Лекция 5: PL/pgSQL

1. SQL

ИТМО ВТ

Логические операторы и NULL

A **AND** B

A	B	Результат
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
TRUE	NULL	NULL
FALSE	FALSE	FALSE
FALSE	NULL	FALSE
NULL	NULL	NULL

A **OR** B

A	B	Результат
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	NULL	TRUE
FALSE	FALSE	FALSE
FALSE	NULL	NULL
NULL	NULL	NULL

NOT A

A	Результат
TRUE	FALSE
FALSE	TRUE
NULL	NULL

2. Добавление данных в базу данных

ИТМО ВТ

DML

Операторы манипуляции данными *(Data Manipulation Language, DML)*:

- INSERT добавляет новые данные;
- UPDATE изменяет существующие данные;
- DELETE удаляет данные.

ИТМО ВТ

INSERT

INSERT позволяет добавить данные в таблицу.

INSERT INTO **STUDENT** VALUES (123, 'Vasya', 345);

INSERT INTO **STUDENT** (StudID, GroupID, Name)
VALUES (123, 345, 'Vasya');

STUDENT

StudID	Name	GroupID
123	Vasya	345

INSERT

Для заполнения можно использовать результат запроса:

```
INSERT INTO STUDENT
        SELECT ID, Name, 345 FROM TMP_PERSON
        WHERE Exam > 70;
```

STUDENT

StudID	Name	GroupID
12	Petr	345
15	Ivan	345

ИТМО ВТ

UPDATE

UPDATE позволяет изменить данные в таблице.

STUDENT

StudID	Name	GroupID
12	Petr	345
15	Ivan	345

UPDATE **STUDENT** SET GroupID = 578;

STUDENT

StudID	Name	**GroupID**
12	Petr	**578**
15	Ivan	**578**

UPDATE с условием

При обновлении можно задавать условие.

STUDENT

StudID	Name	GroupID
12	Petr	578
15	Ivan	578

UPDATE **STUDENT** SET GroupID = 34

WHERE Name = 'Ivan';

STUDENT

StudID	Name	GroupID
12	Petr	578
15	Ivan	**34**

DELETE

DELETE позволяет удалить данные из таблицы.

STUDENT

StudID	Name	GroupID
12	Petr	578
15	Ivan	34

DELETE FROM **STUDENT** WHERE GroupID = 34;

STUDENT

StudID	Name	**GroupID**
12	Petr	578

DELETE

DELETE позволяет удалить данные из таблицы.

STUDENT

StudID	Name	**GroupID**
12	Petr	578

DELETE FROM **STUDENT**;

STUDENT

StudID	Name	**GroupID**

TRUNCATE

TRUNCATE позволяет эффективно очистить таблицу (особенно, если в ней много данных):

STUDENT

StudID	Name	**GroupID**
12	Petr	578

TRUNCATE TABLE **STUDENT**;

STUDENT

StudID	Name	**GroupID**

Можно удалить данные из нескольких таблиц:

TRUNCATE TABLE **STUDENT**, **GROUP**, **EXAM**;

ИТМО ВТ

Скрипты

Запуск скрипта можно осуществить:

- при вызове psql:

  psql … -f script.sql

- с использованием мета-команд (в psql):

  \i script.sql

  \include script.sql

3. Представления

ИТМО ВТ

Создание таблиц

Таблицы:

- *Базовые* — в действительности существующие, хранящиеся в физической памяти машины.

- ***Виртуальные*** — представления, неименованные таблицы (таблицы, которые не существуют постоянно в базе данных).

ИТМО ВТ

Представление

Представление — именованный запрос.

```
CREATE VIEW VIEW_NAME
[ ( ColumnName [, ...] ) ]
AS подзапрос
...
```

ИТМО ВТ

Представление

```
CREATE VIEW PICTStudents AS
    SELECT * FROM STUDENT
    WHERE GroupID IN (
        SELECT GroupID FROM GROUP
         WHERE GroupName LIKE 'P3%'
    );
```

ИТМО ВТ

Представление

```
CREATE VIEW PICTStudents2 AS
    (PICTId, pSurname) AS
    SELECT StudentID, Surname FROM STUDENT
    WHERE GroupID IN (
        SELECT GroupID FROM GROUP
         WHERE GroupName LIKE 'P3%'
    );


SELECT PICTid FROM PICTStudents2;
```

ИТМО ВТ

Материализованные представления

```
CREATE MATERIALIZED VIEW PICTStudents3 AS
    (PICTId, pSurname) AS
    SELECT StudentID, Surname FROM STUDENT
    WHERE GroupID IN (
        SELECT GroupID FROM GROUP
         WHERE GroupName LIKE 'P3%'
    );
```

- Результат запроса сохраняется в базе данных.
- Для обновления данных:

```
REFRESH MATERIALIZED VIEW PICTStudents3;
```

4. PL/pgSQL

ИТМО ВТ

Пользовательские функции

- функции на языке запросов (функции, написанные на языке SQL);
- функции на процедурных языках (функции, написанные, на PL/pgSQL);
- функции на языке Perl, Python, C, ...

ИТМО ВТ

PL/pgSQL

- PL/pgSQL это процедурный язык для СУБД PostgreSQL.

- Функции PL/pgSQL могут использоваться везде, где допустимы встроенные функции.

ИТМО ВТ

Создание функции

Добавление пользовательских функций:

CREATE FUNCTION newFunc(int)

RETURNS int

AS 'body logic'

LANGUAGE chosenLanguage;

При добавлении функции можно использовать **CREATE OR REPLACE** — чтобы заменить ранее загруженную функцию.

ИТМО ВТ

SQL-функция

Функция для изменения группы студента:

```
CREATE FUNCTION updateStudentGroup(int, int)
    RETURNS void AS
        'UPDATE STUDENT
            SET GroupID=$2 WHERE StudID=$1;'
    LANGUAGE SQL;
```

- Аргументы могут использоваться как значения, но не идентификаторы.

ИТМО ВТ

SQL-функция, параметры

- Начиная с PostgreSQL 9.2:

```
CREATE FUNCTION updateStudentGroup(
    StudIDParam int, GroupIDParam int)
    RETURNS void AS
        'UPDATE STUDENT
            SET GroupID=GroupIDParam
            WHERE StudID=StudIDParam;'
    LANGUAGE SQL;
```

Есть ли проблема в приведенном коде?

```
CREATE FUNCTION updateStudentGroup(
 GroupIDParam int )
 RETURNS void AS
        'UPDATE STUDENT
                SET GroupID=GroupIDParam
                WHERE StudName='Petr';'
LANGUAGE SQL;
```

Экранирование строк

```
CREATE FUNCTION updateStudentGroup(
 GroupIDParam int )
 RETURNS void AS
        'UPDATE STUDENT
                SET GroupID=GroupIDParam
                WHERE StudName=''Petr'';'
LANGUAGE SQL;
```

Синтаксис для записи строк через $$

Строка:

string of characters with ' symbol

может быть представлена:

- 'string of characters with '' symbol'
- $someId$string of characters with ' symbol$someId$

Синтаксис для записи строк через $$

```
CREATE FUNCTION updateStudentGroup(
 GroupIDParam int )
 RETURNS void AS $$
        UPDATE STUDENT
                SET GroupID=GroupIDParam
                WHERE StudName='Petr';
$$ LANGUAGE SQL;
```

ИТМО ВТ

Вызов функции

Для вызова функции можно использовать SELECT:

```
SELECT updateStudentGroup(950, 345);
```

ИТМО ВТ

Процедура

Добавление процедуры:

```
CREATE PROCEDURE newProc(integer)
AS $$ body logic $$
LANGUAGE chosenLanguage;
```

- В процедуре отсутствует RETURNS часть.
- Для вызова процедуры можно использовать CALL:

```
CALL newProc(42);
```

ИТМО ВТ

PL/pgSQL

PL/pgSQL это блочно-структурированный язык. Текст определения функции - блок.

Структура блока:

```
[ <<метка>> ]
[ DECLARE
    объявления ]
BEGIN
    операторы
END [ метка ];
```

ИТМО ВТ

Пример

```
CREATE FUNCTION somefunc() RETURNS integer AS $$ –- SQL command
<< outerblock »                                  –- pl/pgSQL
DECLARE
    quantity integer := 30;
BEGIN
    -- Создаем вложенный блок
    DECLARE
        quantity integer := 80;
    BEGIN
        RAISE NOTICE 'Inner quantity = %', quantity;  -- Выводится 80
        RAISE NOTICE 'Outer quantity = %', outerblock.quantity;  -- Выводится 30
    END;
    RAISE NOTICE 'Сейчас quantity = %', quantity;  -- Выводится 30
    RETURN quantity;
END;
$$ LANGUAGE plpgsql;
```

ИТМО ВТ

Анонимный блок

```
DO $$
<<studentBlock>>
DECLARE
  studCount integer := 0;
BEGIN
   SELECT COUNT(*)
   INTO studCount
   FROM STUDENT;
   RAISE NOTICE 'Students: %', studCount;
end studentBlock $$;
```

5. Ограничения целостности: триггеры

ИТМО ВТ

Ограничения целостности

- Типы данных — один из способов ограничивать данные, но его не всегда **достаточно**.

- SQL позволяет определять ограничения для колонок и таблиц:
  - CHECK, NOT NULL, UNIQUE;
  - триггеры;

ИТМО ВТ

Триггеры

Триггер — сочетание хранимой в базе данных процедуры и события, которое заставляет ее выполняться.

Могут быть объявлены как для таблицы из пользовательской БД, так и для представления (PostgreSQL).

ИТМО ВТ

Запуск триггеров

События:

- ввод новой строки;
- изменение значений одного или нескольких столбцов таблицы;
- удаление строк таблицы;

ИТМО ВТ

Запуск триггеров

При возникновении события:

- производится проверка условий срабатывания триггера;
- запускаются созданные триггеры;

ИТМО ВТ

Триггеры

Триггеры дают возможность:

- Реализовывать **сложные ограничения целостности** данных, которые невозможно реализовать через ограничения, устанавливаемые при создании таблицы *(CHECK …)*.

- Контролировать информацию, хранимую в таблице, посредством **регистрации вносимых изменений** и пользователей, производящих эти изменения.

ИТМО ВТ

Создание триггера (PostgreSQL)

Триггеры создаются на основе триггерной функции:

- функция без аргументов;
- возвращает тип *trigger, event_trigger*.

CREATE OR REPLACE FUNCTION **fname()** …

RETURNS trigger ...;

CREATE TRIGGER name …

EXECUTE PROCEDURE **fname()**;

ИТМО ВТ

Синтаксис

CREATE TRIGGER *name*

{ BEFORE | AFTER | INSTEAD OF } { event [OR ...] }

ON table_name

...

[FOR [EACH] { ROW | STATEMENT }]

[WHEN (condition)]

EXECUTE PROCEDURE **function_name** ( )

Где event:

INSERT, UPDATE [OF column_name [, ...]], DELETE, TRUNCATE

ИТМО ВТ

Виды триггеров

- **Строковый триггер** — запускается один раз для каждой строки, обрабатываемой активизирующим предложением.
- **Табличный триггер** — выполняется только один раз, независимо от того, сколько строк содержит запускающее его предложение.

Выбор того или иного типа триггера зависит от требований, определенных для базы данных.

ИТМО ВТ

Строковые/Табличные триггеры

- **ROW** — процедура будет вызываться для каждой модифицируемой записи
- **STATEMENT** — процедура будет вызываться один раз для всех обрабатываемых записей в рамках команды

CREATE TRIGGER *trigger1* BEFORE UPDATE ON table FOR **EACH ROW** EXECUTE ...;

Колоночный триггер:

CREATE TRIGGER *trigger2* BEFORE **UPDATE OF** *attr1* **ON** table **FOR EACH ROW** EXECUTE ...;

ИТМО ВТ

Типы триггеров

- Триггеры DML (обычные и условные).
- Триггеры замещения (instead of).
- Событийные триггеры (event triggers).

ИТМО ВТ

Триггеры DML

Триггеры DML активизируются предложениями ввода, обновления и удаления информации (INSERT , UPDATE , DELETE) до или после выполнения предложения, на уровне строки или таблицы.

CREATE TRIGGER *trname* BEFORE **DELETE** ON *table*

ИТМО ВТ

Триггеры DML

CREATE TRIGGER *name* { **BEFORE** | **AFTER** | ...}

- **BEFORE** — процедура вызывается перед выполнением операции (включая проверки ограничений целостности, реализуемые при выполнении команд INSERT и DELETE).
- **AFTER** — процедура вызывается после завершения операции, приводящей в действие триггер.

ИТМО ВТ

Триггеры замещения

- Триггеры замещения (**instead of**) можно создавать только для представлений.
- В отличие от триггеров DML триггеры замещения выполняются вместо предложений DML, вызывающих их срабатывание.
- Триггеры замещения должны быть строковыми триггерами.

ИТМО ВТ

Активация триггеров

Активация триггеров DML — при выполнении предложений INSERT, UPDATE, DELETE.

Для событий (связанных с таблицами) может быть создано 4 вида триггеров: табличный BEFORE, табличный AFTER, строковый BEFORE, строковый AFTER.

ИТМО ВТ

Порядок активации

Порядок активации триггеров:

1. Табличный триггер BEFORE

2. Строковый триггер BEFORE

3. Строковый триггер AFTER

4. Табличный триггер AFTER

Внутри групп порядок определяется **по алфавиту** (для PostgreSQL)

ИТМО ВТ

Переменные триггерной процедуры

Когда PL/pgSQL функция вызывается триггером, создается ряд переменных:

- NEW — (RECORD) содержит новую строку базы данных для команд INSERT/UPDATE в строковых триггерах.
- OLD — (RECORD) содержит старую строку базы данных для команд UPDATE/DELETE в строковых триггерах.
- TG_NAME — имя сработавшего триггера.
- TG_WHEN — BEFORE, AFTER или INSTEAD OF, в зависимости от определения триггера.

ИТМО ВТ

Пример

```
CREATE TABLE EMPLOYEE(
   ID              INT             PRIMARY KEY,
   NAME            TEXT            NOT NULL,
   ADDR            CHAR(50),
   SALARY          REAL
);

CREATE TABLE AUDIT(
    EMP_ID              INT         NOT NULL,
    ENTRY_DATE          TEXT        NOT NULL
);
```

ИТМО ВТ

Пример

```
CREATE TRIGGER my_trigger
AFTER INSERT ON EMPLOYEE
FOR EACH ROW EXECUTE PROCEDURE auditfunc();

CREATE OR REPLACE FUNCTION auditfunc()
RETURNS TRIGGER AS $$
    BEGIN
        INSERT INTO AUDIT(EMP_ID, ENTRY_DATE)
                VALUES (NEW.ID, current_timestamp);
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;
```

ИТМО ВТ

Событийные триггеры

- Срабатывают на предложения DDL (CREATE/ALTER/DROP).
- Процедура, которая вызывается как событийный триггер, должна объявляться без аргументов. Типом возвращаемого значения должен быть event_trigger.

ИТМО ВТ

Событийные триггеры

- Обрабатываемые события (event trigger):
  - ddl_command_start — перед началом работы DDL-команды;
  - ddl_command_end — после окончания работы DDL-команды;
  - table_rewrite — перед ALTER TABLE;
  - sql_drop — перед ddl_command_end при удалении объектов;

Событийные триггеры

```
CREATE OR REPLACE FUNCTION eventtest() RETURNS
event_trigger AS $$

BEGIN

    RAISE NOTICE 'eventtest: %', tg_event;

END;

$$ LANGUAGE plpgsql;



CREATE EVENT TRIGGER eventtest ON ddl_command_start
EXECUTE PROCEDURE eventtest();
```

Событийные триггеры

tg_event — событие, вызывающее триггер;

```
$ create table SOMETABLE (myattr int4);

NOTICE:  eventtest: ddl_command_start CREATE

CREATE TABLE
```

ИТМО ВТ

Триггеры: общие правила

- Триггерная функция должна вернуть либо NULL, либо запись/строку, соответствующую структуре таблицы, для которой сработал триггер.
- Если строковый триггер с BEFORE возвращает NULL, то все дальнейшие действия с этой строкой прекращаются (не срабатывают последующие триггеры, команда INSERT/UPDATE/DELETE для этой строки не выполняется).

Удаление триггеров

DROP TRIGGER *name*;

6. Транзакции

ИТМО ВТ

Транзакция

- Транзакции объединяют последовательность действий в одну операцию.
- Промежуточные состояния внутри последовательности операций **не видны** другим транзакциям.
- Если что-то помешает успешно завершить транзакцию, ни один из результатов этих действий не сохранится в базе данных.

ИТМО ВТ

ACID

- Atomicity (Атомарность)
- Consistency (Согласованность)
- Isolation (Изолированность)
- Durability (Устойчивость, долговечность)

ИТМО ВТ

Atomicity (Атомарность)

- Гарантирует, что результаты работы транзакции не будет зафиксированы в системе частично.
- Будут либо выполнены все операции в транзакции, либо не выполнено ни одной.

ИТМО ВТ

Consistency (Согласованность)

- После выполнения транзакции база данных должна быть в согласованном (**целостном**) состоянии.
- Во время выполнения транзакции (после выполнения отдельных операций в рамках транзакции) согласованность не требуется.

ИТМО ВТ

Isolation (Изолированность)

- Во время выполнения транзакции другие транзакции (выполняющиеся параллельно) не должны оказывать влияние на результат транзакции.

ИТМО ВТ

Долговечность (Durability)

- При успешном завершении транзакции результаты ее работы должны остаться в системе независимо от возможных сбоев оборудования, системы и тд.

ИТМО ВТ

Пример

```
UPDATE ACCOUNT  SET balance = balance - 50.00
    WHERE name = 'Alex';
UPDATE ACCOUNT  SET balance = balance + 50.00
    WHERE name = 'Ivan';
```

ИТМО ВТ

Пример

```
BEGIN;
  UPDATE ACCOUNT  SET balance = balance - 50.00
                WHERE name = 'Alex';
  UPDATE ACCOUNT  SET balance = balance + 50.00
                WHERE name = 'Ivan';
COMMIT;
```

ИТМО ВТ

Точки сохранения

- Позволяют выборочно зафиксировать часть транзакции.
- Определить точку сохранения можно с помощью **SAVEPOINT**
- Вернуться к ней можно с помощью команды **ROLLBACK TO**.
- Все изменения в базе данных, произошедшие после точки сохранения и до момента отката, **отменяются**; изменения, произведённые ранее, **сохраняются**.

ИТМО ВТ

Пример

```
BEGIN;
UPDATE ACCOUNT  SET balance = balance - 50.00
    WHERE name = 'Alex';
SAVEPOINT savepoint1;
UPDATE ACCOUNT  SET balance = balance + 50.00
    WHERE name = 'Ivan';
-- Обрабатываем некоторое условие (CASE)
   ROLLBACK TO savepoint1;
UPDATE ACCOUNT  SET balance = balance + 50.00
    WHERE name = 'Ivan2';
COMMIT;
```

ИТМО ВТ

Точки сохранения

- При удалении или откате к точке сохранения все точки сохранения, определённые после этой точки сохранения, автоматически уничтожаются.

ИТМО ВТ

Литература

При подготовке презентации использовались материалы из:

- Введение в реляционные базы данных / В. В. Кириллов, Г. Ю. Громов, Издательство: BHV, 2009 г.
- Документация PostgreSQL.

  https://www.postgresql.org/about/licence/