) TO (10);
create table t2 PARTITION OF t FOR VALUES IN ('KRS') PARTITION BY RANGE (id);
CREATE TABLE t21 PARTITION OF t2 FOR VALUES FROM (1) TO (10);
analyze t;
explain select * from t where id=2;
 Append  (cost=0.00..0.01 rows=2 width=40)
   ->  Seq Scan on t21 t_1  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
   ->  Seq Scan on t11 t_2  (cost=0.00..0.00 rows=1 width=40)
         Filter: (id = 2)
```

Подсекции

- Каждая секция может иметь подсекции (секции-наследники) или не иметь их
- Ограничений по числу уровней нет
- Подсекции могут использовать метод секционирования
- В примере секции `t1`, `t2`, `t3` состоят из подсекций:

```
drop table if exists t;
CREATE TABLE t (id bigint GENERATED BY DEFAULT AS IDENTITY, reg int, primary key
(id, reg)) PARTITION BY RANGE (id);
create table t1 PARTITION OF t FOR VALUES FROM  (0) TO (10) PARTITION BY HASH (reg);
create table t2 PARTITION OF t FOR VALUES FROM (10) TO (15) PARTITION BY LIST (reg);
create table t3 PARTITION OF t FOR VALUES FROM (15) TO (20) PARTITION BY RANGE(reg);
CREATE TABLE t11 PARTITION OF t1 FOR VALUES WITH (MODULUS 2, REMAINDER 0);
CREATE TABLE t12 PARTITION OF t1 FOR VALUES WITH (MODULUS 2, REMAINDER 1);
create table t21 PARTITION OF t2 FOR VALUES IN (10,11,12);
create table t22 PARTITION OF t2 FOR VALUES IN (13,14);
CREATE TABLE t31 PARTITION OF t3 FOR VALUES FROM (15) TO (17);
CREATE TABLE t32 PARTITION OF t3 FOR VALUES FROM (17) TO (20);
insert into t (reg) select generate_series(1,19);
```


Подсекции

В примере два уровня секционирования. Каждая секция может иметь подсекции (секции-наследники) или не иметь их, то есть дерево наследования может быть "несбалансированным", то есть путь от вершины до "листовой" секции, может иметь разное число уровней.

В примере на слайде, выражения, отмеченные цветом можно убрать.

Промежуточные секции `t1`, `t2`, `t3` не хранят данные, они хранятся в "листовых" секциях.

Каждая секция может использовать свой метод секционирования. В примере, подсекции секций `t1`, `t2`, `t3` используют разные методы секционирования.

Подсекции можно секционировать по тому же столбцу, что и родительская секция, но делать это изначально это не имеет смысла, так как можно родительскую разделить на несколько секций на её же уровне, а не создавать наследников.

Несколько уровней секционирования используется, если:

1) секции получаются крупными и их нужно дополнительно разделить на более мелкие части

2) нужно убирать из таблицы (`DETACH`) секции по разным критериям. В примере на слайде - убрать данные со старыми идентификаторами (`id=0..9`), но оставить старые данные по каким-то регионам (`reg=1..16`) - убрать секцию `t32` (`id=0..9 AND reg=17..20`).

4) Реплики заполняют буферный кэш независимо друг от друга. Если реплики обслуживают регионы и запросы на репликах идут по региональным данным, то добавление региона в ключ секционирования позволит не загружать в буферный кэш блоки, относящиеся к другим регионам.

5) в процессе работы оказывается, что строки распределяются неравномерно по секциям и какие-то секции оказываются большими. **В 19 версии** появилась команда расщепления секции на несколько частей:

```
ALTER TABLE t SPLIT PARTITION t3 INTO (PARTITION t31 FOR VALUES FROM (15) TO
(17), PARTITION t32 FOR VALUES FROM (17) TO (20));
```

Также, **в 19 версии** появилась команда слияния секций:

```
ALTER TABLE t MERGE PARTITIONS (t31, t32) INTO t3;
```

Обе команды эксклюзивно блокируют секционированную таблицу на время работы. Для секций по диапазону секции должны быть смежными. Для `HASH` не реализовывали, так как этот метод секционирования не сильно востребован.

Просмотр сведений о секциях

- Команды psql:

```
\dP+                    List of partitioned relations
Schema |  Name  |  Owner   |       Type        |Table|Access method|Total size
-------+--------+----------+-------------------+-----+-------------+----------
public | t      | postgres | partitioned table |     |             | 24 kB
public | t_pkey | postgres | partitioned index | t   | btree       | 48 kB
\d+ t
                  Partitioned table "public.t"
 Column  |  Type   | Nullable |             Default              | Storage |
---------+---------+----------+----------------------------------+---------+
 id      | bigint  | not null | generated by default as identity | plain   |
 balance | integer | not null |                                  | plain   |
Partition key: RANGE (id)
Indexes:
    "t_pkey" PRIMARY KEY, btree (id, reg)
Not-null constraints:
    "t_id_not_null" NOT NULL "id"
    "t_reg_not_null" NOT NULL "reg"
Partitions: t1 FOR VALUES FROM ('0') TO ('10'), PARTITIONED,
            t2 FOR VALUES FROM ('10') TO ('15'), PARTITIONED,
            t3 FOR VALUES FROM ('15') TO ('20'), PARTITIONED
```


Просмотр сведений о секциях

Команда `psql` **`\dP`** показывает список секционированных объектов.
`\dPt` - секционированные таблицы
`\dPi` - секционированные индексы
Команда `psql` **`\d+`** (с модификатором `+`) покажет список секций. Без модификатора, только тип секционирования и число секций:

```
\d t
                        Partitioned table "public.t"
Column  |  Type   | Nullable |              Default
--------+---------+----------+----------------------------------
id      | bigint  | not null | generated by default as identity
balance | integer | not null |
Partition key: RANGE (id)
Indexes:
    "t_pkey" PRIMARY KEY, btree (id, reg)
Number of partitions: 3 (Use \d+ to list them.)
```

Последовательность для автоинкрементального столбца создана на таблицу, а не на секции:

```
select pg_get_serial_sequence('t', 'id');
 pg_get_serial_sequence
------------------------
 public.t_id_seq
\d t_id_seq
                           Sequence "public.t_id_seq"
 Type  |Start |Minimum |       Maximum       |Increment |Cycles? |Cache
-------+------+--------+---------------------+----------+--------+-----
bigint |    1 |      1 |9223372036854775807 |        1 |no      |    1
Sequence for identity column: public.t.id
```

При вставке строки в таблицу, генерируется значение последовательности, по которому определяется секция для вставки. Сама секционированная таблица физически не хранит строки, их хранят только секции, не являющиеся секционированными.

Функции для получения сведений о секциях

- функция для просмотра дерева секционирования:
- **корневая** таблица или индекса в дереве секций:

```
select pg_partition_root('t31_pkey');
 t_pkey
```

- список **вышестоящих** таблиц или индексов:

```
select pg_partition_ancestors('t21');
 t21
 t2
 t
```

```
select * from pg_partition_tree('t');
 relid | parentrelid | isleaf | level
-------+-------------+--------+------
 t     |             | f      |     0
 t1    | t           | f      |     1
 t2    | t           | f      |     1
 t3    | t           | f      |     1
 t11   | t1          | t      |     2
 t12   | t1          | t      |     2
 t21   | t2          | t      |     2
 t22   | t2          | t      |     2
 t31   | t3          | t      |     2
 t32   | t3          | t      |     2
```

- логически эквивалентное определение секции:

```
select pg_get_partition_constraintdef('t1'::regclass);
 ((id IS NOT NULL) AND (id >= '0'::bigint) AND (id < '10'::bigint))
```


Функции для получения сведений о секциях

Функция `pg_partition_tree('имя')` выдаёт данные в виде дерева для секционированной таблицы или секционированного индекса, по одной секции в отдельной строке. Выдаёт имя секции, её непосредственного родителя, признак того, что секция листовая, и целое число, показывающее уровень секции в иерархии. На уровне 0 будет находится запрашиваемая таблица или индекс, на уровне 1 её непосредственные потомки-секции и т. д.:

```
select * from pg_partition_tree('t1');
 relid | parentrelid | isleaf | level
-------+-------------+--------+------
 t1    | t           | f      |     0
 t11   | t1          | t      |     1
 t12   | t1          | t      |     1
```

Функция `pg_get_partition_constraintdef(OID)` восстанавливает (не то, которое было в команде создания секции, а логически эквивалентное) определение секции:

```
select pg_get_partition_constraintdef('t1'::regclass);
 ((id IS NOT NULL) AND (id >= '0'::bigint) AND (id < '10'::bigint))
select pg_get_partition_constraintdef('t11'::regclass);
 ((id IS NOT NULL) AND (id >= '0'::bigint) AND (id < '10'::bigint) AND
satisfies_hash_partition('18736'::oid, 2, 0, reg))
```

Функция, которая выдаёт название корневой таблицы или индекса в дереве секционирования:

```
select pg_partition_root('t31_pkey');
 pg_partition_root
-------------------
 t_pkey
```

Функция, выдающая список вышестоящих таблиц или индексов:

```
select pg_partition_ancestors('t21');
 relid
-------
 t21
 t2
 t
```

https://docs.tantorlabs.ru/tdb/ru/18_3/se/functions-admin.html#FUNCTIONS-INFO-PARTITION

Смена строкой секции

- `tableoid` - `OID` секции, в которой находится строка
- `ctid` - адрес физического расположения строки
  - **специальное значение `(FFFFFFF, FFFD)`** - более новая версия строки находится в другой секции

```
delete from t where id>3;
select xmin, xmax, ctid, tableoid, * from t;
  xmin  | xmax | ctid  | tableoid | id | reg
--------+------+-------+----------+----+-----
 488766 |    0 | (0,1) |    25802 |  1 |   1
 488766 |    0 | (0,2) |    25802 |  2 |   2
 488766 |    0 | (0,1) |    25809 |  3 |   3
update t set id = 11, reg = 11 where id = 1;
select lp, t_xmin, t_xmax, t_ctid, t_data from
heap_page_items(get_raw_page('t11','main',0));
 lp | t_xmin | t_xmax |       t_ctid       |            t_data
----+--------+--------+--------------------+----------------------------
  1 | 488766 | 488773 | (4294967295,65533) | \x010000000000000001000000
  2 | 488766 |      0 | (0,2)              | \x020000000000000002000000
```


Смена строкой секции

`tableoid` - `OID` секции, в которой физически содержится строка. Значения имеют смысл для секционированных и унаследованных таблиц.

`ctid` - адрес физического расположения версии строки. Используя `ctid`, серверный процесс получает прямой доступ к блоку данных таблицы без полного сканирования всех блоков. В заголовке версии строки хранится адрес следующей (созданной в результате `UPDATE`) версии строки. Это не всегда "цепочка", связь может теряться, так как вакуум может удалить более новую версию строки раньше, чем старую (обработал блок раньше) и старая версия строки будет ссылаться на отсутствующую версию.

Если версия последняя, то `ctid` хранит адрес этой версии (указывает на самого себя).

Для секционированных таблиц, если `UPDATE` привел к тому, что новая версия переместилась в другую секцию (значение столбца, входящего в ключ секционирования изменилось), устанавливается **специальное значение `(FFFFFFF, FFFD)`**:

```
update t set id = 11, reg = 11 where id = 1;
select lp, t_xmin, t_xmax, t_ctid, t_data from
heap_page_items(get_raw_page('t11','main',0));
 lp | t_xmin | t_xmax |       t_ctid       |            t_data
----+--------+--------+--------------------+----------------------------
  1 | 488766 | 488773 | (4294967295,65533) | \x010000000000000001000000
  2 | 488766 |      0 | (0,2)              | \x020000000000000002000000
select * from heap_page_items(get_raw_page('t','main',0));
ERROR:  cannot get raw page from relation "t"
DETAIL:  This operation is not supported for partitioned tables.
```

Перемещать строки между секциями можно только между нижестоящими секциями. Если в `UPDATE` использовать имя корневой таблицы, проблем нет. Если использовать имя секции, то попытка переместить строку на соседний уровень выдаст ошибку:

```
ERROR: new row for relation "секция" violates partition constraint
```

Имена секций для изменения строк можно использовать, но это нежелательно.

Большое число секций не увеличивает производительность, а упрощает администрирование и оправдано только на больших объемах данных. Не стоит создавать большое число маленьких секций. Размер секций стоит выбирать такой, больше которого длительность выполнения команд (например, заполнение секции строками командой `COPY`) над секцией ощутим.

Ошибка при смене строкой секции

- после секционирования таблицы возможны ошибки при обновлении и удалении строк:

```
begin;
BEGIN
update t set id = 12, reg = 12 where id = 2;
UPDATE 1
commit;
COMMIT
```

```
update t set id = 12, reg = 12 where id = 2;

ERROR:  tuple to be locked was already moved to
another partition due to concurrent update
```

- если таблица не секционирована, команда выполнится без ошибки (обновит ноль строк)
- приложение не должно менять значения в столбцах секционирования или нужно изменить логику приложения: использовать перед `UPDATE` и `DELETE` команду `SELECT FOR UPDATE SKIP LOCKED;`

Ошибка при смене строкой секции

Преимущество секционирования в прозрачности для приложения: можно секционировать таблицы и код приложения продолжит работать. Но есть случай, когда работа команд меняется и появляется ошибка при выполнении `UPDATE` или `DELETE`, если строка была перемещена в другую секцию параллельно выполнявшейся командой.

Если транзакция перемещает строку в другую секцию, а во время выполнения транзакции вторая транзакция (или одиночная команда) попытается обновить или удалить перемещаемую строку, то вторая транзакция получит ошибку.

Пример (номера транзакций 1 и 2):

```
1 begin;
1 update t set id = 12, reg = 12 where id = 2;
2 update t set id = 12, reg = 12 where id = 2;
1 commit;
2 ERROR:  tuple to be locked was already moved to another partition due to
concurrent update
```

Если бы таблица была несекционированной:

```
drop table if exists t;
create table t (id bigint GENERATED BY DEFAULT AS IDENTITY, reg int, primary
key (id, reg));
insert into t (reg) select generate_series(1,2);
```

то вторая транзакция не получила бы сообщение, что команда успешно выполнена и обновлено ноль строк:

```
UPDATE 0
```

После снятия блокировки команда перечитывает строку и видит, что она уже не подходит под условие `where`. Если первая транзакция откатится, то команда обновит одну строку и с секционированной и обычной таблицей.

Проблема в том, что если `UPDATE` или `DELETE` меняют более, чем одну строку, то с секционированной таблицей сбойнёт вся команда. Из-за этого внедрение секционирования не полностью прозрачно для приложения.

Чтобы избежать ошибки либо приложение не должно менять значения в столбцах секционирования или нужно изменить логику приложения: использовать перед `UPDATE` и `DELETE` команду `SELECT FOR UPDATE SKIP LOCKED;` а это усложнит код приложения.

Аналогичная ошибка будет при шардинге, если строка переместится в другой шард.

Отсоединение секций

- Секция становится самостоятельной таблицей:

```
ALTER TABLE t DETACH PARTITION t2 CONCURRENTLY;
\d+ t2
Column |  Type   | Nullable | Default | Storage |
-------+---------+----------+---------+---------+
id     | bigint  | not null |         | plain   |
reg    | integer | not null |         | plain   |
Partition key: LIST (reg)
Indexes:
    "t2_pkey" PRIMARY KEY, btree (id, reg)
Check constraints:
    "t2_id_check" CHECK (id IS NOT NULL AND id >= '10'::bigint AND id < '15'::bigint)
Not-null constraints:
    "t_id_not_null" NOT NULL "id"
    "t_reg_not_null" NOT NULL "reg"
Partitions: t21 FOR VALUES IN (10, 11, 12),
            t22 FOR VALUES IN (13, 14)
```

- Вместе с `t2` отсоединена вся ветвь: `t21` и `t22`

Отсоединение секций

Для отсоединения секций используется команда:

`ALTER TABLE имя DETACH PARTITION секция [ CONCURRENTLY | FINALIZE ];`

Отсоединяемая секция становится самостоятельной таблицей. Связи с бывшей родительской таблицей удаляются: индексы, которые входили в иерархию индексов родительской таблицы отсоединяются; триггеры, созданные как копии триггеров на родительской таблице удаляются.

Также **удаляются свойства `generated by default as identity`**, так как значения автоинкрементального столбца генерируются последовательностью, связанной с корневой таблицей.

Если используются подсекции, то в качестве имени таблицы нужно указывать непосредственного родителя. Если у отсоединяемой секции есть наследники, отсоединится вся ветвь и отсоединённая секция станет самостоятельной секционированной таблицей - корнем ветви секций.

На время выполнения команды таблицы с внешними ключами на ключи родительской таблицы блокируются в режиме `SHARE`. Секция блокируется в режиме `ACCESS EXCLUSIVE`. Родительская таблица блокируется в режиме `ACCESS EXCLUSIVE` или, если используется опция `CONCURRENTLY`, в режиме `SHARE UPDATE EXCLUSIVE`.

Режим `SHARE UPDATE EXCLUSIVE` позволяет выполнять команды `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `MERGE`, `COPY`. При отсоединении секции с опцией `CONCURRENTLY`, приложения могут продолжать работать со строками других секций.

Одновременно можно отсоединять только одну секцию.

Команда отсоединения секции не комбинируется с другими командами `ALTER TABLE`. например, нельзя одной командой отсоединять секцию и присоединять секцию или удалять столбец.

Опция `FINALIZE` используется для завершения отсоединения секции, если команда отсоединения прервалась.

Опция `CONCURRENTLY` появилась позже, чем возможность отсоединить секцию с эксклюзивной блокировкой родительской таблицы, поэтому она предпочтительнее, но может быть **причина** не использовать `CONCURRENTLY`: если будет ошибка, которая не даст выполнить `FINALIZE`, таблица перейдёт в неработоспособное состояние. Не будут выполняться команды `ALTER TABLE`. Из подсекций придётся выгрузить данные и удалить подсекции.

Ограничение CHECK при отсоединении секций

- Добавляются ограничения:

```
\d t21                                Table "public.t21"
 Column |  Type   | Collation | Nullable |             Default
--------+---------+-----------+----------+---------------------------------
 id     | bigint  |           | not null | generated by default as identity
 reg    | integer |           | not null |
Partition of: t2 FOR VALUES IN (10, 11, 12)
Indexes: "t21_pkey" PRIMARY KEY, btree (id, reg)
ALTER TABLE t DETACH PARTITION t2 CONCURRENTLY;
\d t21                Table "public.t21"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | bigint  |           | not null |
 reg    | integer |           | not null |
...
Check constraints:
    "t2_id_check" CHECK (id IS NOT NULL AND id >= '10'::bigint AND id < '15'::bigint)
ALTER TABLE t2 DETACH PARTITION t21 CONCURRENTLY;
\d t21                Table "public.t21"
...
Check constraints:
    "t21_reg_check" CHECK (reg IS NOT NULL AND (reg = ANY ('{10,11,12}'::integer[])))
    "t2_id_check" CHECK (id IS NOT NULL AND id >= '10'::bigint AND id < '15'::bigint)
```


Ограничение CHECK при отсоединении секций

При подсоединении секций наличие ограничения `CHECK` с правильной проверкой, соответствующей `FOR VALUES` подсоединяемой секции, позволяет не выполнять проверку строк подсоединяемой секции на соответствие `FOR VALUES`.

Чтобы отсоединённую секцию можно было быстро присоединить, при отсоединении секции, к ней добавляется ограничение `CHECK`, которое соответствует диапазону или списку значений, которые могут быть в отсоединяемой секции. Пример для таблицы t:

Если отсоединить `t1` (секционированную по `HASH`), потом от `t1` отсоединить `t11`, то в `t11` добавится ограничение:

```
"t1_id_check" CHECK (id IS NOT NULL AND id >= '0'::bigint AND id < '10'::bigint)
```

Если сначала отсоединить `t11` ограничение `CHECK` не добавится.

Если отсоединить `t2` (`LIST`), потом от неё `t21`, то в `t21` добавятся:

```
"t21_reg_check" CHECK (reg IS NOT NULL AND (reg = ANY ('{10,11,12}'::integer[])))
"t2_id_check" CHECK (id IS NOT NULL AND id >= '10'::bigint AND id < '15'::bigint)
```

Если сначала от `t2` отсоединить `t21`, то в `t21` добавится эквивалентное по условию ограничение:

```
"t21_check" CHECK (id IS NOT NULL AND id >= '10'::bigint AND id < '15'::bigint AND reg IS NOT NULL AND (reg = ANY ('{10,11,12}'::integer[])))
```

Если отсоединить `t3` (`RANGE`), потом от неё `t31`, то в `t31` добавятся:

```
"t31_reg_check" CHECK (reg IS NOT NULL AND reg >= 15 AND reg < 17)
"t3_id_check" CHECK (id IS NOT NULL AND id >= '15'::bigint AND id < '20'::bigint)
```

Если сначала от `t3` отсоединить `t31`, то в `t3` добавится эквивалентное по условию ограничение:

```
"t31_check" CHECK (id IS NOT NULL AND id >= '15'::bigint AND id < '20'::bigint AND reg IS NOT NULL AND reg >= 15 AND reg < 17)
```

Если в списке секции `LIST` отсутствует `NULL` и во всех остальных случаях, стоит указывать условие `AND ... IS NOT NULL`, иначе при подсоединении секции будет выполняться проверка. **Определение секции для `CHECK` можно получить функцией `pg_get_partition_constraintdef('имя'::regclass);`**

Присоединение секций

- В присоединяемой как секция таблице:
  - столбцы и их типы те же самые, что в родительской
  - те же ограничения NOT NULL и CHECK
  - будут созданы ограничения целостности PRIMARY KEY и UNIQUE, как на родительской таблице, если их ещё нет
  - будут созданы индексы, как на родительской, если они уже есть, то будут присоединены к родительскому индексу
- Присоединяемая таблица может быть внешней (FOREIGN)
- Примеры отсоединения и присоединения секций:

```
ALTER TABLE t ATTACH PARTITION t2 FOR VALUES FROM (10) TO (15);
ALTER TABLE t2 DETACH PARTITION t21;
ALTER TABLE t2 ATTACH PARTITION t21 FOR VALUES IN (10,11,12);
ALTER TABLE t1 DETACH PARTITION t11;
ALTER TABLE t1 ATTACH PARTITION t11 FOR VALUES WITH (MODULUS 2, REMAINDER 0);
```


Присоединение секций

Команда присоединения существующей таблицы как секции:

ALTER TABLE имя ATTACH PARTITION секция {FOR VALUES границы | DEFAULT};

Присоединяемая секция может быть секционированной таблицей (иметь секции).

Таблицу можно присоединить в качестве секции по умолчанию (**DEFAULT**).

Для каждого индекса (кроме индексов ONLY), который определён на родительской в таблице, в присоединяемой таблице будет создан индекс. Если аналогичный индекс уже есть, то он будет присоединён к родительскому индексу.

На родительской таблице могут существовать индексы с опцией ONLY, созданные командой:

CREATE [UNIQUE] INDEX индекс ON ONLY таблица;

После присоединения секции, к такому родительскому индексу можно присоединить заранее созданный индекс секции командой ALTER INDEX индекс_родителя ATTACH PARTITION индекс_секции.

Если присоединяется внешняя таблица (FOREIGN), то её нельзя присоединить к таблице, на которой есть уникальный индекс.

Для каждого триггера FOR EACH ROW родительской таблицы, будет создан такой же триггер на присоединяемой таблице.

Присоединяемая таблица:

1) Должна иметь те же столбцы, что и родительская и никаких других; более того, типы столбцов должны совпадать.

2) Должны быть те же ограничения NOT NULL и CHECK, что и в родительской таблице (кроме ограничений, которые в родительской таблице помечены как NO INHERIT). Ограничения FOREIGN KEY не учитываются и не проверяются.

3) В присоединяемой таблице будут созданы ограничения целостности PRIMARY KEY и UNIQUE, как на родительской таблице, если их ещё нет. Если уже есть, то индексы этих ограничений будут присоединены к родительскому индексу.

Одновременно можно присоединять только одну секцию. Команда присоединения секции не комбинируется с другими командами ALTER TABLE.

Ускорение присоединения секции

- добавить ограничение `CHECK` которое соответствует условию `FOR VALUES`, используемое при подсоединения этой таблицы как секции
  - добавлять как `NOT VALID` и валидировать отдельной командой
- Структура первичных и уникальных ключей секции должна полностью соответствовать секционированной таблице
  - можно создать уникальные индексы, столбцы которых соответствуют уникальным индексам таблицы

Ускорение присоединения секции

Если на подсоединяемой таблице есть ограничение `CHECK`, соответствующее условию `FOR VALUES`, то при подсоединении секции, проверка строк в секции на соответствие выражению `FOR VALUES` команды присоединения не будет проводиться. Лучше это проверить на тестовых данных, так как гарантировать, что условие в ограничении будет сопоставлено с выражением нет.

После подсоединения секции ограничение `CHECK` может быть автоматически удалено, но нет гарантий: это происходит не всегда.

Добавлять ограничение как `NOT VALID`, затем валидировать отдельной командой. Если добавлять с проверкой, то на время проверки таблица будет заблокирована в режиме `ACCESS EXCLUSIVE` так, что с ней не сможет работать ни одна команда.

Структура первичных и уникальных ключей должна полностью соответствовать той таблице, к которой выполняется подсоединение.

Вместо ключей, до подсоединения можно создать уникальные индексы, столбцы которых соответствуют уникальным индексам таблицы, к которой выполняется подсоединение. Тогда индексы смогут встроиться в иерархию наследования индексов. Если будут различия, то будут создаваться индексы, что увеличит время выполнения команды подсоединения секции.

Так как объемы могут быть большими, то лучше проверить процедуру присоединения на тестовых данных. Присоединение должно выполняться моментально, сразу после получения блокировок. Если длительность подсоединения зависит от объема данных, то это значит, что выполняется проверка данных или создание индексов, а этого можно избежать.

Преобразование таблицы в секционированную

- Команды конвертации `ALTER TABLE` несекционированной таблицы в секционированную нет. Шаги:

```
drop table if exists t;
create table t (id bigint GENERATED BY DEFAULT AS IDENTITY, reg int, primary key
(id, reg));
alter table t ADD CONSTRAINT t1_id_check CHECK (id IS NOT NULL AND id >=
'1'::bigint AND id < '10'::bigint) NOT VALID;
alter table t VALIDATE CONSTRAINT t1_id_check;
begin;
set statement_timeout = '5s';
alter table t rename to t1;
alter table t1 rename constraint t_pkey to t1_pkey;
alter table t1 rename constraint t_id_not_null to t1_id_not_null;
alter table t1 rename constraint t_reg_not_null to t1_reg_not_null;
alter table t1 ALTER COLUMN id DROP IDENTITY;
create table t (id bigint GENERATED BY DEFAULT AS IDENTITY, reg int, primary key
(id, reg)) PARTITION BY RANGE (id);
alter table t ALTER COLUMN id RESTART WITH 3;
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
commit;
```


Преобразование таблицы в секционированную

Команды конвертации `ALTER TABLE` несекционированной таблицы в секционированную нет. Если нужно сделать из несекционированной секционированную, то придётся это делать вручную по шагам:

1) Добавить ограничение `CHECK` с опцией `NOT VALID`, соответствующее строкам таблицы и будущему условию `FOR VALUES` секции, в виде которой эта таблица будет подсоединена к будущей секционированной таблице. Не забыть вставить в условие `NOT NULL`, если в списке значений секции оно будет отсутствовать или секция будет `RANGE` или `HASH`.

2) Валидировать добавленные ограничения `CHECK`. **Смысл разнести добавление и валидацию в том, что на время проверки устанавливается не эксклюзиваная блокировка, а более слабая блокировка** `SHARE UPDATE EXCLUSIVE`. Более того, одной командой можно валидировать сразу несколько ограничений типов `CHECK`, `NOT NULL`, `FOREIGN KEY`.

3) Если в новой таблице будет другой первичный ключ, то удалить первичный ключ с опцией `CASCADE` и добавить новый. Перед добавлением нового ключа можно заранее создать для него уникальный индекс в режиме `CONCURRENTLY`. Важно, чтобы набор столбцов **и их порядок следования в ключе и уникальном индексе были такими же, как в секционированной таблице**.

4) Открыть транзакцию

5) Переименовать таблицу в название секции. Таблица будет заблокирована.

6) Сохранить значение автоинкрементальных столбцов и удалить на столбцах секции свойство `IDENTITY`, если оно есть

7) Переименовать ограничения целостности, в соответствии с именем секции

8) Создать секционированную таблицу с именем исходной таблицы. Переустановить значение последовательности для автоинкрементальных столбцов.

9) Присоединить переименованную таблицу, как секцию

10) Зафиксировать транзакцию

11) Если на таблице были внешние ключи или они нужны, то добавить их. Если первичный ключ был изменён, то внешние ключи тоже придётся изменить.

Добавленное ограничение целостности `CHECK` может остаться на секции, его можно удалить. В начале транзакции можно установить таймаут: `set statement_timeout = '5s';` чтобы защититься от того, что какая-то команда будет выполняться долго из-за того, что что-то не было учтено и выполняется долгая проверка или создаётся индекс, при том, что таблица заблокирована. Таймаут позволит прервать выполнение и исследовать проблему.

Отладочные сообщения

- Сообщение о достаточности ограничений:

```
set client_min_messages = debug1;
alter table t ADD CONSTRAINT t1_id_check CHECK (id >= 1 AND id < 10);
DEBUG: verifying table "t"
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
DEBUG: partition constraint for table "t1" is implied by existing constraints
```

- Сообщения о проверке данных при подсоединении:

```
alter table t ADD CONSTRAINT t1_id_check CHECK (id >= 1 AND id <= 10);
DEBUG: verifying table "t"
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
DEBUG: verifying table "t"
```

- Если на секции нет уникального индекса для первичного ключа, он будет создаваться при подсоединении секции:

```
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
DEBUG: ALTER TABLE / ADD PRIMARY KEY will create implicit index "t1_pkey" for
table "t1"
DEBUG: building index "t1_pkey" on table "t1" serially
```


Отладочные сообщения

При правильном выражении ограничения `CHECK`, при подсоединении секции, проверка строк в секции не будет выполняться. Это можно определить по скорости подсоединения. Также можно определить по отладочным сообщениям:

```
set client_min_messages = debug1;
```

Пример сообщения о проверке строк:

```
alter table t ADD CONSTRAINT t1_id_check CHECK (id >= 1 AND id < 10);
DEBUG: verifying table "t"
```

Отладочные сообщения не идеальны: по сообщению не виден уровень блокировки и не видно преимущество добавить ограничение `NOT VALID` и валидировать отдельной командой.

Пример сообщения, когда проверка не выполняется (достаточно ограничений):

```
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
DEBUG: partition constraint for table "t1" is implied by existing constraints
```

Если поменять ограничение на следующее:

```
alter table t ADD CONSTRAINT t1_id_check CHECK (id >= 1 AND id < 100);
DEBUG: verifying table "t"
```

то при присоединении секции будет вторая проверка:

```
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
DEBUG: verifying table "t"
```

Пример сообщений, если на будущей секции не будет создано уникального индекса или первичного ключа:

```
create table t (id bigint NOT NULL, reg int NOT NULL);
```

то при подсоединении секции, на неё будет создан уникальный индекс. В секции много данных и индекс будет создаваться долго, на время создания таблица будет заблокирована для изменений строк, так как индекс создаётся без `CONCURRENTLY`:

```
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
DEBUG: ALTER TABLE / ADD PRIMARY KEY will create implicit index "t1_pkey" for
table "t1"
DEBUG: building index "t1_pkey" on table "t1" serially
DEBUG: index "t1_pkey" can safely use deduplication
```

Замена первичного ключа

- Дополнительные команды:

```
drop table if exists t;
create table t(id bigint GENERATED BY DEFAULT AS IDENTITY,reg int,primary key (id));
ALTER TABLE t ADD CONSTRAINT t1_reg_not_null NOT NULL reg NOT VALID;
alter table t ADD CONSTRAINT t1_id_check CHECK (id IS NOT NULL AND id >= '1'::bigint
AND id < '10'::bigint) NOT VALID;
ALTER TABLE t VALIDATE CONSTRAINT t1_reg_not_null, VALIDATE CONSTRAINT t1_id_check;
create UNIQUE index CONCURRENTLY t1_pkey ON t(id, reg);
begin;
alter table t drop constraint t_pkey;
alter table t add constraint t1_pkey primary key using index t1_pkey;
alter table t rename to t1;
alter table t1 rename constraint t_id_not_null to t1_id_not_null;
alter table t1 ALTER COLUMN id DROP IDENTITY;
create table t (id bigint GENERATED BY DEFAULT AS IDENTITY, reg int, primary key
(id, reg)) PARTITION BY RANGE (id);
alter table t ALTER COLUMN id RESTART WITH 3;
alter table t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (10);
commit;
```


Замена первичного ключа

С маленькими объемами данных секционирование не используется. С большими объемами данных долгие блокировки нежелательны, поэтому используются более сложные процедуры замены первичного ключа и добавления ограничений целостности.

Если в новой таблице будет другой первичный ключ, то удалить первичный ключ с опцией `CASCADE` и добавить новый. Перед добавлением нового ключа можно заранее создать для него уникальный индекс в режиме `CONCURRENTLY`:

```
create UNIQUE index CONCURRENTLY t1_pkey ON t(id, reg);
```

Важно, чтобы набор столбцов **и их порядок следования в ключе и уникальном индексе были такими же, как в секционированной таблице**. Другой первичный ключ может понадобиться из-за того, что первичный ключ должен включать столбцы ключа секционирования. У таблицы может быть только один первичный ключ. Если понадобится в первичный ключ добавить столбец ключа секционирования, то придётся удалить старый первичный ключ и добавить новый:

```
alter table t drop constraint t_pkey;
alter table t add constraint t1_pkey primary key using index t1_pkey;
```

При этом будет удалён старый уникальный индекс и создан новый уникальный индекс. Создание индекса занимает время.

Так как первичный ключ кроме уникальности, которая физически обеспечивается уникальным индексом, добавляет ещё и `NOT NULL` на все свои столбцы, то стоит заранее добавить ограничение `NOT NULL` в режиме `NOT VALID` на столбцы, которые должны войти в первичный ключ и потом валидировать это ограничение:

```
ALTER TABLE t ADD CONSTRAINT t1_reg_not_null NOT NULL reg NOT VALID;
ALTER TABLE t VALIDATE CONSTRAINT t1_reg_not_null;
```

Названия ограничений и индекса, которые приведены в командах, приведены в предположении, что таких объектов ещё нет. Если имена уже используются, то они должны быть другими.

Если на старый первичный ключ ссылались **внешние ключи**, то их придётся удалить и создать новые. Но, внешние ключи создаются на уникальные и первичные и должны иметь тот же набор столбцов. Если в таблице, на которой был внешний ключ нет столбца `reg`, то придётся его добавить, либо не создавать внешний ключ.

https://www.kylehailey.com/post/postgres-partition-conversion-minimal-downtime

Секция DEFAULT

- Можно добавить к таблицам, секционированных методом LIST или RANGE
- Присоединяемая секция и секция DEFAULT блокируются в режиме ACCESS EXCLUSIVE

```
ALTER TABLE t3 DETACH PARTITION t31;
ALTER TABLE t ATTACH PARTITION t31 DEFAULT;
ERROR:  partition constraint of relation "t31" is violated by some row
ALTER TABLE t3 ATTACH PARTITION t31 FOR VALUES FROM (15) TO (17);
CREATE TABLE t4 (id bigint, reg int, primary key (id, reg));
ALTER TABLE t ATTACH PARTITION t4 DEFAULT;
\d t4_pkey
        Index "public.t4_pkey"
 Column |  Type   | Key? | Definition
--------+---------+------+------------
 id     | bigint  | yes  | id
 reg    | integer | yes  | reg
Partition of: t_pkey
primary key, btree, for table "public.t4"
```


Секция DEFAULT

У таблиц, секционированных методом LIST и RANGE (но не для HASH) может существовать секция DEFAULT. Она может отсутствовать, тогда в таблицу нельзя вставить строки, для которых нет подходящей секции. Такая секция может быть только одна.

На время выполнения команды присоединения секции (ATTACH), родительская таблица блокируется в режиме SHARE UPDATE EXCLUSIVE.

Присоединяемая секция **и секция DEFAULT** (если она есть) блокируются в режиме ACCESS EXCLUSIVE.

Блокировка секции DEFAULT нужна, чтобы проверить: нет ли в секции DEFAULT строк, которые должны находиться в присоединяемой секции. Проверка не проводится, если секция DEFAULT является внешней таблицей. Сканирование секции DEFAULT нужно выполнять при каждом добавлении к таблице новой секции.

Чем больше строк в секции DEFAULT, тем дольше выполняется проверка и удерживаются блокировки.

Раздел DEFAULT это откладывание проблем "на потом" и, **если можно обойтись без раздела DEFAULT, то лучше его не создавать**. Например, его создают из страха, что забудут создать новую секцию для наступающего диапазона дат и строки не смогут вставляться, будет выдаваться ошибка:

```
ERROR: no partition of relation "t" found for row
```

Но лучше настроить мониторинг наличия секций, чем потом столкнуться при добавлении секции с тем, что придётся переносить строки из секции DEFAULT в промежуточную таблицу из-за ошибок:

```
ERROR: updated partition constraint for default partition
"имя_DEFAULT_секции" would be violated by some row
```

а потом в новую секцию. Команда расщепления секции, которой можно расщепить секцию ALTER TABLE SPLITPARTITION имя_**DEFAULT**_секции появилась в 19 версии PostgreSQL.

Присоединение секций с подсекциями

- В присоединенную секцию добавляются правила `generated by default as identity`, которые есть на столбцах родительской таблицы
  - если присоединяемая таблица имеет подсекции, то в подсекции правило `IDENTITY` не добавляется
  - добавить его вручную нельзя

```
ALTER TABLE t21 ALTER COLUMN id ADD generated by default as identity;
ERROR:  cannot add identity to a column of a partition
```

  - придется отсоединить все подсекции и подсоединить их заново:

```
ALTER TABLE t2 DETACH PARTITION t21;
ALTER TABLE t2 DETACH PARTITION t22;
ALTER TABLE t2 ATTACH PARTITION t21 FOR VALUES IN (10,11,12);
ALTER TABLE t2 ATTACH PARTITION t22 FOR VALUES IN (13,14);
```


Присоединение секций с подсекциями

В присоединенную секцию добавляются правила `generated by default as identity`, которые есть на столбцах родительской таблицы. **Если присоединяемая таблица имеет подсекции, то в подсекции правило `IDENTITY` не добавляется.**

Добавить это правило нельзя:

```
ALTER TABLE t21 ALTER COLUMN id ADD generated by default as identity;
ERROR:  cannot add identity to a column of a partition
```

Отсутствие правила создаёт проблемы - нельзя выполнять команды:

```
ALTER TABLE t ALTER COLUMN id SET GENERATED BY DEFAULT RESTART;
ERROR:  column "id" of relation "t31" is not an identity column
```

А при использовании `CONCURRENTLY` таблица переходит в неработоспособное состояние:

```
ALTER TABLE t DETACH PARTITION t3 CONCURRENTLY;
ERROR:  column "id" of relation "t31" is not an identity column
ALTER TABLE t DETACH PARTITION t3;
ERROR:  cannot detach partition "t3"
DETAIL: The partition is being detached concurrently or has an unfinished detach.
HINT: Use ALTER TABLE ... DETACH PARTITION ... FINALIZE to complete the pending detach operation.
ALTER TABLE t DETACH PARTITION t3 FINALIZE;
ERROR:  column "id" of relation "t31" is not an identity column
ALTER TABLE t ALTER COLUMN id drop identity;
ALTER TABLE t3 ALTER COLUMN id drop identity;
ERROR:  cannot alter partition "t3" with an incomplete detach
```

Чтобы добавить правило и устранить проблему (но не проблему с невозможностью выполнить `FINALIZE`, с которой придётся удалить подсекцию), придется отсоединить все подсекции и подсоединить их заново:

```
ALTER TABLE t2 DETACH PARTITION t21; ALTER TABLE t2 DETACH PARTITION t22;
ALTER TABLE t2 ATTACH PARTITION t21 FOR VALUES IN (10,11,12);
ALTER TABLE t2 ATTACH PARTITION t22 FOR VALUES IN (13,14);
```

После этого ветвь `t2`, `t21`, `t22` будет подсоединена так же, какой она была до отсоединения. После усечения (`TRUNCATE`) таблицы, последовательность, используемая для генерации значений `IDENTITY` столбцов не сбрасывается в начальное значение. Сбросить последовательность в начальное значение можно командой:

```
ALTER TABLE t ALTER COLUMN id SET GENERATED BY DEFAULT RESTART;
```

Столбцы IDENTITY или bigserial

- При использовании IDENTITY правило не добавляется в подсекции присоединяемой таблицы
- При использовании **serial/bigserial** такой проблемы нет, в подсекции добавляется выражение Default
  - можно отсоединять, затем присоединить секции и результат будет такой же, как до отсоединения

```
CREATE TABLE t (id bigint GENERATED BY DEFAULT AS IDENTITY bigserial, reg int,
primary key (id, reg)) PARTITION BY RANGE (id);
CREATE TABLE t1 PARTITION OF t FOR VALUES FROM (1) TO (20) PARTITION BY RANGE(reg);
CREATE TABLE t11 PARTITION OF t1 FOR VALUES FROM (1) TO (10);
ALTER TABLE t DETACH PARTITION t1 CONCURRENTLY;
ALTER TABLE t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (20);
\d t11 \\
            Table "public.t11"
 Column |  Type   | Nullable | Default
--------+---------+----------+---------
 id     | bigint | not null | nextval('t_id_seq'::regclass)
```


Столбцы IDENTITY или bigserial

Столбцы serial/bigserial (заполняемые выражением Default) появились до IDENTITY столбцов. Логика GENERATED столбцов сложнее и, из-за этого, приходится учитывать больше вариантов использования таких столбцов. Большая сложность - больше багов. Скорость работы одинакова, есть несущественные отличия по поддержке этих способов генерации значений. При использовании IDENTITY с подсекциями, в версиях, где баг не исправлен, в том, что GENERATED не добавляется в подсекции присоединяемой таблицы:

```
drop table if exists t;
CREATE TABLE t (id bigint GENERATED BY DEFAULT AS IDENTITY, reg int, primary
key (id, reg)) PARTITION BY RANGE (id);
CREATE TABLE t1 PARTITION OF t FOR VALUES FROM (1) TO (20) PARTITION BY
RANGE(reg);
CREATE TABLE t11 PARTITION OF t1 FOR VALUES FROM (1) TO (10);
ALTER TABLE t DETACH PARTITION t1 CONCURRENTLY;
ALTER TABLE t ATTACH PARTITION t1 FOR VALUES FROM (1) TO (20);
```

Для добавления нужно отсоединять все подсекции и добавлять их отдельно:

```
ALTER TABLE t1 DETACH PARTITION t11;
ALTER TABLE t1 ATTACH PARTITION t11 FOR VALUES FROM (1) TO (10);
```

При использовании **serial/bigserial** такой проблемы нет, в подсекции добавляется выражение Default. Можно отсоединить секцию, присоединить её и результат будет такой же, как до отсоединения (обратимость команд):

```
\d t11 \\
            Table "public.t11"
 Column |  Type   | Nullable | Default
--------+---------+----------+---------
 id     | bigint | not null | nextval('t_id_seq'::regclass)
```

Использование IDENTITY неудобно тем, что команды ограничены в работе с этими столбцами:

```
ALTER TABLE t11 ALTER COLUMN id ADD generated by default as identity;
ERROR:  cannot add identity to a column of a partition
ALTER TABLE t11 ALTER COLUMN id drop identity;
ERROR:  cannot drop identity from a column of a partition
```

При использовании **serial/bigserial** таких ограничений нет, управляемость лучше.

Логическая репликация

- свойство публикации `publish_via_partition_root`
- По умолчанию `off` - изменения публикуются по имени секции, в которой менялись строки
  - в базе подписки должны быть созданы таблицы с именами секций, они могут быть самостоятельными таблицами, а не секциями одной таблицы
- При включении параметра (`on`):
  - можно реплицировать изменения в несекционированную таблицу или в таблицу, у которой структура секций отличается от публикуемой таблицы
  - изменения будут публиковаться с использованием имени корневой секционированной таблицы
  - на подписчике не обязательно иметь секции с теми же именами, как в публикации

Логическая репликация

У публикации есть свойство **publish_via_partition_root**:

```
CREATE PUBLICATION имя WITH (publish_via_partition_root [ = on|off ] [, ..]);
```

По умолчанию off - изменения публикуются по имени секции, в которой менялись строки. В базе подписки должны быть созданы таблицы с именами секций, они могут быть самостоятельными таблицами, а не секциями одной таблицы.

При включении параметра (`on`):

1) можно реплицировать изменения в несекционированную таблицу или в таблицу, у которой структура секций отличается от публикуемой таблицы. Подписчику не обязательно иметь секции с теми же именами, что в публикации

2) изменения будут публиковаться с использованием имени корневой секционированной таблицы

2) команды усечения секции `TRUNCATE имя_секции;` не публикуются, так как подписчику не обязательно иметь секции с теми же именами, что в публикации

3) присоединение (`ATTACH`) секции не приводит к репликации существующих строк присоединяемой секции

4) фильтр строк для каждой секции будет браться из корневой секционированной таблицы, а не из секций.

Примеры использования фильтров есть в документации:
https://docs.tantorlabs.ru/tdb/ru/18_3/se/logical-replication-row-filter.html#LOGICAL-REPLICATION-ROW-FILTER-EXAMPLES

Параметры конфигурации планировщика

- `enable_partition_pruning` позволяет планировщику исключать секции как из формируемого плана, так и на этапе выполнения
- `enable_partitionwise_join` - соединять секции, если таблицы эквисекционированы и соединение по ключу секционирования
- `enable_partitionwise_aggregate` - группировка отдельно по секциям

```
\dconfig enable_part*
    List of configuration parameters
           Parameter             | Value
---------------------------------+-------
 enable_partition_pruning        | on
 enable_partitionwise_aggregate  | off
 enable_partitionwise_join       | off
```


Параметры конфигурации планировщика

Два параметра конфигурации, по умолчанию, отключают оптимизации:

```
\dconfig enable_part*
    List of configuration parameters
           Parameter             | Value
---------------------------------+------
 enable_partition_pruning        | on
 enable_partitionwise_aggregate  | off
 enable_partitionwise_join       | off
```

Параметр `enable_partition_pruning` позволяет планировщику исключать секции как из формируемого плана, так и на этапе выполнения.

Параметр `enable_partitionwise_join` позволяет планировщику соединять секции секционированных таблиц. Соединение секций применяется только, когда в условии соединения указаны все столбцы, входящие в ключи секционирования таблиц. Типы ключевых столбцов таблиц должны быть одного типа. Секции таблиц должны соответствовать один к одному, то есть иметь одинаковый метод секционирования и одинаковые границы секций.

Параметр `enable_partitionwise_aggregate` позволяет планировщику выполнять группировку отдельно для каждой секции. Если в выражении `GROUP BY` включены не все столбцы из ключа секционирования, то на уровне секций группировка выполняется частично.

Параметры отключены потому, что на этапе планирования и выполнения используется больше памяти, объем которой линейно зависит от числа секций. В базах, где секционирование используется бездумно, секций много. Распространена практика, когда секции хранят данные за сутки, а не за год. На этапе планирования используется больше процессорного времени. Память выделяется в локальной памяти серверного процесса, ограничиваемой параметром конфигурации `work_mem`. Если секций не много (как и должно быть), то параметры рекомендуется включить.

Параметр `constraint_exclusion` не влияет на использование декларативно секционированных таблиц.

https://docs.tantorlabs.ru/tdb/ru/18_3/se/runtime-config-query.html