**Рекомендации по выполнению работы**

Темпоральные (временные, хронологические) данные – это произвольные данные, которые явно или неявно связаны с определенными датами или промежутками времени. Темпоральная БД – это БД хранящая темпоральные данные.

Фактически, если данные обладают свойством темпоральности – это значит, что можно работать не только с текущей версией некоторого объекта, но и с любым из его предыдущих состояний, таким образом, для темпоральных данных поддерживается история их изменения.

Язык PL/pgSQL имеет относительно простую структуру, что объясняется в основном тем, что каждый логически обособленный фрагмент кода существует в виде функции. Хотя на первый взгляд PL/pgSQL мало похож на другие языки программирования, сходство все же существует: логические фрагменты создаются и выполняются в виде функций, все переменные обязательно объявляются перед использованием, функции получают аргументы при вызове и возвращают некоторое значение в конце своей работы.

Программы PL/pgSQL состоят из блоков. Программные блоки вводятся в командах SQL CREATE FUNCTION, которые используются для определения функций PL/pgSQL. Структура программного блока представлена ниже:

CREATE FUNCTION имя\_функции (аргументы)

RETURNS тип\_возвращаемого\_значения AS

DECLARE

объявление;

[…]

BEGIN

команда;

[…]

END;

LANGUAGE ‘plpgsql’;

Переменные, используемые в функции, объявляются в разделе DECLARE, а тело функции находится между командами BEGIN и END.

Программы PL/pgSQL, как и в большинстве языков программирования, состоят из команд и выражений. Команда выполняет некоторое действие – например, присваивает значение переменной или выполняет запрос. Каждая команда заканчивается символом «;». Выражения представляют собой условную запись последовательности операций, результат которой принадлежит одному из базовых типов данных PostgreSQL. Рассмотрим пример функции sql, возводящей число в квадрат.

CREATE FUNCTION sqr\_int(pInt integer)

RETURNS integer AS

$BODY$

DECLARE

pRes integer;

BEGIN

pRes = pInt \* pInt;

RETURN pRes;

END;

$BODY$

LANGUAGE plpgsql;

Для того, чтобы вызвать функцию достаточно написать запрос:

SELECT \* FROM sqr\_int(12);

Ключевое слово $BODY$ используется для выделения в тексте тела объявления функции (хранимой процедуры).

*Триггер* является расширением стандартных возможностей СУБД, позволяющим определять функцию, которая должна выполняться до или после некоторой операции с базой данных. Триггеры могут быть реализованы на языке С, PL/pgSQL или любом другом процедурном языке (кроме SQL), который может использоваться в PostgreSQL для определения функций.

Создание триггера выполняется при наличии существующий триггерной функции и имеет следующий синтаксис:

CREATE TRIGGER триггер {BEFORE | AFTER}

{событие [ OR событие …] }

ON таблица

FOR EACH {ROW | STATEMENT}

EXECUTE PROCEDURE функция(аргументы)

В аргументе триггер указывается произвольное имя создаваемого триггера. Имя может совпадать с именем триггера, уже существующего в базе данных – при условии, что этот триггер установлен для другой таблицы.

Ключевое слово BEFORE означает, что функция должна выполняться перед попыткой выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд к таблице. Ключевое слово AFTER означает, что функция вызывается после завершения операции, приводящей в действие триггер.

Ключевое слово, следующее за конструкцией FOR EACH определяет количество вызовов функции при наступлении указанного события: ROW означает, что функция вызывается для каждой модифицируемой записи, а STATEMENT – функция вызывается один раз для всей команды.

Иногда требуется реализовать более сложные проверки перед вставкой или обновлением данных в таблице, чем те, которые позволяет реализовать механизм ограничений (см. ключевое слово CONSTRAINT).

Ниже приводится пример создания триггера:

CREATE TRIGGER update\_users -- имя триггера

AFTER INSERT OR UPDATE OR DELETE -- типы действия с данными

ON student -- таблица БД

FOR EACH ROW -- метод обработки записей

EXECUTE PROCEDURE user\_change(); -- выполняемая функция

Триггерные функции на языке PL/pgSQL синтаксически не отличаются от обычных функций на этом языке. Рассмотрим отрывок примера триггерной функции, используемой для работы с аутентификационными данными пользователей на информационно-образовательном портале ec.dstu.edu.ru:

CREATE OR REPLACE FUNCTION user\_change()

RETURNS trigger AS

$BODY$

DECLARE

lId bigint;

sFacultyPrefix varchar;

sCondition varchar;

bAddPrefix boolean;

sOldLogin varchar;

sNewLogin varchar;

dInDate date;

BEGIN

-- формирование префикса для логина

-- tg\_op определяет тип операции, для которой был вызван триггер

-- OLD - хранит старое значение записи

IF (tg\_op = 'DELETE' or tg\_op = 'UPDATE') THEN

-- вызов хранимой процедуры и запись значения в sOldLogin

SELECT user\_prefix || OLD.zachnumber INTO sOldLogin FROM user\_prefix(OLD.in\_fakorg\_fk, OLD.in\_date);

END IF;

-- NEW - хранит новое значение записи

IF (tg\_op = 'INSERT' OR tg\_op = 'UPDATE') THEN

-- вызов хранимой процедуры и запись значения в sNewLogin

SELECT user\_prefix || NEW.zachnumber INTO sNewLogin FROM user\_prefix(NEW.in\_fakorg\_fk, NEW.in\_date);

END IF;

IF (tg\_op = 'INSERT' OR tg\_op = 'UPDATE') THEN

-- у студента сменилась зачётка

IF (tg\_op = 'UPDATE') THEN

IF (sNewLogin <> sOldLogin) THEN

-- если пользователя с таким логином еще нет

IF (sNewLogin NOT IN (SELECT login\_name FROM user\_account)) THEN

UPDATE user\_account SET login\_name = sNewLogin

WHERE login\_name = sOldLogin;

-- иначе удаляем старого пользователя

ELSE

-- удаление пользователя заключается в установке атрибута date\_dead

UPDATE user\_account SET date\_dead=now()

WHERE login\_name = sOldLogin;

END IF;

END IF;

END IF;

RETURN NEW; -- возвращаем новое значение записи

ELSEIF (tg\_op = 'DELETE') THEN

-- проверяем, может, есть ещё студент с этой зачёткой

SELECT id INTO lId FROM student WHERE zachnumber = OLD.zachnumber;

-- если запрос SELECT не вернул ниодной записи

IF NOT FOUND THEN

-- удаление пользователя заключается в установке атрибута date\_dead

UPDATE user\_account SET date\_dead = now()

WHERE login\_name = sOldLogin;

END IF;

RETURN OLD; -- возвращаем старое значение записи

END IF;

return NULL;

end

$BODY$

LANGUAGE plpgsql;

*Примечание*: приведенный пример алгоритмически существенно не отличается от действительной процедуры назначения логина, однако, его текст был сокращен и откорректирован в учебных целях.

Источники:

1. Изучение процедурного расширения языка SQL на примере СУБД PostgreSQL 9.0: метод. Указания (см. в электронном виде)
2. Нетривиальные возможности объектно-реляционных СУБД на примере PostgreSQL (см. в электронном виде)
3. Уорсли Дж., Дрейк Дж. PostgreSQL для профессионалов (см. в электронном виде)
4. <http://www.intuit.ru/studies/courses/599/455/lecture/10165>
5. <http://www.postgresql.org/docs/9.1/static/sql-createtrigger.html>

**Задание на самостоятельную работу**

Создать темпоральную БД и приложение реализующее отображение данных на текущую или на любую другую временную отметку:

1. Реализовать базовое отношение temporary\_object, содержащее атрибуты:

* id – идентификатор записи, тип: integer;
* time\_create – время создания записи (время рождения), тип timestamp;
* time\_dead – время смерти записи, тип timestamp.

Первичный ключ (Primary Key): {id, time\_create}

1. Создать таблицы, для которых планируется реализовать свойство темпоральности, объявив их наследниками temporary\_object;
2. Для всех ключевых полей (как первичных ключей, так и внешних) создать индексы командой CREATE INDEX
3. Определить триггерную функцию(-и), которая(-ые) при добавлении новой записи в отношение, поддерживающее свойство темпоральности, проверяет наличие такой записи среди существующих объектов (сравнение по id).

Если запись с таким id найдена, то для этой записи устанавливается время смерти, а добавляемая запись содержит в себе соответственные значения атрибутов найденной записи (кроме тех, которые заданы для новой записи), пустое значение времени смерти и новое значение времени рождения, равное текущему моменту. Таким образом, у новой записи время рождения равно, времени смерти ее предка.

Если запись с таким id на найдена, то происходит ее добавление в БД с установлением в качестве времени рождения текущего момента времени.

1. Создание нескольких пользователей в БД
2. Создать представление, объединяющее и представляющее данные из нескольких темпоральных таблиц.
3. Реализовать разграничение прав доступа для пользователей с использованием представления:

* один пользователь только читает;
* другой может читать и изменять данные, но только в одной из таблиц представления (для реализации изменения данных через представление использовать INSTEAD OF триггер);
* администратор (postgres) может читать и изменять данные в любой из таблиц представления.