Работа с СУБД SQL

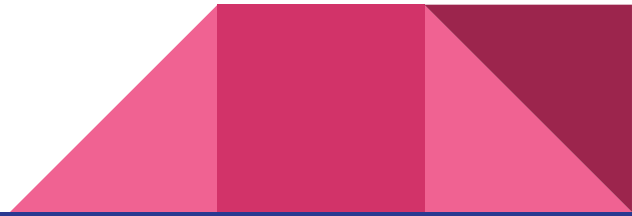
Пользовательские функции.

Проверить, идет ли запись

Меня хорошо видно && слышно?

Знакомство

- настройка микрофона и аудио
- проверка работы чата



Хранимые процедуры

Хранимые процедуры (stored procedures)

- **Хранимая процедура** – это именованный набор SQL- команд либо языковых инструкций, сохраняемая непосредственно на сервере баз данных и оптимизированная для наиболее эффективной работы.
- Хранимая процедура не возвращает каких-либо значений.
- Хранимые процедуры существуют независимо от таблиц или каких-либо других объектов баз данных.
- Команда **CREATE PROCEDURE** (создать хранимую процедуру) определена в стандарте SQL, но лишь в последних версиях PostgreSQL была принята к использованию. В этой СУБД язык программирования для написания хранимой процедуры может быть различным.
- Для изменения содержания кода хранимой процедуры используется команда **ALTER PROCEDURE**, для удаления - **DROP PROCEDURE.**

Хранимые процедуры (stored procedures)

```
-- создаём хранимую процедуру
CREATE PROCEDURE insert_info (
    description varchar(50),
    smin int,
    smax int
)
LANGUAGE SQL
AS $$
    INSERT INTO jobs SELECT 1 + MAX(job_id),
    description, smin, smax FROM jobs;
$$;

-- выполняем хранимую процедуру
CALL insert_info('Разработчик баз даннных', 200, 250);

SELECT * FROM jobs;
```

https://www.db-fiddle.com/f/ozHnQD5tTe72SoRqDc4S8S/2

Пример

Пример процедуры, выполняющей удаление из базы данных «Склад» всего, что относится к уровню классификации товара Tov_ID.

СМОТРИ В БУДУЩЕЕ. ИНВЕСТИРУЙ В ЗНАНИЯ.

Функции

Пользовательские функции

- **Пользовательская функция** – это именованный набор SQL-команд либо языковых инструкций, сохраняемая непосредственно на сервере баз данных и оптимизированная для наиболее эффективной работы.
- Функции имеют возможность их вызова непосредственно из выражений и способны возвращать результат (в том числе как множество записей).
- Функции могут принимать в качестве аргументов (параметров) базовые типы, составные типы или их сочетания.
- Функции можно писать на языках SQL, PL/pgSQL, на процедурных языках (C, Python и др.)

Пользовательские функции

```
FUNCTION clean_emp(v int) RETURNS int AS $$
    DELETE FROM employee
        WHERE job_lvl < v; -- уровень значимости
    SELECT count(*) FROM employee
$$ LANGUAGE SQL;

SELECT clean_emp(40); -- неинформативный вывод
```

Пользовательские функции

```
DROP FUNCTION clean_emp; -- исправим:

CREATE FUNCTION clean_emp(v int) RETURNS int AS $$
DECLARE
    t1 int;
    t2 int;
BEGIN
    SELECT count(*) INTO t1 FROM employee;
    DELETE FROM employee
        WHERE job_lvl < v;
    SELECT count(*) INTO t2 FROM employee;
    RETURN t1-t2;
END;
$$ LANGUAGE plpgsql;

SELECT clean_emp(40); -- теперь лучше
```

Пользовательские функции

```
CREATE FUNCTION bigJobs(integer) RETURNS SETOF character AS $$
-- Функция возвращает описание должностей с указанного уровня
значимости
    SELECT job_desc FROM jobs WHERE job_lvl > $1;
$$ LANGUAGE sql;

SELECT bigJobs(100);

CREATE FUNCTION f_add(x integer, y integer) RETURNS integer AS $$
    SELECT x + y; -- Просто сумма двух чисел
$$ LANGUAGE SQL;

CREATE FUNCTION work_with_tab (x int)
RETURNS SETOF record
AS $$
    SELECT $1 + tab.y, $1 * tab.y FROM tab;
$$ LANGUAGE SQL; -- Функция возвращает таблицу из двух столбцов
```

https://www.db-fiddle.com/f/ozHnQD5tTe72SoRqDc4S8S/0

Практика

Реализовать функцию, вычисляющую:

$$s = \sum_{i=1}^{n} \frac{1}{i^2}$$

Практика

Вычислить суммарную стоимость товара Tovar_ID по текущей цене

Практика

Найти цену товара на текущую дату

Пользовательские агрегирующие функции

```
CREATE AGGREGATE name ( input_data_type [ , ... ] ) (
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
)
```

или старый синтаксис:

```
CREATE AGGREGATE name (
    BASETYPE = base_type,
    SFUNC = sfunc,
    STYPE = state_data_type
    [ , FINALFUNC = ffunc ]
    [ , INITCOND = initial_condition ]
    [ , SORTOP = sort_operator ]
)
```

https://www.postgresql.org/docs/9.1/sql-createaggregate.html

- **Sfunc** — это функция перехода состояний. Она выполняется в агрегации столько раз, сколько есть строк для агрегации.
  - **internal-state** — внутреннее состояние, т. е. накопленное на данный момент значение. При первом вызове значение равно initcond (или null, если initcond не был определен)
  - **next-data-value** — значение из следующей строки

SFUNC

- **Sfunc** — это функция перехода состояний. Она выполняется в агрегации столько раз, сколько есть строк для агрегации.
  - **internal-state** — внутреннее состояние, т. е. накопленное на данный момент значение. При первом вызове значение равно initcond (или null, если initcond не был определен)
  - **next-data-value** — значение из следующей строки

call #	internal-state	next-data-value	result
1	0 (initcond)	3	3 (3>0)
2	3	10	10 (10>3)
3	10	12	12 (12>10)
4	12	5	12 (12>5)

```
select udaf_max(V)
from (
    select 3 as v
    union select 10
    union select 12
    union select 5) alias
```

https://www.postgresql.org/docs/9.1/sql-createaggregate.html

Пример

```
create function greaterint (int, int)
returns int language sql
as $$
    select case when $1 < $2 then $2 else $1 end
$$;

create aggregate udaf_max (int) (
    sfunc = greaterint,
    stype = integer,
    initcond = 0
);
```

СМОТРИ В БУДУЩЕЕ. ИНВЕСТИРУЙ В ЗНАНИЯ.

Триггеры

Триггеры

- **Триггер** определяет операцию (функцию), которая должна выполняться при наступлении некоторого события в базе данных.
- Чаще всего функция-триггер привязывается к некоторой таблице и, в случае применения к этой таблице какой-либо команды (INSERT, UPDATE, DELETE) триггер срабатывает и осуществляет какие-либо действия с этой же таблицей либо с другими объектами базы данных.
- В PostgreSQL триггер создаётся на основе существующей функции, т.е. сначала командой CREATE FUNCTION определяется пользовательская функция, затем на её основе командой **CREATE TRIGGER** определяется сам триггер.
- В триггерных функциях используются специальные переменные (**NEW, OLD, TG_NAME** и др.), содержащие информацию о сработавшем триггере. С помощью этих переменных триггерная функция может работать с данными.

Пример: ведение лога для списка пользователей

```
-- создаём необходимые таблицы
CREATE TABLE users (id SERIAL PRIMARY KEY, "name" TEXT);
CREATE TABLE log (
      id SERIAL PRIMARY KEY,
      info TEXT NOT NULL,
      ts timestamp WITHOUT TIME ZONE
);

-- создаём пользовательскую функцию
CREATE FUNCTION addLog()
RETURNS TRIGGER AS $$
BEGIN
      IF TG_OP='INSERT' THEN
            INSERT INTO log (info, ts)
            VALUES ('Добавлен пользователь', now());
            RETURN NEW;
      END IF;
END;
$$ LANGUAGE plpgsql;

-- создаём триггер
CREATE TRIGGER TrUser AFTER INSERT -- OR UPDATE OR DELETE
ON users FOR EACH ROW
EXECUTE PROCEDURE addLog();

-- добавляем запись в таблицу users
INSERT INTO users(name) VALUES ('Иван Иванов');
```

Виды триггеров

В PostgreSQL можно создавать триггеры двух типов:

- триггер на изменение данных - объявляется как функция без аргументов и с типом результата trigger. Пример такого триггера приведен выше.
- триггер на событие - объявляется как функция без аргументов и с типом результата event_trigger

Триггеры

Триггер – это специфический тип процедуры, которая вызывается автоматически, когда выполняются операции INSERT, UPDATE, DELETE.

Никакая процедура, или функция не вызывают триггер явно. Триггер относится к одной конкретной таблице и неявно вызывается, когда в неё вносятся изменения операторами *insert, update, delete*.

Целями, которые преследует триггер, могут быть:

- отслеживание ссылочной и семантической целостности базы данных
- Выполнение действий, обеспечивающих дополнительный побочный эффект при выполнении операций вставки, модификации и удаления.

Триггеры

- В PostgreSQL триггер создаётся на основе существующей функции, т.е. сначала командой **CREATE FUNCTION** определяется пользовательская функция, затем на её основе командой **CREATE TRIGGER** определяется сам триггер.
- В триггерных функциях используются специальные переменные (**NEW, OLD, TG_NAME** и др.), содержащие информацию о сработавшем триггере. С помощью этих переменных триггерная функция может работать с данными

В PostgreSQL можно создавать триггеры двух типов:

- **триггер на изменение данных** - объявляется как функция без аргументов и с типом результата trigger. Пример такого триггера приведен выше.
- **триггер на событие** - объявляется как функция без аргументов и с типом результата event_trigger.

Пример

Оператор INSERT, выполняющий вставку записи(ей) во view обязан предоставить значения всех полей view, которые не допускают неопределенных значений. Для приведенного примера оператор INSERT мог бы иметь вид:

```
insert into TovarWithCurPrice (Tovar_ID, TovarName, IsTovar, Amount,   MeasUnit_ID, Parent_ID,
      Price, DateStart)
values(
  0 /* Tovar_ID */,
  'Новый товар' /* TovarName */
  1, /*IsTovar*/
  23.66, /* Amount */
  3, /* MeasUnit_ID /
  7, /* Parent_ID */
  22.76, /* PriceList.Price */
  '20120901' /* PriceList.DateStart */)
```

Пример

При выполнении операции INSERT для этого view должна быть добавлена одна запись в таблицу Tovar и одна запись с его текущей ценой – в таблицу PriceList. Это может быть реализовано триггером instead of insert.

https://www.db-fiddle.com/f/vWZm2DuAS15EiawLv6B7wF/0

СПАСИБО ЗА ВНИМАНИЕ!

#аис
#учисьваис