ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Учебно-методическое пособие

по дисциплине

«БАЗЫ ДАННЫХ»

Автор
Кузин А.П.

Ростов-на-Дону, 2018

Аннотация

Учебно-методическое пособие предназначено для студентов очной формы обучения направления 09.03.04 «Программная инженерия».

Авторы

Ст.преподаватель
Кузин А.П.

Оглавление

1. ЛАБОРАТОРНАЯ РАБОТА №1: ОСНОВЫ ПРОЕКТИРОВАНИЯ СТРУКТУРЫ БАЗ ДАННЫХ

1.1. Цель работы

Овладеть основными теоретическими знаниями и практическими приемами проектирования схемы реляционных отношений базы данных.

1.2. Задание к лабораторной работе

Поставленная цель предполагает решение следующих задач:

1. Выбор предметной области, для которой будет проектироваться БД;
2. Определение множества конечных пользователей, выбранной предметной области, которыми предполагается использование разрабатываемой БД;
3. Составление от лица выбранных пользователей (в терминах предметной области!) описания требований к разрабатываемой базе данных, в том числе включающих описания объектов предметной области и принципов их взаимодействия;
4. Выделение общих особенностей на базе описаний конкретных пользователей и отбрасывание лишних деталей (абстрагирование);
5. На базе абстрактного описания формирование логической схемы БД в следующих видах: модель «Сущность-Связь» (ER-модель) и инфологическая (или датологическая) модель данных;
6. Подготовка отчета с представлением результатов выполнения поставленных задач.

Комментарии:

При выборе предметной области рекомендуется выбирать предметную область, которая максимально знакома разработчику БД. Результаты выполнения данной работы настоятельно рекомендуется использовать при выполнении следующих лабораторных работ, а также при реализации курсовой работы (проекта). Таким образом, целесообразно выбрать предметную область один раз, чтобы планомерно в течение всего семестра использовать полученные на лабораторных работах знания и результаты в

оформлении отчета по курсовой работе.

Проектирование инфологической диаграммы и ER-модели рекомендуется осуществлять с использованием специализированного программного инструментария (например, MS Visio).

Описание предметной области с точки зрения конечных пользователей системы не должно содержать специализированных терминов и жаргонизмов (кортежи, таблицы, сущности, функционал программы и т.п.), т.е. конечные пользователи формулируют для Вас, как для разработчика БД, требования, но исключительно в терминах своей предметной области.

На ER-диаграмме не забудьте указать не только связи между сущностями, но также их вид. Число сущностей может быть произвольным и зависит от уровня детализации выбранной предметной области, предъявляемых конечными пользователями требований, а также (безусловно) от ваших знаний в данной предметной области.

Физическая реализация разработанной схемы отношений не требуется.

Критерии оценивания (требования):

1. Оценивается только работа студента, оформленная в виде печатного отчета;
2. Работа может быть оценена *неудовлетворительно* (0 баллов) в случае, если студент не показывает знаний по сути сделанной работы, т.е. затрудняется ответить на вопросы, ответы на которые в явном или неявном виде присутствуют в его отчете;
3. Работа может быть оценена *удовлетворительно* (1-2 балла) в случае, если решены все поставленные перед студентом задачи, однако студент затрудняется ответить на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, и/или имеются ошибки в составленной логической схеме и другие недостатки в оформлении работы;
4. Работа может быть оценена *хорошо* (3-4 балла) в случае, если решены все поставленные перед студентом задачи и студент правильно отвечает на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, однако имеются ошибки в составленной логической схеме и/или другие недостатки в оформлении работы;
5. Работа может быть оценена *отлично* (5 баллов) в случае,

если решены все поставленные перед студентом задачи, студент правильно отвечает на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, и отсутствуют ошибки в составленной логической схеме.

Контрольные вопросы:

1. Назовите 3 уровня абстракции и объясните, что означает каждый из них.
2. Что подразумевает понятие абстрагирование в СУБД?
3. Назовите и определите виды связей, возникающих между объектами.
4. Что такое ER-диаграмма: для чего применяется, из чего состоит?
5. Что такое реляционные базы данных, в чем их особенности?
6. Дайте определения таким базовым понятиям теории реляционных БД, как атрибут, отношение, домен, схема БД.

Пример (краткий) выполнения:

В качестве предметной области выполнения лабораторной работы выбирается – деканат. С точки зрения проектирования базы данных в данной предметной области имеются следующие конечные пользователи: декан факультета, заместители декана факультета, ведущий документовед. Опишем предметную область с точки зрения конечных пользователей.

Ведущий документовед:

В своей работе мне требуется постоянно иметь возможности просмотра и управления информацией о студенте как личной (фамилия, имя, отчество, паспортные данные, адреса, контакты и т.д.), так и учебной (учебный статус: учится, представлен к отчислению, отчислен, в академическом отпуске, номер семестра обучения, год поступления, направление подготовки, текущая успеваемость и т.д.). Работа с контингентом студентов также предполагает наличие возможности объединения их в группы для добавления в приказы (на перевод, отчисление, стипендию и т.д.) с последующим выводом на печать. Важным аспектом работы документоведа является работа с текущей успеваемостью студентов – требуется обеспечить возможность внесения и модификации текущей успеваемости и информации о пропусках занятий, а так-

же итоговой успеваемости студентов по различным изучаемым дисциплинам.

Заместитель декана:

В своей работе мне требуется оперативно получать контактную информацию о студентах (телефоны, адрес, icq, электронная почта), о учебных группах (составе, старостах и т.д.), о преподаваемых дисциплинах в каждой группе, а также о успеваемости отдельных студентов в случае поступления запроса направления на пересдачу. Требуется обеспечить возможность регистрации продления учебной сессии с регистрацией причины продления. Необходимо иметь возможность просмотра/поиска информации о преподавателях, закрепленных за различными группами, а также возможность регистрации информации о кураторах групп.

Важным аспектом является возможность регистрации различной информации о социальной и спортивной активности студента, его спортивных разрядах, участия в различных организациях, мероприятиях и т.д.

Декан:

Мне требуется доступ ко всем уже заявленным моими сотрудниками возможностями, а также важным аспектом является возможность мониторинга общих тенденций изменения ситуации по успеваемости. Требуется поддерживать возможность определения общего рейтинга студентов с учетом его учебных, спортивных и социальных достижений.

На базе представленных описаний можно выделить следующие базовые объекты предметной области: Студент, Человек, Преподаватель, Учебный план, Группа, Специализация (Профиль), Специальность (Направление подготовки), Учебный план группы, Студенческий план, Система оценивания, Оценка.

Ниже представлены примеры (фрагменты) ER–диаграммы и даталогической модели.

ВНИМАНИЕ! На ER-диаграмме представлена лишь некоторая абстракция конкретной предметной области с прикладной точки зрения, т.е. обусловленная решаемой задачей. Кроме того, в целях экономии места на конкретных примерах ниже представлены лишь некоторые атрибуты сущностей предметной области. В ре-

альной лабораторной работе рекомендуется представить как можно больше атрибутов характеризующие объекты предметной области.

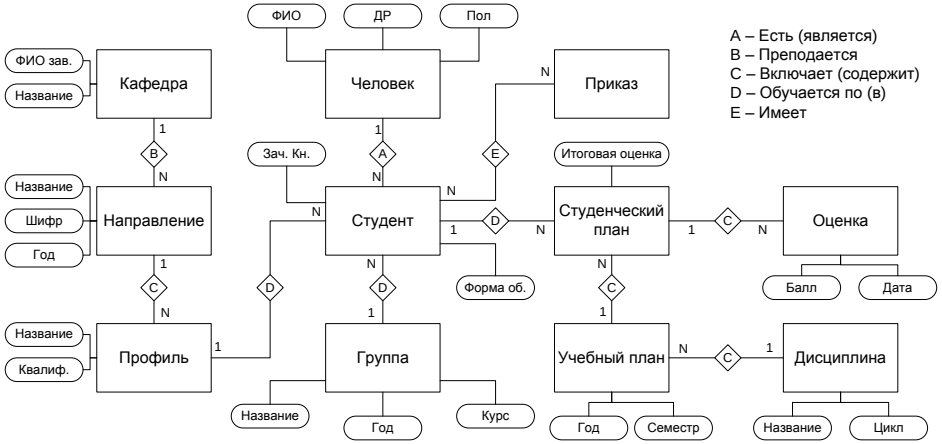

Рис. 1 – ER-диаграмма

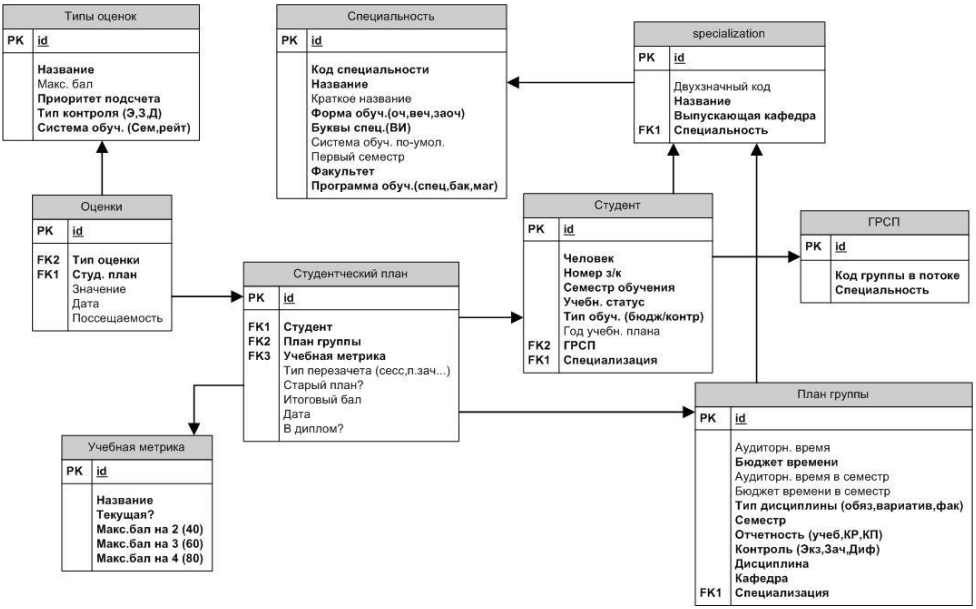

Рис. 2 – Даталогическая схема

Выбор задания по варианту

№ вар.	предметная область	№ вар.	предметная область
1	Аптека	12	Приемная комиссия
2	Склад продуктов	13	Автозаправка
3	Театр	14	Аэропорт
4	Оружейный магазин	15	Магазин продуктов
5	Парфюмерия	16	«McDonalds»
6	Магазин одежды	17	Фитнес-центр
7	Столовая	18	Книжный магазин
8	Фотоателье	19	Расписание занятий
9	Призывной пункт	20	Ресторан
10	Автовокзал	21	«Спорт-мастер»
11	Ж/д вокзал	22	Музыкальный магазин

2. ЛАБОРАТОРНАЯ РАБОТА №2: ОСНОВЫ СОЗДАНИЯ БД В POSTGRESQL

2.1. Цель работы

Овладеть основными практическими навыками типового взаимодействия с использованием интегрированных утилит СУБД с БД на примере PostgreSql.

2.2. Задание к лабораторной работе

Поставленная цель предполагает решение следующих **задач**:

1. Определить домены атрибутов, для схемы, созданной в лабораторной работе №1;
2. Создать пустую базу данных в среде PostgreSql требуемой структуры (create database);
3. Создать таблицы в базе данных, при помощи языка запросов SQL (create table);
4. Заполнить созданные таблицы данными при помощи SQL запросов (insert).
5. С помощью оператора языка SQL select вывести данные из созданных таблиц.
6. Подготовка отчета с представлением результатов выполнения поставленных задач.

Контрольные вопросы:

7. Опишите структуру оператора SELECT.
8. Как при помощи операторов SQL можно создать внешние ключи.
9. Опишите структуру оператора INSERT.

3. ЛАБОРАТОРНАЯ РАБОТА №3: SQL КОМАНДЫ SELECT И CREATE VIEW

3.1. Цель работы

Овладеть основными практическими навыками выборки данных при помощи SQL запросов Select, создания представлений при помощи SQL запроса CREATE VIEW.

3.2. Задание к лабораторной работе

Поставленная цель предполагает решение следующих **задач**:

7. К базе данных, созданной в лабораторной работе №2, описать в текстовом виде запросы требуемые для пользователей базы данных;
8. Изучить структуру запроса SELECT, агрегатные функции, структуру GROUP by и HAVING;
9. Составить на языке SQL ранее описанные запросы в лабораторной работе №1;
10. Проанализировать Вашу базу данных и составить основные возможные представления
11. Создать эти представления при помощи SQL запроса CREATE VIEW;
12. Модифицировать данные в представлении и выяснить как это отражается на исходных таблицах и наоборот;
13. Переписать ранее созданные запросы с учетом имеющихся представлений.
14. Составить отчет о выполнении работы.

Контрольные вопросы:

10. Что такое агрегатные функции, как их можно использовать.
11. Отличия GROUP by и HAVING.
12. Чем отличается представление от хранимой таблицы.

4. ЛАБОРАТОРНАЯ РАБОТА №4: ПОЛЬЗОВАТЕЛЬСКИЕ ФУНКЦИИ В POSTGRESQL

4.1. Цель работы

Изучить правила создания функций. Приобрести практические навыки создания функций в среде PostgreSQL.

4.2. Задание к лабораторной работе

Поставленная цель предполагает решение следующих **задач**:

15. Ознакомиться с теоретическими сведениями о возможностях создания пользовательских функций в PostgreSQL.
16. Проанализировать выбранную предметную область и определить требуемые функции для работы с БД;
17. Создать функции для работы с базой данных. Проверить работоспособность функций путем выполнения этих функций с параметрами, обеспечивающими как успешное выполнение функции, так и невыполнение функции.

Контрольные вопросы:

13. Как описать и использовать пользовательские функции.
14. Описание переменных
15. Типы переменных.

Критерии оценивания (требования):

1. Оценивается только работа студента, оформленная в виде печатного отчета с условием наличия базы данных в электронном виде;
2. Работа может быть оценена *неудовлетворительно* (0 баллов) в случае, если студент не показывает знаний по сути сделанной работы, т.е. затрудняется ответить на вопросы, ответы на которые в явном или неявном виде присутствуют в его отчете;
3. Работа может быть оценена *удовлетворительно* (1-2 балла) в случае, если решены все поставленные перед студентом задачи, однако студент затрудняется ответить на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, и/или имеются ошибки в составленной схеме (например, не все связи определены), составленные запросы

выполняются правильно не для любого набора входных данных, а отчеты отображают неверные данные;

4. Работа может быть оценена *хорошо* (3-4 балла) в случае, если решены все поставленные перед студентом задачи и студент правильно отвечает на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, однако имеются ошибки в составленной схеме (например, не все связи определены), составленные запросы выполняются правильно не для любого набора входных данных, а отчеты отображают неверные данные;
5. Работа может быть оценена *отлично* (5 баллов) в случае, если решены все поставленные перед студентом задачи, студент правильно отвечает на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, и отсутствуют ошибки в физической реализации.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Платформа PostgreSQL поддерживает инструкцию CREATE FUNCTION, но не поддерживает инструкции CREATE PROCEDURE, ALTER FUNCTION и ALTER PROCEDURE. Поскольку PostgreSQL не имеет собственного внутреннего процедурного языка, пользовательские функции обычно представляют внешние функции, хотя они могут быть и простыми SQL запросами.

Создание функций SQL:

CREATE [OR REPLACE] **FUNCTION** *имя_функции* **([[** *метод_аргумента* **][***имя_аргумента* **]** *тип_ аргумента* **[,...]** **])**
RETURNS *тип_возвращаемого_значения*
AS **'**
оператор *SQL;*
[*оператор* *SQL;*
...]
'
LANGUAGE **sql**
[WITH (*атрибут* **[...])];**

В теле функции обращение к параметрам осуществляется по имени или по номеру: $1 – первый параметр, $2- второй параметр и т.д.

В теле функции sql могут стоять только операторы языка

SQL, любые (Select, insert, delete, create,...) за исключением операторов управления транзакциями (commit, rollback...). Возвращаемым значением является результат выполнения оператора SELECT, его тип должен совпадать с типом, указанным после RETURNS. Если в теле функции несколько операторов SELECT, функция вернет результат выполнения последнего такого оператора. Если функция sql не содержит операторов SELECT, тип результата для нее следует указать void (фактически это процедура)

Пример:

Имеется таблица (purchase) всех проданных товаров:

id_record [PK] character(6)	check_code character(6)	product_code character(6)	data_t date	price double precision	amount integer	summa double precision
P0909	01005	52020	2015-10-09	7020	1	7020
P1715	01001	54213	2015-09-14	61790	1	61790
V1515	01002	52144	2015-09-15	6269	1	6269
C0810	01003	52020	2015-10-08	7020	2	14040
C0811	01004	52144	2015-10-08	6269	1	6269
C0812	01004	54213	2015-10-06	61790	1	61790

Функция подсчета количества общего количества (amount) проданного товара определенного кода (product_code):

```
CREATE FUNCTION kol_sales (character) RETURNS SETOF bigint AS '
select sum(amount) from purchase where product_code=$1;
' LANGUAGE sql;
```

Запрос:

```
SELECT DISTINCT product_code,price,
kol_sales (product_code) from purchase
where product_code=product_code
```

Результат запроса:

product_code character(6)	price double precision	kol_sales bigint
52020	7020	3
54213	61790	2
52144	6269	2

Запрос:

```
SELECT DISTINCT product_code as код_товара,price as цена,
kol_sales (product_code) as кол, kol_sales (product_code)*price as сумма from purchase
where product_code=product_code
```

Результат запроса:

код_товара character(6)	цена double precision	кол bigint	сумма double precision
54213	61790	2	123580
52144	6269	2	12538
52020	7020	3	21060

Только в PostgreSql возможен нестандартный вариант использования функций:

SELECT kol_sales ('52020') AS kol ;

Результат запроса:

kol bigint
3

5. ЛАБОРАТОРНАЯ РАБОТА №5: ТРИГГЕРЫ

5.1. Цель работы

Изучить правила создания триггеров. Приобрести практические навыки создания триггеров в среде PostgreSQL.

5.2. Задание к лабораторной работе

Поставленная цель предполагает решение следующих **задач**:

1. Ознакомиться с теоретическими сведениями о возможностях создания триггеров в PostgreSQL.
2. Реализовать триггер вида BEFORE, для проверки возможности изменения или добавления данных (например, при продаже товара проверить если необходимое количество товара на складе).
3. Реализовать триггер AFTER, для коррекции данных с учетом изменения или добавленных новых данных (например, при продаже товара изменить количество товара на складе).
4. Реализовать триггер каскадного удаления данных.
5. Создать триггер, служащий для отслеживания изменений записей таблицы одной таблицы. Необходимо хранить информацию о имени таблицы, имени атрибута, старом значении, новом значении, дате изменения.

Контрольные вопросы:

1. Назначение триггеров.

2. Различия видов триггеров.
3. Проблемы возникающие при использовании тригеров.

Критерии оценивания (требования):

1. Оценивается только работа студента, оформленная в виде печатного отчета с условием наличия базы данных в электронном виде;
2. Работа может быть оценена *неудовлетворительно* (0 баллов) в случае, если студент не показывает знаний по сути сделанной работы, т.е. затрудняется ответить на вопросы, ответы на которые в явном или неявном виде присутствуют в его отчете;
3. Работа может быть оценена *удовлетворительно* (1-2 балла) в случае, если решены все поставленные перед студентом задачи, однако студент затрудняется ответить на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, и/или имеются ошибки в составленной схеме (например, не все связи определены), составленные запросы выполняются правильно не для любого набора входных данных, а отчеты отображают неверные данные;
4. Работа может быть оценена *хорошо* (3-4 балла) в случае, если решены все поставленные перед студентом задачи и студент правильно отвечает на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, однако имеются ошибки в составленной схеме (например, не все связи определены), составленные запросы выполняются правильно не для любого набора входных данных, а отчеты отображают неверные данные;
5. Работа может быть оценена *отлично* (5 баллов) в случае, если решены все поставленные перед студентом задачи, студент правильно отвечает на дополнительные вопросы по материалам лекций, относящимся к выполняемой лабораторной работе, и отсутствуют ошибки в физической реализации.

СПРАВОЧНАЯ ИНФОРМАЦИЯ

Триггер определяет операцию, которая должна выполняться при наступлении некоторого события в базе данных. Триггеры срабатывают при выполнении с таблицей команды SQL INSERT, UPDATE или DELETE.

В PostgreSQL триггеры создаются на основе существующих функции, т.е. сначала командой CREATE FUNCTION определяется триггерная функция, затем на ее основе командой CREATE TRIGGER определяется собственно триггер.

Синтаксис определения триггера

CREATE TRIGGER ***триггер***
BEFORE | AFTER } { ***событие*** [OR ***событие***] } ON ***таблица***
FOR EACH { ROW | STATEMENT }
EXECUTE PROCEDURE ***функция*** (***аргументы***)

Ниже приводятся краткие описания компонентов этого определения.

CREATE TRIGGER *триггер*. 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. Кроме того, по аналогии с большинством других несистемных объектов баз данных, имя триггера (в сочетании с таблицей, для которой он устанавливается) должно быть уникальным лишь в контексте базы данных, в которой он создается.

{ BEFORE | AFTER }. Ключевое слово BEFORE означает, что функция должна выполняться *перед* попыткой выполнения операции, включая все встроенные проверки ограничений данных, реализуемые при выполнении команд INSERT и DELETE. Ключевое слово AFTER означает, что функция вызывается после завершения операции, приводящей в действие триггер.

{ *событие* [OR *событие ...*] }. События SQL, поддерживаемые вPostgreSQL: INSERT, UPDATE или DELETE. При перечислении нескольких событий в качестве разделителя используется ключевое слово OR.

ON *таблица.* Имя таблицы, модификация которой заданным событием приводит к срабатыванию триггера.

FOR EACH { ROW | STATEMENT }. Ключевое слово, следующее за конструкцией FOR EACH и определяющее количество вызовов функции при наступлении указанного события. Ключевое

слово ROW означает, что функция вызывается *для каждой модифицируемой записи.* Если функция должна вызываться всего один раз для всей команды, используется ключевое словоSTATEMENT.

EXECUTE PROCEDURE *функция* (*аргументы*). Имя вызываемой функции с аргументами. На практике аргументы при вызове триггерных функций не используются.

Синтаксис определения триггерной функции

CREATE FUNCTION ***функция*** () RETURNS ***trigger*** AS '
DECLARE
объявления ;
BEGIN
команды;
END; '
LANGUAGE plpgsql;

В триггерных функциях используются специальные переменные, содержащие информацию о сработавшем триггере. С помощью этих переменных триггерная функция работает с данными таблиц. Ниже перечислены некоторые специальные переменные, доступные в триггерных функциях

Имя	**Тип**	**Описание**
NEW	RECORD	Новые значения полей записи базы данных, созданной командой INSERT или обновленной командой UPDATE,при срабатывании триггера уровня записи (ROW). Переменная используется для модификации новых записей. Внимание !!! Переменная NEW доступна только при операциях INSERT и UPDATE. Поля записи NEW могут быть изменены триггером.
OLD	RECORD	Старые значения полей записи базы данных, содержавшиеся в записи перед выполнением команды DELETE или UPDATE при срабатывании триггера уровня записи (ROW) Внимание !!! Переменная OLD доступна только при операциях DELETE и UPDATE. Поля записи OLD можно использовать только для чтения, изменять нельзя.

TG_NAME	name	Имя сработавшего триггера.
TG_WHEN	text	Строка BEFORE или AFTER в зависимости от момента срабатывания триггера, указанного в определении (до или после операции).
TG_LEVEL	text	Строка ROW или STATEMENT в зависимости от уровня триггера, указанного в определении.
TG_OP	text	Строка INSERT, UPDATE или DELETE в зависимости от операции, вызвавшей срабатывание триггера.
TG_RELID	oid	Идентификатор объекта таблицы, в которой сработал триггер.
TG_RELNAME	name	Name Имя таблицы, в которой сработал триггер.

К отдельным полям записей NEW и OLD в триггерных процедурах обращаются следующим образом: NEW.names , OLD.rg.

Примеры создания триггеров

Пример 1. Триггер выполняется перед удалением записи из таблицы поставщиков s. Триггер проверяет наличие в таблице поставок spj записей, относящихся к удаляемому поставщику, и, если такие записи есть, удаляет их.

```
-- Создание триггерной функции
CREATE FUNCTION trigger_s_before_del () RETURNS trigger AS '
BEGIN
if (select count(*) from spj a where trim(a.ns)=trim(OLD.ns))>0
then delete from spj where trim(spj.ns)=trim(OLD.ns);
end if;
return OLD;
END;
' LANGUAGE  plpgsql;

-- Создание триггера
CREATE TRIGGER tr_s_del_befor
BEFORE DELETE ON s FOR EACH ROW
EXECUTE PROCEDURE trigger_s_before_del();
--Проверка работы триггера
Delete from s where ns='S2';
```

Пример 2. Создание триггера-генератора для таблицы поставщиков s.

Триггер выполняется перед вставкой новой записи в таблицу поставщиков s. Триггер проверяет значения, которые должна содержать новая запись (record NEW) и может их изменить:

- если не указан номер поставщика – он генерируется по схеме – S+ уникальный номер из последовательности;
- если не указано имя поставщика – оно генерируется по схеме – Postawchik_ + уникальный номер из последовательности;
- если не указан город – ставится значение по умолчанию – "Novosibirsk" ;
- если не указан рейтинг или рейтинг <=0 – устанавливается рейтинг = 10 для поставщиков из Novosibirska и 0 для всех остальных.

```
-- Создание последовательности
CREATE SEQUENCE s_seq INCREMENT BY 1 START WITH 25;
-- Создание триггерной функции
-- в этой функции вызывается перегружаемая функция nvl, ее определение здесь
CREATE FUNCTION trigger_s_before_lns () RETURNS trigger AS '
BEGIN
NEW.ns=nvl(NEW.ns,'S'||trim(to_char(nextval('s_seq'),'99999')));
NEW.names=nvl(NEW.names,'Postawchik_'||trim(to_char(currval('s_seq'),'99999')));
NEW.town = nvl(NEW. town, 'Novosibirsk' );
 if (nvl(NEW.rg,0)<=0)  then
      If  NEW.town= 'Novosibirsk'   then  NEW.rg=10;
     else NEW.rg=0;
     end if;
end if;
return NEW;
END;
' LANGUAGE  plpgsql;
-- Создание триггера
CREATE TRIGGER s_bi
BEFORE INSERT ON s FOR EACH ROW
EXECUTE PROCEDURE trigger_s_before_lns ()

--Проверка работы триггера
insert into s values(null,null,null,null);
```

```
insert into s values(null,null,null,null);
insert into s values(null,null,null,null);
insert into s values(null,'Ivanov',null,null);
insert into s values(null,'Sidorov',50,null);
insert into s values(null,'Petrov',null,'Moskva');
```

--Результат

S25	Postawchik_25	10	Novosibirsk
S26	Postawchik_26	10	Novosibirsk
S27	Postawchik_27	10	Novosibirsk
S28	Petrov	0	Moskva
S29	Ivanov	10	Novosibirsk
S30	Sidorov	50	Novosibirsk