Проектирование баз данных

2 семестр

Содержание курса

- Раздел 4. Объектные и объектно-реляционные базы данных
- Тема 4.1. Выбор типа СУБД применительно к особенностям предметной области
- *Тема 4.2. Объектно-реляционные базы данных*
- Тема 4.3. Объектные базы данных
- Раздел 5. NoSQL базы данных
- Тема 5.1. Базы данных «Ключ-значение»
- *Тема 5.2. Документные базы данных*
- Тема 5.3. Графовые базы данных
- Тема 5.4. Базы данных «Семейство столбцов»

Объектно-реляционные СУБД

Объектно-реляционные СУБД

Достоинства

- *повторное* и *совместное* использование компонентов
- расширенный реляционный подход позволяет воспользоваться обширным объемом накопленных знаний и опыта, связанных с разработкой реляционных приложений

Недостатки

- сложность и связанные с ней дополнительные расходы.
- объектно-реляционных системах искажается объектная терминология
- Объекты фактически не являются очередным расширением понятия данных, поскольку представляют совершенно другую концепцию, с большим потенциалом выражения связей и правил поведения объектов реального мира

Манифесты баз данных третьего поколения (1990)

- 1. Наличие развитой системы типов.
- 2. Поддержка механизма наследования.
- 3. Поддержка функций, включая процедуры и методы базы данных, а также механизма инкапсуляции.
- 4. Уникальные идентификаторы для записей должны присваиваться средствами СУБД только в том случае, когда нельзя использовать определяемые пользователем первичные ключи.
- 5. Правила (триггеры и ограничения) станут важнейшим компонентом будущих систем. Они не должны быть связаны с какой-то конкретной функцией или коллекцией.
- 6. Очень важно, чтобы все программные способы доступа к базе данных осуществлялись с помощью непроцедурного языка высокого уровня.
- 7. Должны существовать по меньшей мере два способа определения коллекций: один на основе перечисления элементов коллекции, а другой — с использованием языка запросов для определения принадлежности элемента к коллекции.
- 8. Важным является наличие обновляемых представлений.
- 9. Средства измерения производительности не должны иметь никакого отношения к моделям данных и не должны в них присутствовать.
- 10. СУБД третьего поколения должны быть доступны из многих языков высокого уровня.
- 11. Желательно, чтобы во многих языках программирования высокого уровня поддерживались конструкции, обеспечивающие доступ к перманентным данным. Эта поддержка должна обеспечиваться на основе применения каждой отдельной СУБД с помощью дополнительных средств компилятора и сложной системы поддержки выполнения программ.
- 12. Плохо ли это или хорошо, но язык SQL должен оставаться "межгалактическим языком работы с данными".
- 13. Запросы и ответы на них должны составлять самый нижний уровень взаимодействия клиента и сервера.

Пример наследования Posgres

- CREATE TABLE cities
  (name text,
  population real,
  altitude int -- (высота в футах)
  );
- CREATE TABLE capitals
  (state char(2))
  INHERITS (cities);

- SELECT name, altitude FROM cities WHERE altitude > 500;
- SELECT name, altitude FROM ONLY cities WHERE altitude > 500;

Перечень объектно-реляционных СУБД

- CUBRID
- Oracle с 8 версии
- OpenLink Virtuoso
- PostgreSQL
- Informix

Объектно-ориентированные возможности PL/SQL (Oracle)

Средства и возможности	8.0	8.1	9.1	9.2 и выше	11g и выше
Абстрактные типы данных как равноправные сущности базы данных	*	*	*	*	*
Абстрактные типы данных как параметры PL/SQL	*	*	*	*	*
Атрибуты с типами коллекций	*	*	*	*	*
Атрибуты типа REF для навигации по объектам внутри базы данных	*	*	*	*	*
Реализация логики методов на PL/SQL и C	*	*	*	*	*
Определяемая программистом семантика сравнения объектов	*	*	*	*	*
Представление реляционных данных как данных объектных типов	*	*	*	*	*
Статический полиморфизм (перегрузка методов)	*	*	*	*	*
Возможность «эволюции» типа путем модификации логики существующих методов или добавления новых методов	*	*	*	*	*
Реализация логики методов на языке Java		*	*	*	*
«Статические» методы (уровня класса, а не уровня экземпляра)		*	*	*	*
Возможность использования первичного ключа как идентификатора хранимого объекта, обеспечивающая декларативную целостность REF-ссылок		*	*	*	*
Наследование атрибутов и методов от пользовательских типов			*	*	*
Динамическая диспетчеризация методов			*	*	*
Супертипы, для которых не могут создаваться экземпляры (аналоги абстрактных классов языка Java)			*	*	*
Возможность эволюции типа путем удаления методов (и добавления для изменения сигнатуры)			*	*	*

Объектно-ориентированные возможности PL/SQL

Средства и возможности	8.0	8.1	9.1	9.2 и выше	11g и выше
Возможность эволюции типа путем добавления и удаления атрибутов, автоматическое распространение изменений на связанные структуры физической базы данных			*	*	*
«Анонимные» типы: ANYTYPE, ANYDATA, ANYDATASET			*	*	*
Операторы понижающего преобразования (TREAT) и определения типа (IS OF), доступные в SQL			*	*	*
Операторы TREAT и IS OF в PL/SQL				*	*
Определяемые пользователем конструкторы				*	*
Вызовы методов супертипа в производном типе					*
Приватные атрибуты, переменные, константы и методы					
Наследование от нескольких супертипов					
Совместное использование объектных типов или экземпляров в распределенных базах данных без обращения к объектным представлениям					

https://oracle-patches.com/db/sql/3940-объектно-ориентированные-возможности-pl-sql

Объектные возможности ОР СУБД

- определение новых базовых классов (типов);
- определение новых составных классов (типов) на базе существующих;
- введение новых функций, работающих как с предопределенными типами и классами данных, так и с новыми;
- наследование на уровне классов;
- наследование на уровне таблиц;
- обеспечение инкапсуляции для типов;

Процедуры и функции postgres

- **Функции на языке запросов (функции, написанные на SQL)**
- **Функции на процедурных языках (функции, написанные, например, на PL/pgSQL или PL/Tcl)**

PostgreS позволяет разрабатывать собственные функции и на языках, отличных от SQL и C.Эти другие языки в целом обычно называются *процедурными языками* (PL, Procedural Languages).Процедурные языки не встроены в сервер PostgreSQL; они предлагаются загружаемыми модулями.

- **Внутренние функции**

Внутренние функции — это функции, написанные на языке C, и статически скомпонованные в исполняемый код сервера PostgreSQL. В «теле» определения функции задаётся имя функции на уровне C, которое не обязательно должно совпадать с именем, объявленным для использования в SQL.

- **Функции на языке C**

функции «на языке C» от «внутренних» функций отличает метод динамической загрузки— правила написания кода по сути одни и те же.

- Функции **на языке C** компилируются в динамически загружаемые объекты (также называемые разделяемыми библиотеками) и загружаются сервером по требованию. PostgreS не будет компилировать функцию на C автоматически, поэтому прежде чем ссылаться на объектный файл в команде CREATE FUNCTION, его нужно скомпилировать.

Особенности процедур и функций postgres

- Функции вызываются как часть запроса или команды DML, а процедуры вызываются отдельно командой CALL.
- Процедура, в отличие от функции, может фиксировать или откатывать транзакции во время её выполнения (а затем автоматически начинать новую транзакцию), если вызывающая команда CALL находится не в явном блоке транзакции.
- К аргументам SQL-функции можно обращаться в теле функции по именам или номерам.
- Чтобы использовать имя, объявите аргумент функции как именованный, а затем просто пишится это имя в теле функции. Если имя аргумента совпадает с именем какого-либо столбца в текущей SQL-команде внутри функции, имя столбца будет иметь приоритет. Чтобы всё же перекрыть имя столбца, необходимо дополнить имя аргумента именем самой функции, то есть записать его в виде
- *имя_функции.имя_аргумента*.
- Функции SQL могут быть объявлены как принимающие переменное число аргументов, с условием, что все «необязательные» аргументы имеют один тип данных. Необязательные аргументы будут переданы такой функции в виде массива. Для этого в объявлении функции последний параметр помечается как VARIADIC; при этом он должен иметь тип массива.

Строковые константы

Строковая константа, заключённая в доллары, начинается со знака доллара ($), необязательного «тега» из нескольких символов и ещё одного знака доллара, затем содержит обычную последовательность символов, составляющую строку, и оканчивается знаком доллара, тем же тегом и замыкающим знаком доллара. Например, строку «Жанна д'Арк» можно записать в долларах двумя способами:

- $$Жанна д'Арк$$
- $SomeTag$Жанна д'Арк$SomeTag$

Функции и процедуры, написанные на SQL

- SQL-функции выполняют произвольный список операторов SQL и возвращают результат последнего запроса в списке.
- Тело SQL-функции должно представлять собой список SQL-операторов, разделённых точкой с запятой. Точка с запятой после последнего оператора может отсутствовать. Если только функция не объявлена как возвращающая void, последним оператором должен быть SELECT, либо INSERT, UPDATE или DELETE с предложением RETURNING.

```
CREATE FUNCTION clean_student() RETURNS void AS '
DELETE FROM student
WHERE id_gr is null;
' LANGUAGE SQL;
```

- SELECT clean_student ();

```
CREATE PROCEDURE clean_student () AS '
DELETE FROM student
WHERE id_gr is null;
' LANGUAGE SQL;
```

- CALL clean_student();

Обращение к аргументам функций

```
CREATE FUNCTION add_em(x integer, y integer) RETURNS integer AS $$
SELECT x + y;
$$ LANGUAGE SQL;
SELECT add_em(1, 2) AS answer;
```

- answer
- --------
- 3

```
CREATE FUNCTION add_em(integer, integer) RETURNS integer AS $$
SELECT $1 + $2;
$$ LANGUAGE SQL;
SELECT add_em(1, 2) AS answer;
```

- answer
- --------
- 3

Создание процедуры

CREATE [OR REPLACE] PROCEDURE

имя ([[*режим_аргумента*] [*имя_аргумента*] *тип_аргумента* [{ DEFAULT |= } *выражение_по_умолчанию*] [, ...]])

{ LANGUAGE *имя_языка*

| TRANSFORM { FOR TYPE *имя_типа* } [, ...]

| [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER

| SET *параметр_конфигурации* { TO *значение* | = *значение* | FROM CURRENT }

| AS '*определение*'

| AS '*объектный_файл*', '*объектный_символ*'

| *тело_sql*

} ...

Создание функции

CREATE [OR REPLACE] FUNCTION

имя ([[*режим_аргумента*] [*имя_аргумента*] *тип_аргумента* [{ DEFAULT |= } *выражение_по_умолчанию*] [, ...]])

[RETURNS *тип_результата*

| RETURNS TABLE (*имя_столбца тип_столбца* [, ...])]

{ LANGUAGE *имя_языка*

| TRANSFORM { FOR TYPE *имя_типа* } [, ...]

| WINDOW

| { IMMUTABLE | STABLE | VOLATILE }

| [NOT] LEAKPROOF

| { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }

| { [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER }

| PARALLEL { UNSAFE | RESTRICTED | SAFE }

| COST *стоимость_выполнения*

| ROWS *строк_в_результате*

| SUPPORT *вспомогательная_функция*

| SET *параметр_конфигурации* { TO *значение* | = *значение* | FROM CURRENT }

| AS '*определение*'

| AS '*объектный_файл*', '*объектный_символ*'

| *тело_sql*

} ...

Параметры

- *имя*

Имя создаваемой процедуры/функции (возможно, дополненное схемой).

- *режим_аргумента*

Режим аргумента: IN, OUT, INOUT или VARIADIC. По умолчанию подразумевается IN. За единственным аргументом VARIADIC могут следовать только аргументы OUT. Кроме того, аргументы OUT и INOUT нельзя использовать с предложением RETURNS TABLE.

- *имя_аргумента*

Имя аргумента.

- *тип_аргумента*

Тип данных аргумента процедуры (возможно, дополненный схемой), при наличии аргументов.

Тип аргументов может быть базовым, составным или доменным, либо это может быть ссылка на столбец таблицы.

- *выражение_по_умолчанию*

Выражение, используемое для вычисления значения по умолчанию, если параметр не задан явно. Результат выражения должен сводиться к типу соответствующего параметра. Для всех входных параметров, следующих за параметром с определённым значением по умолчанию, также должны быть определены значения по умолчанию.

- *имя_языка*

Имя языка, на котором реализована процедура. Это может быть sql, c, internal либо имя процедурного языка, определённого пользователем, например, plpgsql. Если присутствует *тело_sql*, подразумевается язык sql. Вариант написания этого имени в апострофах считается устаревшим и требует точного совпадения регистра.

Параметры

- *определение*

Строковая константа, определяющая реализацию процедуры; её значение зависит от языка. Это может быть имя внутренней процедуры, путь к объектному файлу, команда SQL или код на процедурном языке.

- *объектный_файл*, *объектный_символ*

Эта форма предложения AS применяется для динамически загружаемых процедур на языке C, когда имя процедуры в коде C не совпадает с именем процедуры в SQL.

- Строка *объектный_файл* задаёт имя файла, содержащего скомпилированную процедуру на C (данная команда воспринимает эту строку так же, как и LOAD).
- Строка *объектный_символ* задаёт

символ скомпонованной процедуры, то есть имя процедуры в исходном коде на языке C. Если объектный символ не указан, предполагается, что он совпадает с именем определяемой SQL- процедуры. Если повторные вызовы CREATE PROCEDURE ссылаются на один и тот же объектный файл, он загружается в рамках сеанса только один раз. Чтобы выгрузить и загрузить этот файл снова (например, в процессе разработки), начните новый сеанс.

- *тело_sql*

Тело процедуры в стиле LANGUAGE SQL. Это должен быть блок вида

BEGIN ATOMIC

оператор;

оператор;

...

оператор;

END

Удаление функции

DROP FUNCTION [IF EXISTS] *имя* [([[*режим_аргумента*] [*имя_аргумента*] *тип_аргумента* [, ...]])] [, ...][CASCADE | RESTRICT]

- Имя существующей функции (возможно, дополненное схемой). Если список аргументов не указан, имя функции должно быть уникальным в её схеме.
- DROP FUNCTION не учитывает аргументы OUT, так как для идентификации функции нужны только типы входных аргументов.
- Тип данных аргументов функции нужны для удаления перегруженных функций
- Пример:

DROP FUNCTION sqrt(integer);

Пользовательские типы

- Перечисления
- составные типы
- Диапазоны
- Тип-пустышка(заглушка)

- Базовые типы
- массивы

составной тип

- CREATE TYPE имя AS
  ([имя_атрибута тип_данных [
  COLLATE правило_сортировки]
  [, ...]])
- Пример

CREATE TYPE activity AS (
name varchar(50) ,
status complete_status,
created timestamp,
started timestamp,
ended timestamp);

Перечисления

- `CREATE TYPE имя AS ENUM`
  `( [ 'метка' [, ... ] ] )`
- Пример

- CREATE TYPE complete_status AS ENUM ('создано','назначено', 'в работе','на согласовании','выполнено');

Тип-пустышка

- `CREATE TYPE имя`

- Тип-пустышка представляет собой заготовку для типа, который будет определён позже.
- Типы-пустышки необходимы для определения прямых ссылок при создании базовых типов и типов-диапазонов.

диапазон

```
CREATE TYPE имя AS RANGE (
SUBTYPE = подтип
[ , SUBTYPE_OPCLASS = класс_оператора_подтипа ]
[ , COLLATION = правило_сортировки ]
[ , CANONICAL = каноническая_функция ]
[ , SUBTYPE_DIFF = функция_разницы_подтипа ]
[ , MULTIRANGE_TYPE_NAME = имя_мультидиапазонного_типа ]
)
```

Параметры

- Задаваемый для диапазона *подтип* может быть любым типом со связанным классом операторов B-дерева (что позволяет определить порядок значений в диапазоне). Обычно порядок элементов определяет класс операторов B-дерева по умолчанию, но его можно изменить, задав имя другого класса в параметре *класс_операторов_подтипа*. Если подтип поддерживает сортировку и требуется, чтобы значения упорядочивались с нестандартным правилом сортировки, его имя можно задать в параметре *правило_сортировки*.
- Необязательная *каноническая_функция* должна принимать один аргумент определяемого типа диапазона и возвращать значение того же типа. Это используется для преобразования значений диапазона в каноническую форму, когда это уместно
- Создаётся *каноническая_функция* несколько нетривиально, так как она должна быть уже определена, прежде чем можно будет объявить тип-диапазон. Для этого нужно :

1. создать тип-пустышку, который будет заготовкой типа, не имеющей никаких свойств, кроме имени и владельца.
2. Затем можно объявить функцию, для которой тип-пустышка будет типом аргумента и результата,
3. объявить тип-диапазон с тем же именем. При этом тип- пустышка автоматически заменится полноценным типом-диапазоном.

- Необязательная *функция_разницы_подтипа* должна принимать в аргументах два значения типа *подтип* и возвращать значение double precision, представляющее разницу между двумя данными значениями. Хотя эту функцию можно не использовать

функцию канонизации (каноническая)

- Если подтип можно рассматривать как дискретный, а не непрерывный, в команде CREATE TYPE следует также задать *функцию канонизации*. Этой функции будет передаваться значение диапазона, а она должна вернуть равнозначное значение, но, возможно, с другими границами и форматированием.
- Для двух диапазонов, представляющих одно множество значений, например, целочисленные диапазоны [1, 7] и [1, 8), функция канонизации должна выдавать один результат.
- Какое именно представление будет считаться каноническим, не имеет значения — главное, чтобы два равнозначных диапазона, отформатированных по-разному, всегда преобразовывались в одно значение с одинаковым форматированием.
- Помимо исправления формата включаемых/исключаемых границ, функция канонизации может округлять значения границ, если размер шага превышает точность хранения подтипа.
- Например, в типе диапазона для подтипа timestamp можно определить размер шага, равный часу, тогда функция канонизации должна будет округлить границы, заданные, например с точностью до минут, либо вместо этого выдать ошибку.

Диапазон пример

```
CREATE TYPE floatrange AS RANGE (
subtype = float8,
subtype_diff = float8mi
);

CREATE FUNCTION time_subtype_diff(x time, y time) RETURNS float8 AS
'SELECT EXTRACT(EPOCH FROM (x - y))' LANGUAGE sql STRICT IMMUTABLE;

CREATE TYPE timerange AS RANGE (
subtype = time,
subtype_diff = time_subtype_diff
);
SELECT '[11:10, 23:00]'::timerange;
```

Изменение типов

ALTER TYPE *имя* OWNER TO { *новый_владелец* | CURRENT_ROLE | CURRENT_USER | SESSION_USER }

ALTER TYPE *имя* RENAME TO *новое_имя*

ALTER TYPE *имя* SET SCHEMA *новая_схема*

ALTER TYPE *имя* RENAME ATTRIBUTE *имя_атрибута* TO *новое_имя_атрибута* [CASCADE |
RESTRICT]

ALTER TYPE *имя* ADD VALUE [IF NOT EXISTS] *новое_значение_перечисления* [{ BEFORE |AFTER } *соседнее_значение_перечисления*]

ALTER TYPE *имя* RENAME VALUE *существующее_значение_перечисления*
TO *новое_значение_перечисления*

ALTER TYPE *имя* SET (*свойство* = *значение* [, ...])

ALTER TYPE *имя действие* [, ...]

действие :

ADD ATTRIBUTE *имя_атрибута тип_данных* [COLLATE *правило_сортировки*]
[CASCADE |RESTRICT]

DROP ATTRIBUTE [IF EXISTS] *имя_атрибута* [CASCADE | RESTRICT]

ALTER ATTRIBUTE *имя_атрибута* [SET DATA] TYPE *тип_данных*
[COLLATE *правило_сортировки*] [CASCADE | RESTRICT]

Удаление типа

- DROP TYPE [IF EXISTS] *имя* [, ...] [CASCADE | RESTRICT]

- CASCADE

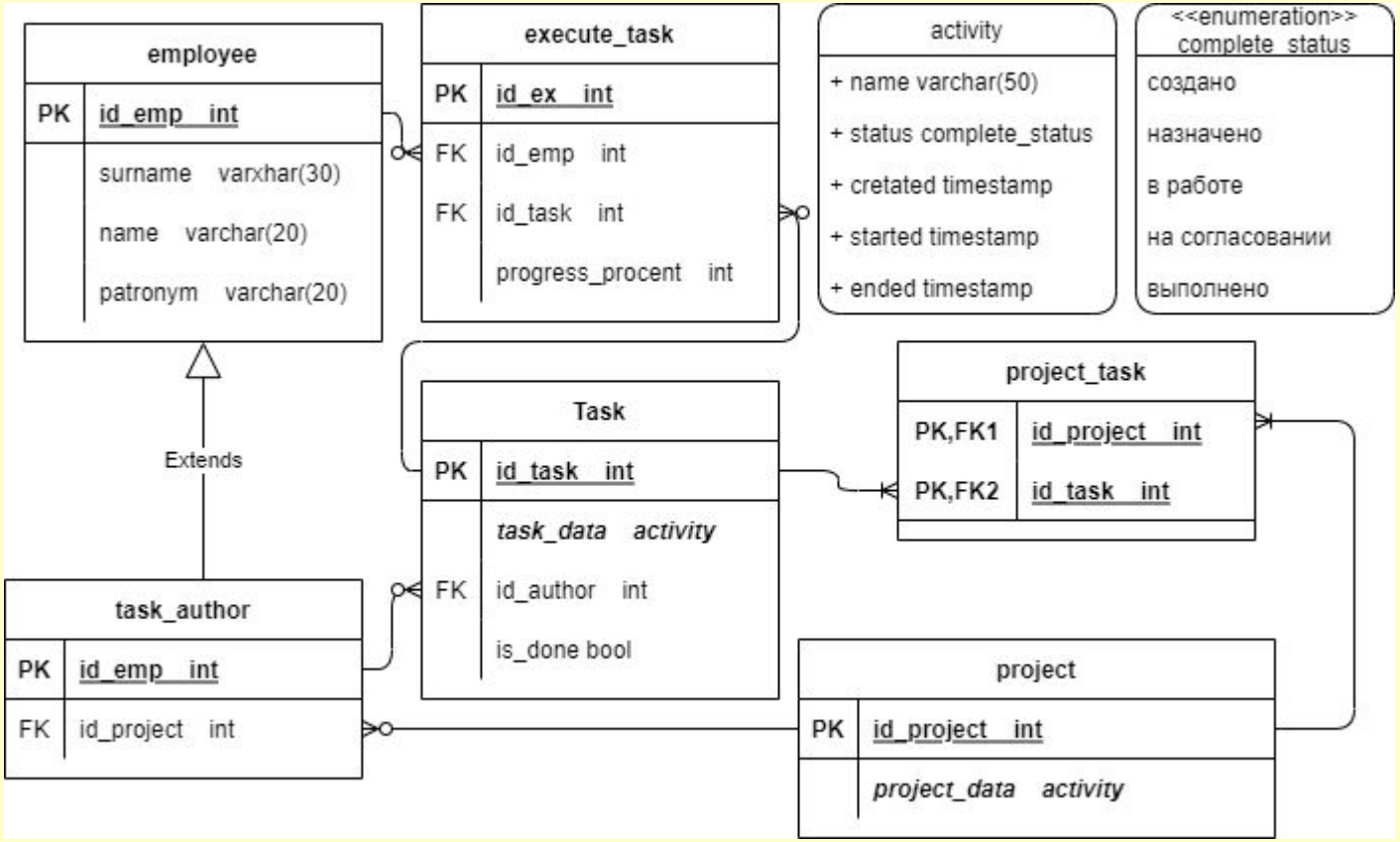
Автоматически удалять объекты, зависящие от данного типа (например, столбцы таблиц, функции и операторы), и, в свою очередь, все зависящие от них объекты/

- RESTRICT

Отказать в удалении типа, если от него зависят какие-либо объекты. Это поведение по умолчанию.

Пример с пользовательскими типами и наследованием

- Отслеживание задач: задачи, проекты, автор и исполнитель задачи, сроки выполнения, статусы людей (выполнение и назначение задач), некоторые не могут назначать задачи (разные статусы)
  a. Задания, в названиях которых содержится слово «интеграция», но оно не последнее
  b. Задача, относящаяся к 2 различным проектам
  c. Задача с самым поздним сроком окончания
  d. Проект с самым большим количеством задач
  e. Человек, у которого нет незавершенных задач



Типы

```
CREATE TYPE complete_status AS ENUM
    ('создано','назначено', 'в работе','на
    согласовании','выполнено');
CREATE TYPE activity AS (
name varchar(50) ,
status complete_status,
created timestamp,
started timestamp,
ended timestamp);
```

Наследование. Пример

```
CREATE TABLE employee (
id_emp serial NOT NULL PRIMARY KEY  ,
  surname varchar(20) not NULL,
  name varchar(15) not NULL,
  patronym varchar(25) DEFAULT NULL
) ;
CREATE TABLE task_author
(
id_proj integer null,
PRIMARY KEY (id_emp),
foreign key (id_proj) references project_(id_proj) on delete cascade
     on update cascade
) INHERITS (employee);
```

```
CREATE TABLE project_
(id_proj serial NOT NULL PRIMARY KEY  ,
  project_data activity NULL);

CREATE TABLE task_ (
id_task integer NOT NULL PRIMARY KEY  ,
task_data activity,
is_done bool,
id_author integer null,
foreign key (id_author) references task_author (id_emp) on delete set null on update cascade);

CREATE TABLE project_task (
id_task integer NOT NULL  ,
id_proj integer NOT NULL ,
PRIMARY KEY (id_task,id_proj),
foreign key (id_proj) references project_(id_proj) on delete cascade on update cascade,
foreign key (id_task) references task_ (id_task)  on delete cascade on update cascade);

CREATE TABLE execute_task (
id_task integer NOT NULL  ,
id_emp integer NOT NULL ,
PRIMARY KEY (id_task,id_emp),
foreign key (id_emp) references employee(id_emp) on delete cascade on update cascade,
foreign key (id_task) references task_ (id_task)  on delete cascade on update cascade);
```

Вставка данных

- INSERT INTO project_(
- id_proj, project_data)
  VALUES (1, ROW('my project name','создано',Now(),Now(),null));

Data Output | Explain | Messages | Notifications

	id_proj [PK] integer	project_data activity
1	1	("my project name",создано,"2022-02-11 00:02:25.823832","2022-02-11 00:02:25.823832",)

INSERT INTO task_author(
id_emp, surname, name, patronym, id_proj)
VALUES (1, 'Иванов', 'Иван', 'Иванович', 1);

Data Output | Explain | Messages | Notifications

	id_emp [PK] integer	surname character varying (20)	name character varying (15)	patronym character varying (25)	id_proj integer
1	1	Иванов	Иван	Иванович	1

типы хранимых процедур в стандарте SQL

- функции
- процедуры
- методы

методы являются функциями, которые привязаны к определению структурированного(составного) типа.

Встроенные методы

- Существует три типа встроенных методов для структурированных типов:
- функция-конструктор,
- Функция-наблюдатель (getter)
- функция-мутатор (setter)

Пользовательские методы Oracle

```
CREATE OR REPLACE TYPE solid_typ AS OBJECT
 ( len INTEGER,
 wth INTEGER,
hgt INTEGER,
MEMBER FUNCTION surface RETURN INTEGER,
 MEMBER FUNCTION volume RETURN INTEGER,
 MEMBER PROCEDURE display (SELF IN OUT NOCOPY solid_typ) );
/
 CREATE OR REPLACE TYPE BODY solid_typ AS
 MEMBER FUNCTION volume RETURN INTEGER IS
 BEGIN
RETURN len * wth * hgt; -- RETURN SELF.len * SELF.wth * SELF.hgt;
END;
 MEMBER FUNCTION surface RETURN INTEGER IS
 BEGIN -- not necessary to include SELF in following line
 RETURN 2 * (len * wth + len * hgt + wth * hgt);
 END;
MEMBER PROCEDURE display (SELF IN OUT NOCOPY solid_typ) IS
BEGIN DBMS_OUTPUT.PUT_LINE('Length: ' || len || ' - ' || 'Width: ' || wth || ' - ' || 'Height: ' || hgt);
      DBMS_OUTPUT.PUT_LINE('Volume: ' || volume || ' - ' || 'Surface area: ' || surface);
END; END; /
```

Пользовательские операторы

CREATE OPERATOR *имя* (
{FUNCTION|PROCEDURE} = *имя_функции*
[, LEFTARG = *тип_слева*]
[, RIGHTARG = *тип_справа*]
[, COMMUTATOR = *коммут_оператор*]
[, NEGATOR = *обратный_оператор*]
[, RESTRICT = *процедура_ограничения*]
[, JOIN = *процедура_соединения*]
[, HASHES] [, MERGES]
)

Параметры создания пользовательского оператора

имя
Имя определяемого оператора.
имя_функции
Функция, реализующая этот оператор.
тип_слева
Тип данных левого операнда оператора, если он есть. Этот параметр опускается для
префиксных операторов.
тип_справа
Тип данных правого операнда оператора.
коммут_оператор
Оператор, коммутирующий для данного.
обратный_оператор
Оператор, обратный для данного.
процедура_ограничения
Функция оценки избирательности ограничения для данного оператора.
процедура_соединения
Функция оценки избирательности соединения для этого оператора.
HASHES
Показывает, что этот оператор поддерживает соединение по хешу.
MERGES
Показывает, что этот оператор поддерживает соединение слиянием.

Имя оператора

Имя оператора образует последовательность из нескольких символов (не более чем NAMEDATALEN-1,
по умолчанию 63) из следующего списка:
+ - * / < > = ~ ! @ # % ^ & | ` ?
Однако выбор имени ограничен ещё следующими условиями:

- Сочетания символов -- и /* не могут присутствовать в имени оператора, так как они будут

обозначать начало комментария.

- Многосимвольное имя оператора не может заканчиваться знаком + или -,

если только оно не содержит также один из этих символов:
~ ! @ # % ^ & | ` ?
Например, @- — допустимое имя оператора, а *- — нет.
• Использование => в качестве имени оператора считается устаревшим и может быть вовсе запрещено в будущих выпусках.
Оператор != отображается в <> при вводе, так что эти два имени всегда равнозначны.

Указанное имя может быть дополнено схемой, например так: CREATE OPERATOR myschema.+ (...).
Если схема не указана, оператор создаётся в текущей схеме. При этом два оператора в одной схеме могут
иметь одно имя, если они работают с разными типами данных. Такое определение операторов называется *перегрузкой*.

Пользовательские операторы. Пример

```
CREATE TYPE complex AS (
        x double precision,
        y double precision
)

CREATE FUNCTION complex_add(complex, complex)
RETURNS complex
AS
'select $1.x+$2.x, $1.y+$2.y'
 LANGUAGE SQL
    IMMUTABLE
    RETURNS NULL ON NULL INPUT;

        CREATE OPERATOR + (
leftarg = complex,
rightarg = complex,
function = complex_add,
commutator = +
);
```

Изменение оператора

ALTER OPERATOR *имя* ({ *тип_слева* | NONE } , *тип_справа*)
OWNER TO { *новый_владелец* | CURRENT_ROLE | CURRENT_USER | SESSION_USER }

ALTER OPERATOR *имя* ({ *тип_слева* | NONE } , *тип_справа*) SET SCHEMA *новая_схема*

ALTER OPERATOR *имя* ({ *тип_слева* | NONE } , *тип_справа*)
SET ({ RESTRICT = { *процедура_ограничения* | NONE }
| JOIN = { *процедура_соединения* | NONE }
} [, ...])

Параметры

имя

Имя существующего оператора (возможно, дополненное схемой).

тип_слева

Тип данных левого операнда оператора; если у оператора нет левого операнда, укажите NONE.

тип_справа

Тип данных правого операнда оператора.

новый_владелец

Новый владелец оператора.

новая_схема

Новая схема оператора.

процедура_ограничения

Функция оценки избирательности ограничения для данного оператора; значение NONE удаляет существующую функцию оценки.

процедура_соединения

Функция оценки избирательности соединения для этого оператора; значение NONE удаляет существующую функцию оценки.

Удаление оператора

- DROP OPERATOR [IF EXISTS] *имя* ({ *тип_слева* | NONE } , *тип_справа*) [, ...] [CASCADE | RESTRICT]
- **Параметры**

IF EXISTS

Не считать ошибкой, если оператор не существует. В этом случае будет выдано замечание.

имя

Имя существующего оператора (возможно, дополненное схемой).

тип_слева

Тип данных левого операнда оператора; если у оператора нет левого операнда, укажите NONE.

тип_справа

Тип данных правого операнда оператора.

CASCADE

Автоматически удалять объекты, зависящие от данного оператора (например, использующие

его представления), и, в свою очередь, все зависящие от них объекты.

RESTRICT

Отказать в удалении оператора, если от него зависят какие-либо объекты. Это поведение по

умолчанию.

- **Пример**

пользовательская агрегатная функция

Агрегатные функции в PostgreSQL определяются в терминах *значений состояния и функций перехода состояния.*

То есть агрегатная функция работает *со значением состояния*, которое

меняется при обработке каждой последующей строки.

Чтобы определить агрегатную функцию, нужно выбрать тип данных для значения состояния, начальное значение состояния и функцию перехода состояния. Функция перехода состояния принимает предыдущее значение состояния и входное агрегируемое значение для текущей строки и возвращает новое значение состояния. Также можно указать *функцию завершения*, на случай, если ожидаемый результат агрегатной функции отличается от данных, которые сохраняются в изменяющемся значении состояния.

Функция завершения принимает конечное значение состояния и возвращает то, что она хочет вернуть в виде результата агрегирования.

Создание агрегатной функции

CREATE [OR REPLACE] AGGREGATE *имя ([режим_аргумента] [имя_аргумента]*
тип_данных_аргумента [, ...]) (

SFUNC = *функция_состояния,*

STYPE = *тип_данных_состояния*

[, SSPACE = *размер_данных_состояния]*

[, FINALFUNC = *функция_завершения]*

[, FINALFUNC_EXTRA]

[, FINALFUNC_MODIFY = { READ_ONLY | SHAREABLE | READ_WRITE }]

[, COMBINEFUNC = *комбинирующая_функция]*

[, SERIALFUNC = *функция_сериализации]*

[, DESERIALFUNC = *функция_десериализации]*

[, INITCOND = *начальное_условие]*

[, MSFUNC = *функция_состояния_движ]*

[, MINVFUNC = *обратная_функция_движ]*

[, MSTYPE = *тип_данных_состояния_движ]*

[, MSSPACE = *размер_данных_состояния_движ]*

[, MFINALFUNC = *функция_завершения_движ]*

[, MFINALFUNC_EXTRA]

[, MFINALFUNC_MODIFY = { READ_ONLY | SHAREABLE | READ_WRITE }]

[, MINITCOND = *начальное_условие_движ]*

[, SORTOP = *оператор_сортировки]*

[, PARALLEL = { SAFE | RESTRICTED | UNSAFE }])

Создание агрегатной функции

CREATE [OR REPLACE] AGGREGATE *имя ([[режим_аргумента] [имя_аргумента*] *тип_данных_аргумента [, ...]]*
ORDER BY [*режим_аргумента] [имя_аргумента*
] *тип_данных_аргумента [, ...]) (*
SFUNC = *функция_состояния,*
STYPE = *тип_данных_состояния*
[, SSPACE = *размер_данных_состояния]*
[, FINALFUNC = *функция_завершения]*
[, FINALFUNC_EXTRA]
[, FINALFUNC_MODIFY = { READ_ONLY | SHAREABLE | READ_WRITE }]
[, INITCOND = *начальное_условие]*
[, PARALLEL = { SAFE | RESTRICTED | UNSAFE }]
[, HYPOTHETICAL]
)

Параметры

- *имя*

Имя создаваемой агрегатной функции (возможно, дополненное схемой).

- *режим_аргумента*

Режим аргумента: IN или VARIADIC. (Агрегатные функции не поддерживают выходные аргументы (OUT).) По умолчанию подразумевается IN. Режим VARIADIC может быть указан только последним.

- *имя_аргумента*

Имя аргумента. В настоящее время используется только в целях документирования. Если опущено, соответствующий аргумент будет безымянным.

- *тип_данных_аргумента*

Тип входных данных, с которым работает эта агрегатная функция. Для создания агрегатной функции без аргументов вставьте * вместо списка с определениями аргументов. (Пример такой агрегатной функции: count(*).)

функция_состояния

Имя функции перехода состояния, вызываемой для каждой входной строки. Для обычных агрегатных функций с *N аргументами, функция_состояния* должна принимать N+1 *аргумент,* первый должен иметь тип *тип_данных_состояния, а остальные — типы соответствующих* входных данных. Возвращать она должна значение типа *тип_данных_состояния*. Эта функция принимает текущее значение состояния и текущие значения входных данных, и возвращает следующее значение состояния. В сортирующих агрегатах функция перехода состояния получает только текущее значение состояния и агрегируемые аргументы, без непосредственных аргументов.

тип_данных_состояния

Тип данных значения состояния для агрегатной функции.

- *размер_данных_состояния*

Средний размер значения состояния агрегата (в байтах). Если этот параметр опущен или равен нулю, применяемая оценка по умолчанию определяется по *типу_данных_состояния.* Планировщик использует это значение для оценивания объёма памяти, требуемого для агрегатного запроса с группировкой.

Параметры. продолжение

- *функция_завершения*

Имя функции завершения, вызываемой для вычисления результата агрегатной функции после обработки всех входных строк. Для обычного агрегата эта функция должна принимать единственный аргумент типа *тип_данных_состояния. Возвращаемым типом агрегата будет тип,* который возвращает эта функция. Если *функция_завершения не указана, результатом агрегата* будет конечное значение состояния, а типом результата — *тип_данных_состояния.*

В сортирующих (и в том числе, гипотезирующих) агрегатах функция завершения получает не только конечное значение состояния, но и значения всех непосредственных аргументов.

Если команда содержит указание FINALFUNC_EXTRA, то в дополнение к конечному значению состояния и всем непосредственным аргументам функция завершения получает добавочные значения NULL, соответствующие обычным (агрегируемым) аргументам агрегата. Это в основном полезно для правильного определения типа результата при создании полиморфной агрегатной функции.

FINALFUNC_MODIFY = { READ_ONLY | SHAREABLE | READ_WRITE }

Этот параметр указывает, является ли функция завершения чистой функцией, которая не изменяет свои аргументы. Это свойство функции передаёт значение READ_ONLY; два других значения показывают, что она может менять значение переходного состояния. По умолчанию подразумевается значение READ_ONLY, за исключением сортирующих агрегатов (для них значение по умолчанию — READ_WRITE).

- *комбинирующая_функция*

Дополнительно может быть указана *комбинирующая_функция, чтобы агрегатная функция* поддерживала частичное агрегирование. Если задаётся, *комбинирующая_функция должна* комбинировать два значения *типа_данных_состояния, содержащих результат агрегирования* по некоторому подмножеству входных значений, и вычислять новое значение *типа_данных_состояния, представляющее результат агрегирования по обоим множествам* данных. Эту функцию можно считать своего рода *функцией_состояния, которая вместо* обработки отдельной входной строки и включения её данных в текущее агрегируемое состояние включает некоторое агрегированное состояние в текущее. Указанная *комбинирующая_функция должна быть объявлена как принимающая два аргумента типа_данных_состояния и возвращающая значение типа_данных_состояния. Эта функция* дополнительно может быть объявлена «строгой». В этом случае данная функция не будет вызываться, когда одно из входных состояний — NULL; в качестве корректного результата будет выдано другое состояние. Для агрегатных функций, у которых *тип_данных_состояния — internal, комбинирующая_функция* не должна быть «строгой». При этом *комбинирующая_функция должна позаботиться о том,* чтобы состояния NULL обрабатывались корректно и возвращаемое состояние располагалось в контексте памяти агрегирования.

- *функция_сериализации*

Агрегатная функция, у которой *тип_данных_состояния — internal, может участвовать в* параллельном агрегировании, только если для неё задана *функция_сериализации, которая* должна сериализовать агрегатное состояние в значение bytea для передачи другому процессу. Эта функция должна принимать один аргумент типа internal и возвращать тип bytea. Также при этом нужно задать соответствующую *функцию_десериализации.*

- *функция_десериализации*

Десериализует ранее сериализованное агрегатное состояние обратно в *тип_данных_состояния.* Эта функция должна принимать два аргумента типов bytea и internal и выдавать результат

bytea

- Двоичные строки представляют собой последовательность октетов (байт) и имеют два отличия от текстовых строк. Во-первых, в двоичных строках можно хранить байты с кодом 0 и другими «непечатаемыми» значениями (обычно это значения вне десятичного диапазона 32..126). В текстовых строках нельзя сохранять нулевые байты, а также значения и последовательности значений, не соответствующие выбранной кодировке базы данных. Во-вторых, в операциях с двоичными строками обрабатываются байты в чистом виде, тогда как текстовые строки обрабатываются в зависимости от языковых стандартов.

Агрегатная сумма пример

```
CREATE TYPE complex AS (
        x double precision,
        y double precision
);

CREATE FUNCTION complex_add(complex, complex)
RETURNS complex
AS
'select $1.x+$2.x, $1.y+$2.y'
 LANGUAGE SQL
   IMMUTABLE
   RETURNS NULL ON NULL INPUT;

CREATE AGGREGATE sum (complex)
(
sfunc = complex_add,
stype = complex,
initcond = '(0,0)'
);
```

Агрегатная сумма пример 2

```
CREATE FUNCTION activity_min(ac1 activity ,ac2 activity )
RETURNS activity LANGUAGE plpgsql
AS
$$begin
IF ac1.started IS NULL
THEN
   RETURN ac2;
ELSEIF ac2.started IS NULL
     THEN RETURN ac1;
       ELSEIF ac1.started<=ac2.started
            THEN RETURN ac1;
              else  RETURN ac2;
END IF;
end; $$
   IMMUTABLE    RETURNS NULL ON NULL INPUT;

CREATE AGGREGATE min (activity)
(
sfunc = activity_min,
stype = activity
);
```

Агрегатная сумма пример 2

```
select id_author, min(task_.task_data) from task_ group by id_author ;
```

a Output | Explain | Messages | Notifications

id_author integer	min activity
2	("разработка архитектуры",создано,"2022-02-15 09:09:00","2022-03-15 09:00:00",)
1	("анализ требований",создано,"2022-02-15 09:05:25","2022-02-15 09:10:00",)

Data Output | Explain | Messages | Notifications

	id_task [PK] integer	task_data activity	is_done boolean	id_author integer
1	1	("анализ требований",создано,"2022-02-15 09:05:25","2022-02-15 09:10:00",)	false	1
2	2	("проектирование интерфейса",создано,"2022-02-15 09:07:00","2022-03-01 09:00:00",)	false	1
3	4	("составление технического задания",создано,"2022-02-15 09:10:00","2022-02-22 09:00:00","2022-02-28 09:10:00")	false	1
4	5	("анализ требований",создано,"2022-02-15 09:12:00",„)	false	1
5	3	("разработка архитектуры",создано,"2022-02-15 09:09:00","2022-03-15 09:00:00",)	false	2

Агрегатное среднее пример 3

```
CREATE TYPE average_state AS (
  accum interval,  qty numeric);
CREATE or replace FUNCTION activity_avg_time(stat average_state,ac1 activity)
RETURNS average_state LANGUAGE plpgsql
AS  $$
begin
IF (ac1.started IS NULL)or(ac1.ended is null)
THEN    RETURN stat;
ELSE     RETURN ROW(stat.accum+(ac1.ended-ac1.started),stat.qty+1)::average_state;
END IF;
end; $$    IMMUTABLE;

CREATE OR REPLACE FUNCTION average_time_final(
  state average_state
) RETURNS  interval  AS  $$
BEGIN  RETURN CASE WHEN state.qty > 0 THEN state.accum/state.qty
  END;
END; $$ LANGUAGE plpgsql;
CREATE AGGREGATE average_time(activity) (
  sfunc     = activity_avg_time,
  stype     = average_state,
  finalfunc = average_time_final,
  initcond  = '(0,0)'
);
select average_time(task_.task_data)from task_;
```

Агрегатное среднее пример 3

```
102  select id_task,(task_).task_data.name,(task_).task_data.started,(task_).task_data.ended,
103  ((task_).task_data.ended -(task_).task_data.started) as tm from task_;
104
```

Data Output | Explain | Messages | Notifications

	id_task [PK] integer	name character varying (50)	started timestamp without time zone	ended timestamp without time zone	tm interval
1	2	проектирование интерфейса	2022-03-01 09:00:00	[null]	[null]
2	4	составление технического задания	2022-02-22 09:00:00	2022-02-28 09:10:00	6 days 00:10:00
3	5	анализ требований	[null]	[null]	[null]
4	3	разработка архитектуры	2022-03-15 09:00:00	[null]	[null]
5	6	интеграция с существующими системами	2022-07-15 09:00:00	2022-08-31 09:00:00	47 days
6	7	интеграция с внешними системами	2022-07-30 09:00:00	2022-08-31 18:00:00	32 days 09:00:00
7	1	анализ требований	2022-02-15 09:10:00	2022-02-22 09:00:00	6 days 23:50:00

```
139  select average_time(task_.task_data)from task_;
140
```

Data Output | Explain | Messages | Notifications

	average_time interval
1	22 days 26:15:00

Типы агрегатных функций

- обычная («normal»),
- сортирующая («ordered-set»)
- гипотезирующая («hypothetical-set»)
- Гипотезирующая функция

Сортирующие функции

- **Процентиль** — мера, в которой процентное значение общих значений равно этой мере или меньше ее. Например, 90 % значений данных находятся ниже 90-го **процентиля**, а 10 % значений данных находятся ниже 10-го **процентиля**.
- **percentile_disc** Вычисляет *дискретный процентиль — первое значение в отсортированном* множестве значений агрегатного аргумента, позиция которого в этом множестве равна или больше значения *fraction.* Агрегируемый аргумент должен быть сортируемого типа.
- **percentile_cont** (*fraction double precision) WITHIN GROUP (ORDER BY interval)* →interval
- Вычисляет *непрерывный процентиль — значение, соответствующее дроби,* заданной параметром *fraction, в отсортированном множестве значений* агрегатного аргумента. При этом в случае необходимости соседние входные значения будут интерполироваться.
- **mode ()** WITHIN GROUP (ORDER BY anyelement) → anyelement

Вычисляет *моду* — наиболее часто встречающееся в *агрегируемом аргументе* значение (если одинаково часто встречаются несколько значений, произвольно выбирается первое из них). Агрегируемый аргумент должен быть сортируемоготипа.

Сортирующие функции

```
56  SELECT  id_author,   count(id_task)
57      FROM task_ group by task_.id_author;
58
```

Data Output | Explain | Messages | Notifications

	id_author integer	count bigint
1	3	2
2	2	1
3	1	4

- (2+1+4)/3= 2,3(3)

```
63  SELECT  percentile_disc(0.5) within group ( order by q.id_author)
64      FROM    (SELECT id_author, count(id_task) as cnt
65      FROM task_ group by task_.id_author) as q;
66
```

Data Output | Explain | Messages | Notifications

	percentile_disc integer
1	2

Data Output | Explain

	id_author integer	count bigint
1	2	1
2	3	2
3	1	4

Оконные функции

- *Оконные функции* дают возможность выполнять вычисления с набором строк, каким-либо образом связанным с текущей строкой. Её действие можно сравнить с вычислением, производимым агрегатной функцией. Однако с оконными функциями строки не группируются в одну выходную строку,как с обычными, не оконными, агрегатными функциями. Вместо этого, эти строки остаются отдельными сущностями. Внутри же, оконная функция, как и агрегатная, может обращаться не только к текущей строке результата запроса. запроса.

Оконные функции

```
74
75  select id_proj,min((task_).task_data.started) from
76  project_task join task_ using(id_task) group by id_proj;
77
```

Data Output | Explain | Messages | Notifications

	id_proj integer	min timestamp without time zone
1	2	2022-07-15 09:00:00
2	1	2022-02-15 09:10:00

```
select id_proj,task_.id_task,(task_).task_data.started,
min((task_).task_data.started ) over (PARTITION BY id_proj)
from project_task join task_ using(id_task);
```

ata Output | Explain | Messages | Notifications

	id_proj integer	id_task integer	started timestamp without time zone	min timestamp without time zone
	1	1	2022-02-15 09:10:00	2022-02-15 09:10:00
	1	2	2022-03-01 09:00:00	2022-02-15 09:10:00
	1	4	2022-02-22 09:00:00	2022-02-15 09:10:00
	1	5	[null]	2022-02-15 09:10:00
	1	3	2022-03-15 09:00:00	2022-02-15 09:10:00
	2	6	2022-07-15 09:00:00	2022-07-15 09:00:00
	2	7	2022-07-30 09:00:00	2022-07-15 09:00:00

```
select id_proj,task_.id_task,(task_).task_data.started,
min((task_).task_data.started ) over ()
from project_task join task_ using(id_task);
```

a Output | Explain | Messages | Notifications

id_proj integer	id_task integer	started timestamp without time zone	min timestamp without time zone
1	1	2022-02-15 09:10:00	2022-02-15 09:10:00
1	2	2022-03-01 09:00:00	2022-02-15 09:10:00
1	4	2022-02-22 09:00:00	2022-02-15 09:10:00
1	5	[null]	2022-02-15 09:10:00
1	3	2022-03-15 09:00:00	2022-02-15 09:10:00
2	6	2022-07-15 09:00:00	2022-02-15 09:10:00
2	7	2022-07-30 09:00:00	2022-02-15 09:10:00

Оконные функции rank

rank Возвращает ранг текущей строки с пропусками; то же, что и row_number для первой родственной ей строки.

```
81
82  select id_proj,task_.id_task,(task_).task_data.started,
83  rank( ) over (PARTITION BY id_proj order by (task_).task_data.started)
84  from project_task join task_ using(id_task);
85
86
```

Data Output | Explain | Messages | Notifications

	id_proj integer	id_task integer	started timestamp without time zone	rank bigint
1	1	1	2022-02-15 09:10:00	1
2	1	4	2022-02-22 09:00:00	2
3	1	2	2022-03-01 09:00:00	3
4	1	3	2022-03-15 09:00:00	4
5	1	5	[null]	5
6	2	6	2022-07-15 09:00:00	1
7	2	7	2022-07-30 09:00:00	2

Гипотезирующие функции

- Все «гипотезирующие» агрегатные функции связаны с одноимёнными оконными функциями.В каждом случае их результат — значение, которое бы вернула связанная оконная функция для «гипотетической» строки, полученной из *аргументов, если бы такая строка была добавлена в сортированную* группу строк, которую образуют *сортирующие_аргументы. Для всех этих функций список* непосредственных аргументов, переданный в качестве *аргументов, по числу и типу элементов* должен соответствовать списку, передаваемому в качестве *сортирующих_аргументов.*
- rank (*аргументы) WITHIN GROUP (ORDER BY сортирующие_аргументы) → bigint*
- Вычисляет ранг гипотетической строки с пропусками, то есть номер первой родственной ей строки.

Гипотезирующие функции. rank

```
82  select id_proj,task_.id_task,(task_).task_data.started,
83  rank( ) over (PARTITION BY id_proj order by (task_).task_data.started)
84  from project_task join task_ using(id_task);
85
86
```

Data Output | Explain | Messages | Notifications

	id_proj integer	id_task integer	started timestamp without time zone	rank bigint
1	1	1	2022-02-15 09:10:00	1
2	1	4	2022-02-22 09:00:00	2
3	1	2	2022-03-01 09:00:00	3
4	1	3	2022-03-15 09:00:00	4
5	1	5	[null]	5
6	2	6	2022-07-15 09:00:00	1
7	2	7	2022-07-30 09:00:00	2

```
90  select now(), rank(1, now() ) within group (order by id_proj, (task_).task_data.started)
91  from project_task join task_ using(id_task);
92
```

Data Output | Explain | Messages | Notifications

	now timestamp with time zone	rank bigint
1	2022-02-20 17:19:57.926538+03	2

```
90  select now(), rank(2, now() ) within group (order by id_proj, (task_).task_data.started)
91  from project_task join task_ using(id_task);
92
```

Data Output | Explain | Messages | Notifications

	now timestamp with time zone	rank bigint
1	2022-02-20 17:19:08.369831+03	6

Гипотезирующие функции. rank

```
82  select id_proj,task_.id_task,(task_).task_data.started,
83  rank( ) over (PARTITION BY id_proj order by (task_).task_data.started)
84  from project_task join task_ using(id_task);
85
86
```

Data Output | Explain | Messages | Notifications

	id_proj integer	id_task integer	started timestamp without time zone	rank bigint
1	1	1	2022-02-15 09:10:00	1
2	1	4	2022-02-22 09:00:00	2
3	1	2	2022-03-01 09:00:00	3
4	1	3	2022-03-15 09:00:00	4
5	1	5	[null]	5
6	2	6	2022-07-15 09:00:00	1
7	2	7	2022-07-30 09:00:00	2

```
90  select id_proj,now(), rank( 2,now() )
91  within group
92  (order by id_proj,(task_).task_data.started)
93  from project_task join task_
94  using(id_task) group by id_proj;
95
```

Data Output | Explain | Messages | Notifications

	id_proj integer	now timestamp with time zone	rank bigint
1	1	2022-02-20 17:31:58.029265+03	6
2	2	2022-02-20 17:31:58.029265+03	1

```
90  select id_proj,now(), rank( 1,now() )
91  within group
92  (order by id_proj,(task_).task_data.started)
93  from project_task join task_
94  using(id_task) group by id_proj;
95
```

Data Output | Explain | Messages | Notifications

	id_proj integer	now timestamp with time zone	rank bigint
1	1	2022-02-20 17:31:01.318824+03	2
2	2	2022-02-20 17:31:01.318824+03	1

Агрегатное среднее пример 3

```
CREATE TYPE average_state AS (
  accum interval,  qty numeric);
CREATE or replace FUNCTION activity_avg_time(stat average_state,ac1 activity)
RETURNS average_state LANGUAGE plpgsql
AS  $$
begin
IF (ac1.started IS NULL)or(ac1.ended is null)
THEN    RETURN stat;
ELSE     RETURN ROW(stat.accum+(ac1.ended-ac1.started),stat.qty+1)::average_state;
END IF;
end; $$    IMMUTABLE;

CREATE OR REPLACE FUNCTION average_time_final(
  state average_state
) RETURNS  interval  AS  $$
BEGIN  RETURN CASE WHEN state.qty > 0 THEN state.accum/state.qty
  END;
END; $$ LANGUAGE plpgsql;
CREATE AGGREGATE average_time(activity) (
  sfunc     = activity_avg_time,
  stype     = average_state,
  finalfunc = average_time_final,
  initcond  = '(0,0)'
);
select average_time(task_.task_data)from task_;
```

Агрегатное со скользящим окном

```
CREATE OR REPLACE FUNCTION activity_avg_time_inverse(stat average_state, ac1 activity)
RETURNS average_state AS $$
BEGIN
  IF (ac1.started IS NULL)or(ac1.ended is null)
THEN    RETURN stat;
ELSE    RETURN ROW(stat.accum-(ac1.ended-ac1.started),stat.qty-1)::average_state;
END IF;
END;
$$ LANGUAGE plpgsql;

CREATE AGGREGATE average_time_move(activity) (
  -- обычный вариант
  sfunc     = activity_avg_time,
  stype     = average_state,
  finalfunc = average_time_final,
  initcond   = '(0,0)',
  -- вариант с “обратной” функцией
  msfunc     = activity_avg_time,
  minvfunc   = activity_avg_time_inverse,
  mstype     = average_state,
  mfinalfunc = average_time_final,
  minitcond  = '(0,0)'
);
```

Скользящее окно

```
102  select id_task,(task_).task_data.name,(task_).task_data.started,(task_).task_data.ended,
103  ((task_).task_data.ended -(task_).task_data.started) as tm from task_;
104
```

Data Output | Explain | Messages | Notifications

	id_task [PK] integer	name character varying (50)	started timestamp without time zone	ended timestamp without time zone	tm interval
1	2	проектирование интерфейса	2022-03-01 09:00:00	[null]	[null]
2	4	составление технического задания	2022-02-22 09:00:00	2022-02-28 09:10:00	6 days 00:10:00
3	5	анализ требований	[null]	[null]	[null]
4	3	разработка архитектуры	2022-03-15 09:00:00	[null]	[null]
5	6	интеграция с существующими системами	2022-07-15 09:00:00	2022-08-31 09:00:00	47 days
6	7	интеграция с внешними системами	2022-07-30 09:00:00	2022-08-31 18:00:00	32 days 09:00:00
7	1	анализ требований	2022-02-15 09:10:00	2022-02-22 09:00:00	6 days 23:50:00

```
170  select average_time_move(task_.task_data)
171  OVER (ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) from task_;
172
```

Data Output | Explain | Messages | Notifications

	average_time_move interval
1	[null]
2	6 days 00:10:00
3	6 days 00:10:00
4	6 days 00:10:00
5	47 days
6	39 days 16:30:00
7	28 days 18:56:40

```
168  select average_time_move(task_.task_data)from task_;
169
```

Data Output | Explain | Messages | Notifications

	average_time_move interval
1	22 days 26:15:00

Скользящее окно

```
102  select id_task,(task_).task_data.name,(task_).task_data.started,(task_).task_data.ended,
103  ((task_).task_data.ended -(task_).task_data.started) as tm from task_;
104
```

Data Output | Explain | Messages | Notifications

	id_task [PK] integer	name character varying (50)	started timestamp without time zone	ended timestamp without time zone	tm interval
1	2	проектирование интерфейса	2022-03-01 09:00:00	[null]	[null]
2	4	составление технического задания	2022-02-22 09:00:00	2022-02-28 09:10:00	6 days 00:10:00
3	5	анализ требований	[null]	[null]	[null]
4	3	разработка архитектуры	2022-03-15 09:00:00	[null]	[null]
5	6	интеграция с существующими системами	2022-07-15 09:00:00	2022-08-31 09:00:00	47 days
6	7	интеграция с внешними системами	2022-07-30 09:00:00	2022-08-31 18:00:00	32 days 09:00:00
7	1	анализ требований	2022-02-15 09:10:00	2022-02-22 09:00:00	6 days 23:50:00

```
170  select average_time_move(task_.task_data)
171  OVER (ROWS BETWEEN 2 PRECEDING AND CURRENT ROW) from task_;
172
```

Data Output | Explain | Messages | Notifications

	average_time_move interval	tm interval	tm interval	tm interval
1	[null]	[null]	[null]	[null]
2	6 days 00:10:00	6 days 00:10:00	6 days 00:10:00	6 days 00:10:00
3	6 days 00:10:00	[null]	[null]	[null]
4	6 days 00:10:00	[null]	[null]	[null]
5	47 days	47 days	47 days	47 days
6	39 days 16:30:00	32 days 09:00:00	32 days 09:00:00	32 days 09:00:00
7	28 days 18:56:40	6 days 23:50:00	6 days 23:50:00	6 days 23:50:00

ALTER AGGREGATE изменение

ALTER AGGREGATE *имя (сигнатура_агр_функции) RENAME TO новое_имя*

ALTER AGGREGATE *имя (сигнатура_агр_функции)* OWNER TO { *новый_владелец | CURRENT_ROLE | CURRENT_USER |* SESSION_USER }

ALTER AGGREGATE *имя (сигнатура_агр_функции) SET SCHEMA новая_схема*

Здесь *сигнатура_агр_функции:*

* |

[*режим_аргумента] [имя_аргумента] тип_аргумента [, ...] |*

[[*режим_аргумента] [имя_аргумента] тип_аргумента [, ...]] ORDER BY*

[*режим_аргумента] [имя_аргумента] тип_аргумента [, ...]*

Удаление агрегатной функции

- DROP AGGREGATE [IF EXISTS] *имя (сигнатура_агр_функции)* [, ...] [CASCADE |RESTRICT]

сигнатура_агр_функции:

* |

[*режим_аргумента] [имя_аргумента] тип_аргумента [, ...] |*

[[*режим_аргумента] [имя_аргумента] тип_аргумента [, ...]] ORDER BY*

[*режим_аргумента] [имя_аргумента] тип_аргумента [, ...]*

- *тип_аргумента*

Тип входных данных, с которыми работает агрегатная функция. Чтобы сослаться на агрегатную функцию без аргументов, укажите вместо списка аргументов *, а чтобы сослаться на сортирующую агрегатную функцию, добавьте ORDER BY между указаниями непосредственных и агрегируемых аргументов.

- Пример

DROP AGGREGATE min (activity)