



# Основные объекты БД



Таблицы, типы данных и ограничения целостности

Последовательности

Индексы

Функции пользователя

Триггеры и правила

Секционирование

Представления

Материализованные представления

Обычные

Нежурналируемые

- не происходит запись в WAL

- невозможно восстановление при сбое (init-файл)

Временные

- существуют на время сеанса или транзакции

- не журналируются

- не попадают в общий буферный кэш

```
pg_class where relkind='r'  
pg_tables
```

```
pg_attribute  
pg_attrdef
```

```
\d (\dt)
```

## Числовые

целые, с плавающей и фиксированной запятой, денежный

## Символьные

с фиксированной и переменной длиной

## Двоичные

байтовые и битовые массивы

## Даты

даты с временной зоной и без, время, интервалы

## Логический

трехзначная логика (с null)

## Перечислимые

## Геометрические

точка, прямая, отрезок, прямоугольник, ломаная, многоугольник, круг

## Массивы

в том числе многомерные

## Составные (записи)

## Диапазоны

## Сложносоставные

XML, JSON

...

pg\_type            \dT  
pg\_enum

Первичный ключ (primary key)

Уникальность (unique)

Внешний ключ (foreign key)

Обязательность (not null)

Проверка (check)

на уровне одной строки

Исключение (exclude)

на уровне всех строк

Проверка выполняется сразу или в конце транзакции

`pg_constraint`

## Генерация номеров

гарантируется уникальность

не гарантируется строгая последовательность

## Реализация

специальная однострочная таблица

## Кэширование значений

на уровне сеансов

```
pg_class where relkind='S'      \d (\ds)
```

Используются для быстрого доступа к данным

## Типы индексов

B-tree	сбалансированное дерево используется ограничением уникальности
GiST	обобщенное сбалансированное дерево
SP-GiST	обобщенное несбалансированное дерево
GIN	инвертированный список

## Вариации

индексы по нескольким столбцам  
индексирование выражений  
частичные индексы

```
pg_class where relkind='i'      \di
pg_index
```



Языки программирования

чистый SQL

Си (статически или динамически подключаемые)

процедурные языки (PL/pgSQL, PL/Python, PL/Perl, PL/Tcl и др.)

Перегруженные функции

Агрегатные и оконные функции

pg\_language            \df  
pg\_proc

Функции, срабатывающие при DML-командах  
на любом языке, кроме SQL

Объекты  
таблицы и представления

Действия  
вставка, изменение, удаление, очистка  
до, после, или вместо  
для всей операции или для каждой строки

```
pg_trigger      \df
pg_proc
```

Функции, срабатывающие при DDL-командах  
на любом языке, кроме SQL

События

ALTER, CREATE, DROP

перед или после команды

```
pg_event_trigger      \dy  
pg_proc
```

Наследуются столбцы

включая ограничения целостности (не связанные с индексами)

Допускается множественное наследование

При изменении родительской таблицы

изменения спускаются по иерархии

удаление не приводит к удалению дочерних таблиц

При обращении к родительской таблице

select, update, delete обходят всю иерархию

(можно ограничить только родительской таблицей)

insert вставляет строки только в указанную таблицу

`pg_inherits`

Разделение таблицы на части для более эффективного управления и доступа к данным

## Наследование

секции — дочерние таблицы

при запросе к родительской таблице, дочерние также учитываются

## Ограничения проверки (check)

определяют условия на данные внутри каждой секции

учитываются оптимизатором при выборе плана запроса, чтобы не перебирать лишние секции

## Триггер для вставки

данные вставляются в родительскую таблицу,

триггер переносит их в одну из дочерних

должен быть согласован с ограничениями на секциях

Правила для переписывания запроса  
подставляют один запрос вместо другого  
применяются и при вставках, изменениях и удалениях

Правила для вставки, изменения и удаления  
подменяют действие или добавляют новые

`pg_rewrite`

## Представления

хранимый текст запроса

текст подставляется в другие запросы с помощью правил

## Материализованные представления

результат запроса вычисляется и сохраняется в таблице

для получения данных используется готовая таблица

обновление таблицы происходит по требованию

```
pg_class where relkind='v'      \d (\dv)
pg_class where relkind='m'      \d (\dm)
```

# Демонстрация



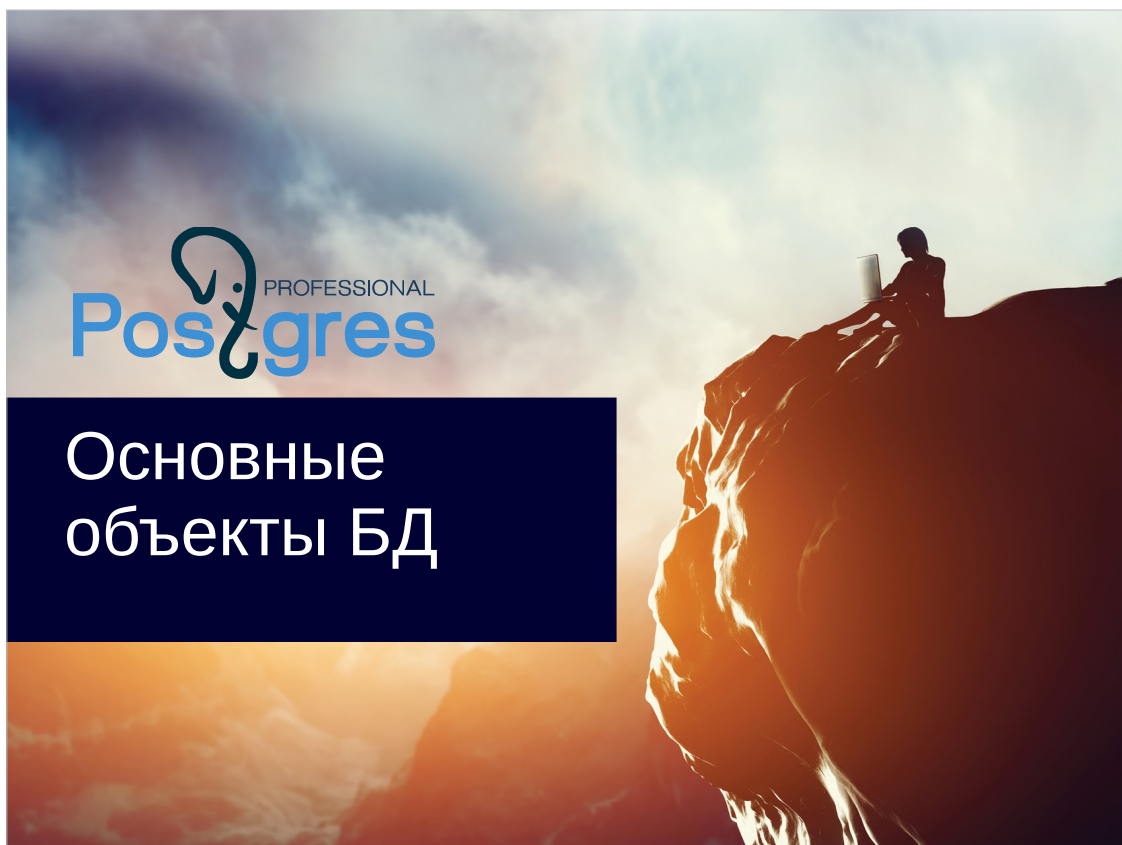


Познакомились с основными объектами БД

Рассмотрели примеры их использования

Узнали про соответствующие таблицы системного каталога

1. В базе данных DV8 создать таблицу observations для ведения наблюдений за температурой.  
Таблица должна содержать поля:
  - date\_taken — дата измерений, не пустое
  - temp — температура, от -60°C до +60°C
2. Добавить в таблицу несколько произвольных записей; убедиться, что ограничения действуют.
3. Создать представление observ\_fahrenheit, показывающее температуру в градусах Фаренгейта ( $^{\circ}\text{F} = ^{\circ}\text{C} \cdot 9/5 + 32$ ).
4. Проиндексировать таблицу по дате измерений.



### **Авторские права**

Курс «Администрирование PostgreSQL 9.4. Базовый курс» разработан в компании Postgres Professional (2015 год).

Авторы: Егор Рогов, Павел Лузанов

### **Использование материалов курса**

Некоммерческое использование материалов курса (презентации, демонстрации) разрешается без ограничений. Коммерческое использование возможно только с письменного разрешения компании Postgres Professional. Запрещается внесение изменений в материалы курса.

### **Обратная связь**

Отзывы, замечания и предложения направляйте по адресу:  
[edu@postgrespro.ru](mailto:edu@postgrespro.ru)

### **Отказ от ответственности**

Компания Postgres Professional не несет никакой ответственности за любые повреждения и убытки, включая потерю дохода, нанесенные прямым или косвенным, специальным или случайным использованием материалов курса. Компания Postgres Professional не предоставляет каких-либо гарантий на материалы курса. Материалы курса предоставляются на основе принципа «как есть» и компания Postgres Professional не обязана предоставлять сопровождение, поддержку, обновления, расширения и изменения.

Таблицы, типы данных и ограничения целостности

Последовательности

Индексы

Функции пользователя

Триггеры и правила

Секционирование

Представления

Материализованные представления

В презентации содержатся основные теоретические сведения.  
Практическое применение рассматривается в демонстрации.

## Обычные

### Нежурналируемые

не происходит запись в WAL

невозможно восстановление при сбое (init-файл)

### Временные

существуют на время сеанса или транзакции

не журналируются

не попадают в общий буферный кэш

```
pg_class where relkind='r'  
pg_tables
```

```
pg_attribute  
pg_attrdef
```

```
\d (\dt)
```

Таблицы — основной объект базы данных.

Кроме обычных таблиц, с которыми мы уже встречались в предыдущих темах, поддерживаются две их разновидности.

Таблица может быть нежурналируемой. Изменения, связанные с такой таблицей, не будут записываться в журнал упреждающей записи. С одной стороны, это позволяет обходиться меньшими накладными расходами.

С другой — в случае сбоя восстановить содержимое таблицы будет невозможно. Для нежурналируемых таблиц хранится дополнительный файл «\_init», который соответствует пустой таблице. При сбое этот файл просто копируется поверх основного, затирая его содержимое. Аналогично обстоит дело и с индексами, созданными для нежурналируемой таблицы.

Временные таблицы предназначены для хранения данных, которые должны быть доступны только текущему сеансу (и только на время его жизни, или даже на время текущей транзакции).

Временные таблицы также являются нежурналируемыми. Кроме того, страницы таких таблиц не попадают в общий буферный кэш.

Внизу этого и следующих слайдов приведены имена таблиц словаря данных, в которых можно найти информацию о рассматриваемых объектах.

<http://www.postgresql.org/docs/current/static/ddl-basics.html>

## Числовые

целые, с плавающей и фиксированной запятой, денежный

## Символьные

с фиксированной и переменной длиной

## Двоичные

байтовые и битовые массивы

## Даты

даты с временной зоной и без, время, интервалы

## Логический

трехзначная логика (с null)

## Перечислимые

Столбцы таблиц должны иметь определенный тип данных. Тип определяет, какие значения могут храниться в этом столбце и какие операции определены над этими значениями.

Поддерживается множество различных типов, большая часть которых приведена на этом и следующем слайдах.

<http://www.postgresql.org/docs/current/static/datatype.html>

<http://www.postgresql.org/docs/current/static/functions.html>

## Геометрические

точка, прямая, отрезок, прямоугольник, ломаная, многоугольник, круг

## Массивы

в том числе многомерные

## Составные (записи)

## Диапазоны

## Сложносоставные

XML, JSON

...

pg\_type        \dT  
pg\_enum

Геометрические типы могут, например, использоваться в геоинформационных системах, а полноценная поддержка XML и JSON позволяет использовать PostgreSQL в области NoSQL, не жертвуя при этом ни эффективностью, ни транзакционностью.

Первичный ключ (primary key)

Уникальность (unique)

Внешний ключ (foreign key)

Обязательность (not null)

Проверка (check)

на уровне одной строки

Исключение (exclude)

на уровне всех строк

Проверка выполняется сразу или в конце транзакции

`pg_constraint`

Для таблиц могут быть заданы ограничения целостности. Большая часть ограничений задается для одного определенного поля таблицы, но некоторые можно задать и для строки или даже целой таблицы.

Ограничения могут проверяться постоянно, либо откладываться до конца транзакции. В любом случае PostgreSQL гарантирует свойство согласованности: транзакция должна переводить базу данных из одного целостного состояния в другое.

<http://www.postgresql.org/docs/current/static/ddl-constraints.html>



## Генерация номеров

гарантируется уникальность

не гарантируется строгая последовательность

## Реализация

специальная однострочная таблица

## Кэширование значений

на уровне сеансов

```
pg_class where relkind='S'      \d (\ds)
```

7

Последовательности удобно использовать для генерации уникальных номеров, в частности, для первичных ключей таблиц (для этого случая удобно воспользоваться псевдотипом `serial` или `bigserial` — при этом создается последовательность и поле будет автоматически получать из нее значения).

Внутренняя реализация последовательностей использует специальную однострочную таблицу, которая хранит текущий номер.

В общем случае последовательность гарантирует только уникальность номеров. Если разрешить кэширование, то каждый сеанс будет получать сразу диапазон значений и выдавать их по одному; при этом параллельные сеансы будут выдавать номера не в общем порядке. Таким образом повышается эффективность, так как обращение к таблице происходит не каждый раз.

<http://www.postgresql.org/docs/current/static/functions-sequence.html>

Используются для быстрого доступа к данным

## Типы индексов

B-tree	сбалансированное дерево используется ограничением уникальности
GiST	обобщенное сбалансированное дерево
SP-GiST	обобщенное несбалансированное дерево
GIN	инвертированный список

## Вариации

- индексы по нескольким столбцам
- индексирование выражений
- частичные индексы

```
pg_class where relkind='i'      \di
pg_index
```

8

Индексы используются для получения быстрого доступа к (обычно) небольшой части имеющихся данных. Некоторые типы индексов выдают упорядоченные значения и могут также использоваться для сортировки.

Поддерживаются различные типы индексов. Индексы B-tree являются стандартными и наиболее распространенными в различных СУБД. Этот тип поддерживает ограничение уникальности; соответствующий индекс создается автоматически при объявлении первичного или уникального ключа.

Индексы могут строиться как по одному, так и нескольким столбцам. Кроме того, можно индексировать не столбец, а выражение. Частичные индексы позволяют индексировать не все строки, уменьшая таким образом размер и увеличивая эффективность.

Индексные типы GiST, SP-GiST и GIN являются обобщенными решениями, позволяющими на их основе строить конкретные индексы для различных структур данных.

<http://www.postgresql.org/docs/current/static/indexes.html>

## Языки программирования

чистый SQL

Си (статически или динамически подключаемые)

процедурные языки (PL/pgSQL, PL/Python, PL/Perl, PL/Tcl и др.)

## Перегруженные функции

## Агрегатные и оконные функции

```
pg_language      \df
pg_proc
```

Хранимые функции позволяют пользователям расширить набор имеющихся встроенных функций.

Функции могут быть написаны как на SQL, так и на ряде процедурных языков. Там, где важна особая эффективность, можно разработать функцию на языке Си.

Поддерживается перегрузка функций, то есть допускаются функции с одинаковыми именами, отличающиеся типом и числом параметров.

Можно создавать и собственные агрегатные и оконные функции.

<http://www.postgresql.org/docs/current/static/xfunc.html>

Функции, срабатывающие при DML-командах  
на любом языке, кроме SQL

Объекты  
таблицы и представления

Действия  
вставка, изменение, удаление, очистка  
до, после, или вместо  
для всей операции или для каждой строки

```
pg_trigger    \df
pg_proc
```

10

Триггеры — это функции, которые срабатывают при определенных действиях.

Триггер может быть задан для таблицы (в том числе внешней; эта тема рассматривается в курсе DBA2) или представления. Он срабатывает при вставке, изменении, удалении или очистке.

Триггер может срабатывать как перед действием, так и после или даже вместо него; а также для каждой строки или только один раз для всей операции.

Триггеры могут (с известной осторожностью и аккуратностью) использоваться для обеспечения таких ограничений целостности, которые невозможно реализовать декларативно.

<http://www.postgresql.org/docs/current/static/triggers.html>

Функции, срабатывающие при DDL-командах  
на любом языке, кроме SQL

## События

ALTER, CREATE, DROP

перед или после команды

```
pg_event_trigger    \dy
pg_proc
```

11

Событийные триггеры аналогичны обычным триггерам, но не связаны с какой-либо таблицей или представлением. Они срабатывают при событиях DDL (изменяющих определение данных), например, при выполнении команд `create index`, `alter table`, `drop type` и т. п.

<http://www.postgresql.org/docs/current/static/event-triggers.html>

Наследуются столбцы

включая ограничения целостности (не связанные с индексами)

Допускается множественное наследование

При изменении родительской таблицы

изменения спускаются по иерархии

удаление не приводит к удалению дочерних таблиц

При обращении к родительской таблице

select, update, delete обходят всю иерархию

(можно ограничить только родительской таблицей)

insert вставляет строки только в указанную таблицу

pg\_inherits

12

PostgreSQL является объектно-реляционной СУБД. В частности, это проявляется в наличие механизма наследования таблиц.

При создании таблицы можно указать, что она наследует другой (родительской) таблице. При этом дочерняя таблица получит все столбцы из родительской таблицы. Допускается и множественное наследование.

Наследственную связь можно задать для уже существующих таблиц, если у дочерней таблицы есть как минимум все столбцы родительской. Заданную связь можно в любой момент разорвать.

При изменении родительской таблицы изменения спускаются по иерархии наследования к дочерним таблицам.

При обращении к родительской таблице в запросе, данные выбираются как из нее, так и из всех дочерних таблиц (если явно не указано обратное). Однако вставка данных всегда происходит точно в указанную таблицу.

<http://www.postgresql.org/docs/current/static/ddl-inherit.html>

Разделение таблицы на части для более эффективного управления и доступа к данным

## Наследование

секции — дочерние таблицы

при запросе к родительской таблице, дочерние также учитываются

## Ограничения проверки (check)

определяют условия на данные внутри каждой секции

учитываются оптимизатором при выборе плана запроса, чтобы не перебирать лишние секции

## Триггер для вставки

данные вставляются в родительскую таблицу,

триггер переносит их в одну из дочерних

должен быть согласован с ограничениями на секциях

13

Практическое применение наследования — реализация секционирования.

Секции создаются как дочерние таблицы.

Для дочерних таблицы задаются ограничения проверки (check constraint). При обращении в запросе к родительской таблице данные должны выбираться из всех дочерних таблиц. Но оптимизатор учитывает ограничения, чтобы не заглядывать в те секции, где данных точно не может быть.

Чтобы данные попадали в нужную секцию, для родительской таблицы задается триггер, перенаправляющий вставку в нужную секцию.

Используя эти механизмы, можно реализовать практически любой вид секционирования, включая секционирование по списку, интервалу или хэшу, а также подсекции.

Однако настройка секционирования требует значительных ручных действий, поэтому существует расширение `pg_partman` (не входящее в стандартную поставку), облегчающее эту задачу.

<http://www.postgresql.org/docs/current/static/ddl-partitioning.html>

## Правила для переписывания запроса

подставляют один запрос вместо другого

применяются и при вставках, изменениях и удалениях

## Правила для вставки, изменения и удаления

подменяют действие или добавляют новые

`pg_rewrite`

14

Правила — механизм, позволяющий переписать запрос в ходе его выполнения (на этапе переписывания, после разбора и перед оптимизацией).

Правила позволяют переписать запрос `select`, а также целиком подменить запрос `insert`, `update`, `delete` или добавить к нему дополнительные запросы.

С помощью правил можно реализовать некоторые действия, которые можно сделать с помощью триггеров, но это значительно более сложный способ.

<http://www.postgresql.org/docs/current/static/rules.html>



## Представления

хранимый текст запроса

текст подставляется в другие запросы с помощью правил

## Материализованные представления

результат запроса вычисляется и сохраняется в таблице

для получения данных используется готовая таблица

обновление таблицы происходит по требованию

```
pg_class where relkind='v'      \d (\dv)
pg_class where relkind='m'      \d (\dm)
```

15

Правила в PostgreSQL используются, в частности, для реализации представлений: имя представления заменяется текстом запроса, после чего оптимизируется уже общий запрос.

Представления могут использоваться, чтобы предоставить удобный доступ к «низлежащим» таблицам или ввести ограничение доступа.

Материализованное представление отличается от обычного тем, что при его создании запрос выполняется и его результат сохраняется в специальную таблицу. При обращении к материализованному представлению обращение фактически происходит к заранее подготовленной таблице.

Обновление материализованного представления происходит по требованию.

Использование материализованных представлений может существенно улучшить производительность, но следует иметь в виду устаревание собранных данных, накладные расходы на их пересчет и необходимость выполнять такой пересчет на периодической основе.

<http://www.postgresql.org/docs/current/static/rules-views.html>

<http://www.postgresql.org/docs/current/static/rules-materializedviews.html>



Познакомились с основными объектами БД

Рассмотрели примеры их использования

Узнали про соответствующие таблицы системного каталога

1. В базе данных DB8 создать таблицу observations для ведения наблюдений за температурой. Таблица должна содержать поля:
  - date\_taken — дата измерений, не пустое
  - temp — температура, от -60°C до +60°C
2. Добавить в таблицу несколько произвольных записей; убедиться, что ограничения действуют.
3. Создать представление observ\_fahrenheit, показывающее температуру в градусах Фаренгейта ( $^{\circ}\text{F} = ^{\circ}\text{C} \cdot 9/5 + 32$ ).
4. Проиндексировать таблицу по дате измерений.

## Решение

```
# create database db8;
# \c db8
# create table observations(
  date_taken date not null,
  temp numeric check( temp between -60 and 60 )
);

# insert into observations values (current_date, 25),
  (current_date-1, 22);
# insert into observations values (current_date, 70);
-- new row for relation "observations" violates check constraint
  "observations_temp_check"
# insert into observations values (null, 20);
-- null value in column "date_taken" violates not-null constraint

# create view observ_fahrenheit as
  select date_taken, temp as celsius, round(temp*5/9+32) as
    fahrenheit from observations;

# create index observations_date on observations(date_taken);
```