МИНИСТЕРСТВО ВЫСШЕГО ОБРАЗОВАНИЯ И НАУКИ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Астахова И. Ф., Борисенков Д. В., Маковий К. А., Хицкова Ю. В.

Учебное пособие по работе с СУБД PostgreSQL

Воронеж 2023

УДК 004.066

ББК 22.18

С 89

Астахова И. Ф. Учебное пособие по работе с СУБД PostgreSQL / И. Ф. Астахова, Д. В. Борисенков, К. А. Маковий, Ю. В. Хицкова – Воронеж: Издательский дом Воронежского государственного университета, 2023. – 127 с.

Настоящее пособие подготовлено сотрудниками кафедры Математического обеспечения ЭВМ факультета прикладной математики, информатики и механики. Оно предназначено для начального обучения работе с базами данных и рекомендуются студентам бакалавриата направления 01.03.02 «Прикладная математика и информатика» для использования в курсе Б1.Б.18 «Базы данных».

Пособие также может быть использовано студентами, аспирантами и научными работниками всех направлений, использующими СУБД PostgreSQL. Авторы с благодарностью примут любые замечания, пожелания и исправления, которые будут способствовать улучшению качества пособия, по адресу 394018, г. Воронеж, Университетская пл., д. 1, ком. 8.

e-mail: ifa@amm.vsu.ru

Рецензент: д.т.н., доц.ф-та ПММ Бондаренко Ю.В.

к.ф.-м.н., доц. ф-та ПММ Ухлова В.

Содержание

Введение

Информационные системы с базами данных являются в настоящее время одной из важнейших областей современных компьютерных технологий. С этой сферой связана большая часть современного рынка программных продуктов. Одной из общих тенденций в развитии информационных систем с базами данных являются процессы интеграции и стандартизации, затрагивающие структуры данных и способы их обработки и интерпретации, системное и прикладное программное обеспечение, средства разработки систем взаимодействия их компонентов и т. д. Основой современных систем управления базами данных (СУБД) является реляционная модель представления данных – в значительной степени благодаря простоте и четкости ее концептуальных понятий и строгому математическому обоснованию.

Данное пособие предназначено в первую очередь для преподавателей и студентов и ориентировано на обучение запросам манипулирования базами данных, которые лежат в основе языка SQL. Структурированный язык запросов к реляционным базам данных SQL и его процедурные расширения в настоящее время являются стандартными средствами, поддерживаемыми всеми современными системами управления базами данных, их знание необходимо любому специалисту в области современных информационных технологий. Эта книга посвящена изучению диалекта языка SQL СУБД PostgreSQL и процедурного расширения языка SQL этой СУБД PL/pgSQL.

Авторы надеются, что пособие окажется полезным не только для преподавателей и студентов, но и для других читателей, заинтересованных в получении начальных навыков использования СУБД PostgreSQL.

1. Установка СУБД, основные процессы и файлы

1.1. Установка СУБД в ОС Windows

Скачайте самоустанавливающийся дистрибутив СУБД PostgreSQL для ОС Windows с сайта российской компании Postgres Professional (https://postgrespro.ru/windows) и запустите его. Выберите язык установки. Компоненты устанавливаемой программы можно выбрать по умолчанию:

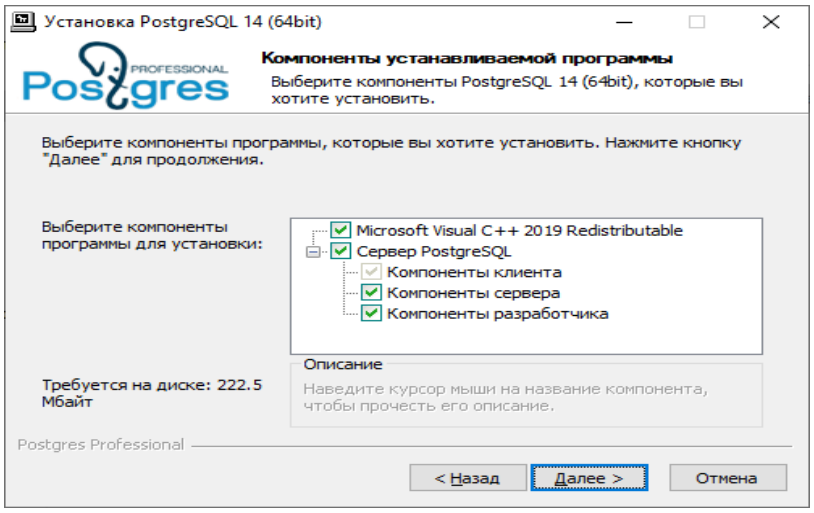

Рис.1. Выбор компонентов устанавливаемой программы

Далее следует выбрать каталог для установки СУБД, по умолчанию – C:\Program Files\PostgreSQL\14 (здесь 14 – номер версии СУБД PostgreSQL, для другого номера версии СУБД имя каталога соответственно изменится). Отдельно можно выбрать расположение каталога баз данных.

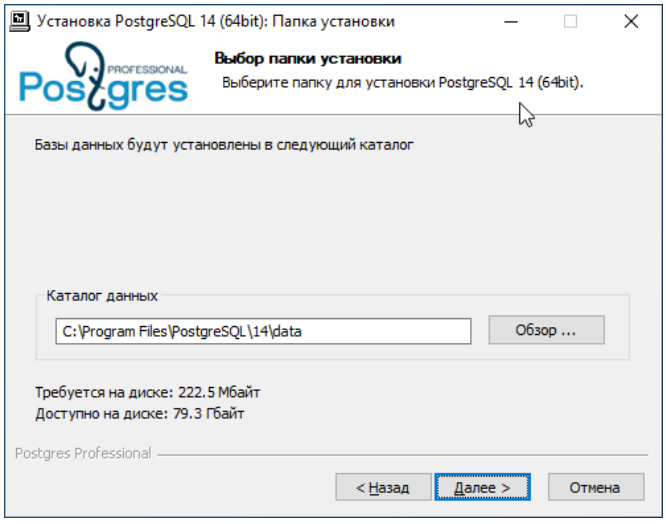

Рис.2 Выбор каталога для установки баз данных

Именно здесь будет находиться хранимая в СУБД информация, так что убедитесь, что на диске достаточно места, если вы планируете хранить много данных. Параметры сервера:

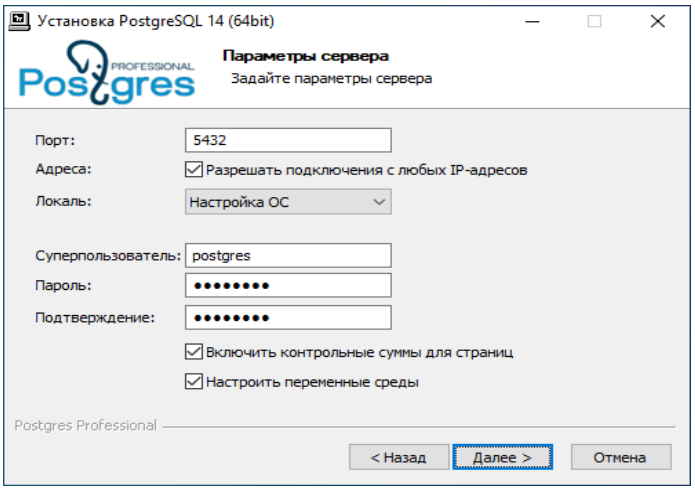

Рис.3. Выбор параметров сервера

Если вы планируете хранить данные на русском языке, выберите «Russian, Russia» (или оставьте вариант «Настройка ОС», если в Windows используется русская локаль). Введите (и подтвердите повторным вводом) пароль пользователя postgres (суперпользователя СУБД PostgreSQL). Также отметьте флажок «Настроить переменные среды», чтобы подключаться к серверу PostgreSQL под текущим пользователем ОС. Для остальных полей можно оставить значения по умолчанию.

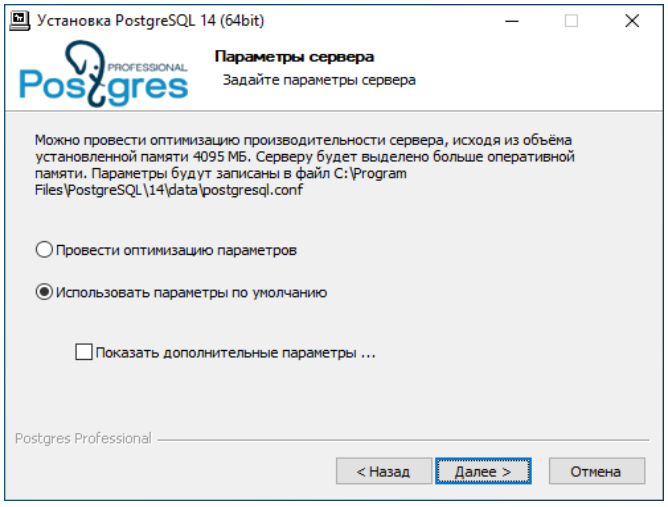

Рис.4 Установка PostgreSQL для ознакомительных целей

Если вы планируете установить PostgreSQL только для ознакомительных целей, можно отметить вариант «Использовать параметры по умолчанию», чтобы СУБД не занимала много оперативной памяти.

1.2. Управление службой и основные файлы в ОС Windows

При установке PostgreSQL в вашей системе регистрируется служба «postgresql-14». Она запускается автоматически при старте компьютера под учетной записью Network Service (Сетевая служба). При необходимости вы можете изменить параметры службы с помощью стандартных средств Windows.

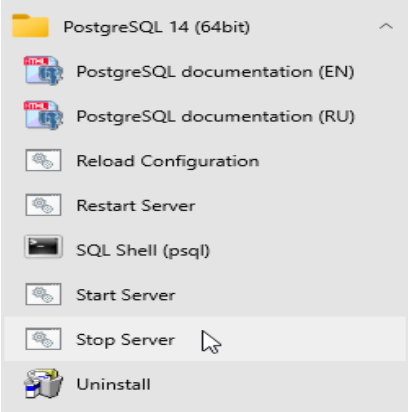

Рис.5 Работа с сервером

Чтобы временно остановить службу сервера баз данных, выполните пункт меню «Stop Server» из папки в меню «Пуск», которую вы указали при установке. Для запуска службы там же находится пункт меню «Start Server». Если при запуске службы произошла ошибка, для поиска причины следует заглянуть в журнал сообщений сервера. Он находится в подкаталоге log каталога, выбранного при установке для баз данных (обычно C:\Program Files\PostgreSQL\14\data\log). Журнал настроен так, чтобы запись периодически переключалась в новый файл. Найти актуальный файл можно по дате последнего изменения или по имени, которое содержит дату и время переключения. Есть несколько важных

конфигурационных файлов, которые определяют настройки сервера. Они располагаются в каталоге баз данных. Вам не нужно их изменять, чтобы начать знакомство с PostgreSQL, но в реальной работе они непременно потребуются:

– postgresql.conf – основной конфигурационный файл, содержащий значения параметров сервера;
– pg_hba.conf – файл, определяющий настройки доступа. В целях безопасности по умолчанию доступ должен быть подтвержден паролем и допускается только с локального компьютера.

Обязательно загляните в эти файлы – они прекрасно документированы. Теперь вы готовы подключиться к базе данных и попробовать некоторые команды и запросы.

1.3. Установка СУБД в ОС Linux (Debian, Ubuntu)

Если вы используете Linux, то для установки необходимо подключить пакетный репозиторий PGDG (PostgreSQL Global Development Group). В настоящее время для системы Debian поддерживаются версии 9 «Stretch», 10 «Buster» и 11 «Bullseye», а для Ubuntu – 18.04 «Bionic», 20.04 «Focal», 21.04 «Hirsute» и 21.10 «Impish».

Выполните в терминале следующие команды:

```
$ sudo apt-get install lsb-release
$ sudo sh -c 'echo "deb \ http://apt.postgresql.org/pub/repos/apt/\
  (lsb_release -cs)-pgdg main" \ > /etc/apt/sources.list.d/pgdg.list'
$ wget --quiet -O - \ https://postgresql.org/media/keys/ACCC4CF8.asc\
  | sudo apt-key add –
```

Репозиторий подключен, обновим список пакетов:

```
$ sudo apt-get update
```

Перед установкой проверьте настройки локализации:

```
$ locale
```

Если вы планируете хранить данные на русском языке, значения переменных LC_CTYPE и LC_COLLATE должны быть равны «ru_RU.UTF8» (значение «en_US.UTF8» тоже подходит, но менее предпочтительно).

При необходимости установите эти переменные:

```
$ export LC_CTYPE=ru_RU.UTF8
$ export LC_COLLATE=ru_RU.UTF8
```

Также убедитесь, что в операционной системе установлена соответствующая локаль (набор параметров, определяющий региональные настройки пользовательского интерфейса) :

```
$ locale -a | grep ru_RU ru_RU.utf8
```

Если это не так, сгенерируйте ее:

```
$ sudo locale-gen ru_RU.utf8
```

Теперь можно приступить к установке:

```
$ sudo apt-get install postgresql-14
```

Как только эта команда выполнится, СУБД PostgreSQL будет установлена, запущена и готова к работе. Чтобы проверить это, выполните команду:

```
$ sudo -u postgres psql -c 'select now()'
```

Если все проделано успешно, в ответ вы должны получить текущее время.

1.4. Управление службой и основные файлы в ОС Linux

При установке PostgreSQL в вашей операционной системе автоматически создается специальный пользователь postgres, от имени которого работают процессы, обслуживающие сервер. Этому пользователю принадлежат все файлы, относящиеся к СУБД. PostgreSQL будет автоматически запускаться при перезагрузке операционной системы. С настройками по умолчанию это не проблема: если вы не работаете с

сервером базы данных, он потребляет совсем немного ресурсов вашей системы. Если вы все-таки захотите отключить автозапуск, выполните:

```
$ sudo systemctl disable postgresql
```

Чтобы временно остановить службу сервера баз данных, выполните команду:

```
$ sudo systemctl stop postgresql
```

Запустить службу сервера можно командой:

```
$ sudo systemctl start postgresql
```

Можно также проверить текущее состояние:

```
$ sudo systemctl status postgresql
```

Если служба не запускается, найти причину поможет журнал сообщений сервера. Внимательно прочитайте самые последние записи из журнала, который находится в файле /var/log/postgresql/postgresql-14-main.log.

2. База данных

2.1. Структура учебной базы данных

Будем рассматривать работу с СУБД PostgreSQL на примере описанной далее базы данных. integer

Таблица STUDENT (Студент)

student_id (integer) – идентификатор студента;

surname (text) – фамилия студента;

name (text) – имя студента;

stipend (integer) – стипендия, которую получает студент;

kurs (integer) – курс, на котором учится студент;

city (text) – город, в котором живет студент;

birthday (date) – дата рождения студента;

univ_id (integer) – идентификатор университета, в котором учится студент.

Таблица LECTURER (Преподаватель)

lecturer_id (integer) – идентификатор преподавателя;

surname (text) – фамилия преподавателя;

name (text) – имя преподавателя;

city (text) – город, в котором живет преподаватель;

univ_id (integer) – идентификатор университета, в котором работает преподаватель.

Таблица SUBJECT (Предмет обучения)

subj_id (integer) – идентификатор предмета обучения;

subj_name (text) – наименование предмета обучения;

hour (integer) – количество часов, отводимых на изучение предмета;

semestr (integer) – семестр, в котором изучается данный предмет.

Таблица UNIVERSITY (Университет)

univ_id (integer) – идентификатор университета;

univ_name (text) – название университета;

rating (integer) – рейтинг университета;

city (text) – город, в котором расположен университет.

Таблица EXAM_MARKS (Экзаменационные оценки)

exam_id (integer)– идентификатор экзамена;

student_id (integer) – идентификатор студента;

subj_id (integer) – идентификатор предмета обучения;

mark (integer) – экзаменационная оценка;

exam_date (date) – дата экзамена.

Таблица SUBJ_LECT (Учебные дисциплины преподавателей)

lecturer _id (integer) – идентификатор преподавателя;

subj_id (integer) – идентификатор предмета обучения.

2.2. Создание базы данных

Давайте создадим новую базу данных с именем department. Выполним:

```
postgres=# CREATE DATABASE department;
CREATE DATABASE
```

Не забудьте про точку с запятой в конце команды – пока PostgreSQL не увидит этот символ, он будет считать, что вы продолжаете ввод (так что команду можно разбить на несколько строк).

Теперь переключимся на созданную базу:

```
postgres=# \c department
```

Вы подключены к базе данных department как пользователь postgres.

```
department=#
```

Как видим, приглашение сменилось на department=#.

Команда, которую мы только что ввели, не похожа на SQL-запрос – она начинается с обратной косой черты. Это команда утилиты psql (SQL shell).

2.3. Создание таблиц

`В реляционных СУБД данные представляются в виде **таблиц**. Заголовок таблицы определяет **столбцы;** собственно данные располагаются в **строках**. Данные не упорядочены (в частности, нельзя полагаться на то, что строки хранятся в порядке их добавления в таблицу).

Для каждого столбца устанавливается **тип данных**; значения соответствующих полей строк должны соответствовать этим типам. PostgreSQL располагает большим числом встроенных типов (https://postgrespro.ru/doc/datatype) и возможностями для создания новых, но мы пока ограничимся несколькими основными:

- integer – целые числа;
- text – текстовые строки;
- date – даты;
- boolean – логические значения true (истина) или false (ложь).

Давайте создадим таблицу дисциплин, читаемых в вузе:

```
department=# CREATE TABLE subject (
department=#     subj_id integer PRIMARY KEY,
department=#     subj_name text,
```

```
department=#     hour integer,
department=#     semestr integer);
CREATE TABLE
```

В этой команде мы определили, что таблица с именем `subject` будет состоять из четырех столбцов: `subj_id` – номер курса, `subj_name` – название курса, `hour` – целое число лекционных часов, `semestr` – номер семестра, в котором читается предмет.

Кроме столбцов (их имен и типов данных), мы можем также определить ограничения целостности, которые будут автоматически проверяться – СУБД не допустит появление в базе некорректных данных. В нашем примере мы добавили ограничение `PRIMARY KEY` для столбца `subj_id`, которое говорит о том, что значения в этом столбце должны быть уникальными, а неопределенные значения не допускаются. Такой столбец можно использовать для того, чтобы отличить одну строку в таблице от других. Полный список ограничений целостности: https://postgrespro.ru/doc/ddl-constraints.

Точный синтаксис команды `CREATE TABLE` можно посмотреть в документации, а можно непосредственно в `psql`:

```
department=# \help CREATE TABLE
```

Такая справка есть по каждой команде SQL, а полный список команд покажет команда `\help` без параметров.

Помимо обычных значений, определяемых типом данных, столбец может содержать **неопределенное значение** (`NULL`-значение) – его можно рассматривать как «неизвестное значение» или «значение не задано».

2.4. Заполнение таблиц

Добавим в созданную таблицу несколько строк:

```
department=# INSERT INTO subject
      (subj_id, subj_name, hour, semestr)
      VALUES (101, 'Базы данных', 32, 5),
             (102, 'Сети ЭВМ', 64, 6);
INSERT 0 2
```

Если вам требуется массовая загрузка данных из внешнего источника, команда INSERT – не лучший выбор; посмотрите на специально предназначенную для этого команду COPY (https://postgrespro.ru/doc/sql-copy).

Для дальнейших примеров нам потребуется еще две таблицы: студенты и экзаменационные оценки. Для каждого студента будем хранить его имя и курс; идентифицироваться он будет числовым номером студенческого билета.

```
department=# CREATE TABLE student
     ( student_id integer PRIMARY KEY,
       surname text,
       name text,
       stipend integer,
       kurs integer,
       city text,
       birthday date,
       univ_id integer );
CREATE TABLE
department=# INSERT INTO student
     (student_id, surname, name, stipend,
      kurs, city, birthday, univ_id)
     VALUES (1451, 'Иванова', 'Анна', 2014,
             2, 'Воронеж', '05/10/2000', 36),
```

```
(1452, 'Петров', 'Виктор', 2014,
 3, 'Воронеж', '15/02/2000', 36),
(1453, 'Сидорова', 'Елена', 1050,
 3, 'Воронеж', /02/2000',24);
INSERT 0 3
```

Строка таблицы экзаменационных оценок содержит оценку, полученную студентом по некоторой дисциплине. Таким образом, студенты и дисциплины связаны друг с другом отношением «многие ко многим»: один студент может сдавать экзамены по многим дисциплинам, а экзамен по одной дисциплине могут сдавать много студентов.

Строка в таблице экзаменов идентифицируется совокупностью номера студбилета и номера курса. Такое ограничение целостности, относящее сразу к нескольким столбцам, определяется с помощью фразы CONSTRAINT:

```
department =# CREATE TABLE exam_marks
  (student_id integer REFERENCES student(student_id),
   subj_id integer REFERENCES subject(subj_id),
   mark integer,
   exam_date date,
   CONSTRAINT e_pk PRIMARY KEY (student_id, subj_id));
CREATE TABLE
```

Кроме того, с помощью фразы REFERENCES мы определили два ограничения ссылочной целостности, называемые **внешними ключами**. Такие ограничения показывают, что значения в одной таблице **ссылаются** на строки в другой таблице.

Теперь при любых действиях СУБД будет проверять, что все идентификаторы student_id, указанные в таблице экзаменов,

соответствуют реальным студентам (то есть строкам в таблице студентов), а номера subj_id – реальным курсам. Таким образом, будет исключена возможность оценить несуществующего студента или поставить оценку по несуществующей дисциплине — независимо от действий пользователя или возможных ошибок в приложении.

Поставим нашим студентам несколько оценок:

```
department=# INSERT INTO exam_marks
    (student_id, subj_id, mark, exam_date)
     VALUES (1452, 101, 5, '04/01/2022'),
            (1453, 102, 4, '14/01/2022'),
            (1452, 101, 3, '04/01/2022'),
            (1453, 102, 4, '14/01/2022');
INSERT 0 4
```

3. Встроенные функции и типы данных

3.1. Нововведения в PostgreSQL 14

Введены новые **мультидиапазонные типы данных** для представления наборов непересекающихся и непустых диапазонов.

Появилась **поддержка индексного обращения** (в квадратных скобках) не только к массивам, но и к другим типам данных. Сейчас такой синтаксис работает для JSON-массивов внутри jsonb и для hstore.

Новые функции: **date_bin** для округления даты до произвольного интервала, агрегатная функция **bit_xor** в компанию к имеющимся bit_and и bit_or, **bit_count** для подсчета числа единиц в двоичной строке, **unistr** для работы со спецпоследовательностями Unicode, **trim_array** для усечения массивов и **ltrim** с **rtrim** для усечения двоичных строк, **string_to_table** для представления частей текстовой строки в виде таблицы.

Функция **extract** возвращает **numeric**; раньше она возвращала double precision, как и date_part.

У типа **numeric появились бесконечные значения** `'Infinity'` и `'-Infinity'`. Это не только логично само по себе, но и позволяет без ограничений приводить к типу numeric числа с плавающей точкой, которые могут принимать бесконечные значения.

Тип **pg_lsn** обзавелся операторами для добавления и вычитания байтов.

Появились **хеш-функции для составных типов**, что позволяет использовать записи в операциях типа `UNION` и в качестве ключей в соединениях хешированием и в секционировании по хешу.

3.2. Типы данных

Табл. 1. Типы данных PostgreSQL 14

Имя типа	**Псевдонимы**	**Описание**
bigint	int8	знаковое целое из 8 байт
bigserial	serial8	восьмибайтное целое с автоувеличением
bit [(*n*)]		битовая строка фиксированной длины
bit varying [(*n*)]	varbit [(*n*)]	битовая строка переменной длины
boolean	bool	логическое значение (true/false)
box		прямоугольник в плоскости
bytea		двоичные данные («массив байт»)
character [(*n*)]	char [(*n*)]	символьная строка фиксированной длины
character varying [(*n*)]	varchar [(*n*)]	символьная строка переменной длины
cidr		сетевой адрес IPv4 или IPv6
circle		круг в плоскости
date		календарная дата (год, месяц, день)
double precision	float8	число двойной точности с плавающей точкой (8 байт)
inet		адрес узла IPv4 или IPv6
integer	int, int4	знаковое четырёхбайтное целое

Имя типа	**Псевдонимы**	**Описание**
interval [*поля*] [(*p*)]		интервал времени
json		текстовые данные JSON
jsonb		двоичные данные JSON, разобранные
line		прямая в плоскости
lseg		отрезок в плоскости
macaddr		MAC-адрес
macaddr8		адрес MAC (Media Access Control) (в формате EUI-64)
money		денежная сумма
numeric [(*p*, *s*)]	decimal [(*p*, *s*)]	вещественное число заданной точности
path		геометрический путь в плоскости
pg_lsn		последовательный номер в журнале Postgres Pro
pg_snapshot		снимок идентификатора транзакций

4. Язык PL/pgSQL

4.1. Операторы управления программой

Операторы IF и CASE позволяют выполнять команды в зависимости от определённых условий. PL/pgSQL поддерживает три формы IF:

- IF ... THEN ... END IF
- IF ... THEN ... ELSE ... END IF
- IF ... THEN ... ELSIF ... THEN ... ELSE ... END IF

и две формы CASE:

- CASE ... WHEN ... THEN ... ELSE ... END CASE
- CASE WHEN ... THEN ... ELSE ... END CASE

Конструкция IF-THEN – простейшая форма IF – IF логическое-выражение THEN операторы END IF. Операторы между THEN и END IF выполняются, если условие (логическое-выражение) истинно. В противном случае они пропускаются.

Пример:

```
IF v_user_id <> 0 THEN
UPDATE users
  SET email = v_email
  WHERE user_id = v_user_id;
END IF;
```

Конструкция IF-THEN-ELSE – IF логическое-выражение THEN операторы ELSE операторы END IF – добавляет к IF-THEN возможность указать альтернативный набор операторов, которые будут выполнены, если условие не истинно (в том числе, если условие принимает значение NULL).

Примеры:

```
IF parentid IS NULL OR parentid = '' 1201
THEN RETURN fullname;
ELSE RETURN hp_true_filename(parentid) || '/' ||
            fullname;
END IF;
IF v_count > 0 THEN
INSERT INTO users_count (count) VALUES (v_count);
RETURN 't';
ELSE RETURN 'f';
END IF;
```

Более гибким является синтаксис IF-THEN-ELSIF – IF логическое-выражение THEN операторы ELSIF логическое-выражение THEN операторы [ELSIF логическое-выражение THEN операторы . . .] [ELSE операторы] END IF. В некоторых случаях двух альтернатив недостаточно. IF-THEN-ELSIF обеспечивает удобный способ проверки нескольких вариантов по очереди. Условия в IF последовательно проверяются до тех пор, пока не будет найдено первое истинное. После этого операторы, относящиеся к этому условию, выполняются, и управление переходит к следующей после END IF команде. (Все последующие условия не проверяются.) Если ни одно из условий IF не является истинным, то выполняется блок ELSE (если он присутствует).

Пример:

```
IF number = 0 THEN result := 'zero';
ELSIF number > 0 THEN result := 'positive';
ELSIF number < 0 THEN result := 'negative';
ELSE -- number имеет значение NULL
result := 'NULL';
```

```
END IF;
```

Вместо ключевого слова ELSIF можно использовать ELSEIF. Другой вариант сделать то же самое – это использование вложенных операторов IF-THEN-ELSE, как в следующем примере:

```
IF demo_row.sex = 'm' THEN pretty_sex := 'man'; ELSE
  IF demo_row.sex = 'f' THEN pretty_sex := 'woman';
  END IF;
END IF;
```

Однако это требует написания соответствующих END IF для каждого IF, что при наличии нескольких альтернатив делает код более громоздким, чем использование ELSIF.

Простой CASE

```
CASE выражение-поиска
WHEN выражение [, выражение [...]]
THEN операторы [WHEN выражение [, выражение [...]]
THEN операторы ...] [ELSE операторы]
END CASE;
```

Простая форма CASE реализует условное выполнение на основе сравнения операндов. Выражениепоиска вычисляется (один раз) и последовательно сравнивается с каждым выражением в условиях WHEN. Если совпадение найдено, то выполняются соответствующие операторы и управление переходит к следующей после END CASE команде. (Все последующие выражения WHEN не проверяются.) Если совпадение не было найдено, то выполняются операторы в ELSE. Если же ELSE нет, то вызывается исключение CASE_NOT_FOUND. Пример:

```
CASE x WHEN 1, 2 THEN msg := 'один или два';
ELSE msg := 'значение, отличное от один или два';
```

```
END CASE;
```

CASE с перебором условий:

```
CASE WHEN логическое-выражение THEN операторы
[WHEN логическое-выражение THEN операторы ...]
[ELSE операторы]
END CASE;
```

Эта форма CASE реализует условное выполнение, основываясь на истинности логических условий. Каждое логическое-выражение в предложении WHEN вычисляется по порядку до тех пор, пока не будет найдено истинное. Затем выполняются соответствующие операторы и управление переходит к следующей после END CASE команде. (Все последующие выражения WHEN не проверяются.) Если ни одно из условий не окажется истинным, то выполняются операторы в ELSE. Если же ELSE нет, то вызывается исключение CASE_NOT_FOUND. Пример:

```
CASE WHEN x BETWEEN 0 AND 10
THEN msg := 'значение в диапазоне между 0 и 10';
WHEN x BETWEEN 11 AND 20
THEN msg := 'значение в диапазоне между 11 и 20';
END CASE;
```

Эта форма CASE полностью эквивалентна IF-THEN-ELSIF, за исключением того, что при невыполнении всех условий и отсутствии ELSE IF-THEN-ELSIF ничего не делает, а CASE вызывает ошибку.

Операторы циклов LOOP, EXIT, CONTINUE, WHILE, FOR и FOREACH позволяют повторить выполнение серии команд в коде на PL/pgSQL.

```
[<метка>] LOOP операторы END LOOP [ метка ];
```

LOOP организует безусловный цикл, который повторяется до бесконечности, пока не будет прекращён операторами EXIT или RETURN. Для вложенных циклов можно использовать метку в операторах EXIT и CONTINUE, чтобы указать, к какому циклу эти операторы относятся.

```
EXIT [метка] [WHEN логическое-выражение];
```

Если метка не указана, то завершается самый внутренний цикл, далее выполняется оператор, следующий за END LOOP. Если метка указана, то она должна относиться к текущему или внешнему циклу, или это может быть метка блока. При этом в именованном цикле/блоке выполнение прекращается, а управление переходит к следующему оператору после соответствующего END. При наличии WHEN цикл прекращается, только если логическое-выражение истинно. В противном случае управление переходит к оператору, следующему за EXIT. EXIT можно использовать со всеми типами циклов, не только с безусловным. Когда EXIT используется для выхода из блока, управление переходит к следующему оператору после окончания блока. Обратите внимание, что для выхода из блока нужно обязательно указывать метку. EXIT без метки не позволяет прекратить работу блока. Примеры:

```
LOOP -- здесь производятся вычисления
IF count > 0 THEN EXIT; -- выход из цикла
END IF;
END LOOP;
LOOP -- здесь производятся вычисления
EXIT WHEN count > 0; -- аналогично предыдущему
END LOOP;
BEGIN -- здесь производятся вычисления
IF stocks > 100000 THEN EXIT ablock;
```

```
-- выход из блока BEGIN
END IF;
-- вычисления не будут выполнены, если stocks > 100000 END;
CONTINUE [ метка ] [WHEN логическое-выражение];
```

Если метка не указана, то начинается следующая итерация самого внутреннего цикла. То есть все оставшиеся в цикле операторы пропускаются, и управление переходит к управляющему выражению цикла (если есть) для определения, нужна ли ещё одна итерация цикла. Если метка присутствует, то она указывает на метку цикла, выполнение которого будет продолжено. При наличии WHEN следующая итерация цикла начинается только тогда, когда логическое-выражение истинно. В противном случае управление переходит к оператору, следующему за CONTINUE. CONTINUE можно использовать со всеми типами циклов, не только с безусловным. Примеры:

```
LOOP -- здесь производятся вычисления
EXIT WHEN count > 100;
CONTINUE WHEN count < 50;
-- вычисления для count в диапазоне 50 .. 100
 END LOOP;
```

WHILE логическое-выражение LOOP операторы END LOOP [метка];

WHILE выполняет серию команд до тех пор, пока истинно логическое-выражение. Выражение проверяется непосредственно перед каждым входом в тело цикла.

Пример:

```
WHILE amount_owed > 0 AND
gift_certificate_balance > 0
```

```
LOOP -- здесь производятся вычисления
END LOOP;
WHILE NOT done LOOP -- здесь производятся вычисления
END LOOP;
 FOR (целочисленный вариант) [<>]
FOR имя IN [REVERSE] выражение
.. выражение [BY выражение]
 LOOP операторы
END LOOP [ метка ];
```

В этой форме цикла FOR итерации выполняются по диапазону целых чисел. Переменная имя автоматически определяется с типом integer и существует только внутри цикла (если уже существует переменная с таким именем, то внутри цикла она будет игнорироваться). Выражения для нижней и верхней границы диапазона чисел вычисляются один раз при входе в цикл. Если не указано BY, то шаг итерации 1, в противном случае используется значение в BY, которое вычисляется, опять же, один раз при входе в цикл. Если указано REVERSE, то после каждой итерации величина шага вычитается, а не добавляется.

Примеры целочисленного FOR:

```
FOR i IN 1..10 LOOP -- внутри цикла переменная
--i будет иметь значения 1,2,3,4,5,6,7,8,9,10
END LOOP;
FOR i IN REVERSE 10..1 LOOP -- внутри цикла
переменная
-- i будет иметь значения 10,9,8,7,6,5,4,3,2,1
END LOOP;
 FOR i IN REVERSE 10..1 BY 2 LOOP -- внутри цикла
 -- переменная i будет иметь значения 10,8,6,4,2
```

```
END LOOP;
```

Если нижняя граница цикла больше верхней границы (или меньше, в случае REVERSE), то тело цикла не выполняется вообще. При этом ошибка не возникает. Если с циклом FOR связана метка, к целочисленной переменной цикла можно обращаться по имени, указывая эту метку. Цикл по результатам запроса Другой вариант FOR позволяет организовать цикл по результатам запроса.

Синтаксис: [<>] FOR цель IN запрос LOOP операторы END LOOP [метка];

Переменная цель может быть строковой переменной, переменной типа record или разделённым запятыми списком скалярных переменных. Переменной цель последовательно присваиваются строки результата запроса, и для каждой строки выполняется тело цикла.

Пример:

```
CREATE FUNCTION refresh_mviews()
RETURNS integer AS $$
DECLARE mviews RECORD;
BEGIN RAISE NOTICE
'Refreshing all materialized views...';
FOR mviews IN SELECT n.nspname AS mv_schema,
c.relname AS mv_name,
 pg_catalog.pg_get_userbyid(c.relowner) AS owner
FROM pg_catalog.pg_class c
LEFT JOIN pg_catalog.pg_namespace n
ON (n.oid = c.relnamespace)
WHERE c.relkind = 'm' ORDER BY 1;
    LOOP -- Здесь mviews содержит одну запись с информацией о матпредставлении
```

Если цикл завершается по команде EXIT, то последняя присвоенная строка доступна и после цикла. В качестве запроса в этом типе оператора FOR может задаваться любая команда SQL, возвращающая строки. Чаще всего это SELECT, но также можно использовать и INSERT, UPDATE или DELETE с предложением RETURNING.

Цикл по элементам массива FOREACH очень похож на FOR. Отличие в том, что вместо перебора строк SQL-запроса происходит перебор элементов массива. Синтаксис цикла FOREACH:

```
[ <> ]
FOREACH цель [ SLICE число ] IN ARRAY выражение
LOOP операторы END LOOP [ метка ];
```

Без указания SLICE, или если SLICE равен 0, цикл выполняется по всем элементам массива, полученного из выражения. Переменной цель последовательно присваивается каждый элемент массива и для него выполняется тело цикла. Пример цикла по элементам целочисленного массива:

```
CREATE FUNCTION sum(int[]) RETURNS int
AS $$ DECLARE s int8 := 0; x int;
BEGIN
FOREACH x IN ARRAY $1 LOOP s := s + x;
END LOOP; RETURN s;
END; $$ LANGUAGE plpgsql;
```

Обход элементов проводится в том порядке, в котором они сохранялись, независимо от размерности массива. Как правило, цель это одиночная переменная, но может быть и списком переменных, когда элементы массива имеют составной тип (записи). В этом случае переменным присваиваются значения из последовательных столбцов

составного элемента массива. При положительном значении SLICE FOREACH выполняет итерации по срезам массива, а не по отдельным элементам. Значение SLICE должно быть целым числом, не превышающим размерности массива. Переменная цель должна быть массивом, который получает последовательные срезы исходного массива, где размерность каждого среза задаётся значением SLICE. Пример цикла по одномерным срезам:

```
CREATE FUNCTION scan_rows(int[])
RETURNS void AS $$
DECLARE x int[];
BEGIN
    FOREACH x SLICE 1
    IN ARRAY $1 LOOP RAISE NOTICE 'row = %', x;
    END LOOP;
END; $$ LANGUAGE plpgsql;
SELECT
scan_rows(ARRAY[[1,2,3],[4,5,6],[7,8,9],[10,11,1
2]]);
NOTICE: row = {1,2,3} NOTICE: row = {4,5,6}
NOTICE: row = {7,8,9} NOTICE: row = {10,11,12
```

4.2. Подпрограммы

Текст тела функции должен быть блоком. Структура блока:

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

Функция с параметрами:

```
CREATE FUNCTION sales_tax(subtotal real)
RETURNS real AS $$
BEGIN
  RETURN subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

Другой способ для определения параметра:

```
имя ALIAS FOR $n;
CREATE FUNCTION sales_tax(real)
    RETURNS real AS $$
DECLARE
   subtotal ALIAS FOR $1;
   BEGIN RETURN subtotal * 0.06;
   END; $$ LANGUAGE plpgsql;
```

Эти версии не эквивалентны, к ним обращаться надо по-разному:

```
CREATE FUNCTION instr(varchar, integer)
RETURNS integer AS $$
DECLARE
v_string ALIAS FOR $1;
index ALIAS FOR $2;
BEGIN -- вычисления, использующие v_string и index
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION concat_selected_fields
(in_t sometablename) RETURNS text AS $$
BEGIN RETURN in_t.f1 || in_t.f3 || in_t.f5 ||
```

```
            in_t.f7;
END;
$$ LANGUAGE plpgsql;
```

Когда функция на PL/pgSQL объявляется с выходными параметрами, им выдаются цифровые идентификаторы $n и для них можно создавать псевдонимы точно таким же способом, как и для обычных входных параметров. Выходной параметр – это фактически переменная, стартующая с NULL и которой присваивается значение во время выполнения функции. Возвращается последнее присвоенное значение. Например, функция sales_tax может быть переписана так:

```
CREATE FUNCTION sales_tax(subtotal real,
    OUT tax real) AS $$
BEGIN tax := subtotal * 0.06;
END;
$$ LANGUAGE plpgsql;
```

Обратите внимание, что мы опустили `RETURNS real` — хотя можно было и включить, но это было бы излишним. Чтобы вызвать функцию с параметрами `OUT`, при вызове функции не указывайте выходные параметры:

```
SELECT sales_tax(100.00);
CREATE FUNCTION concat_selected_fields
(in_t sometablename) RETURNS text AS $$
BEGIN RETURN in_t.f1 || in_t.f3 || in_t.f5 ||
            in_t.f7;
END;
$$ LANGUAGE plpgsql
CREATE FUNCTION sales_tax(real) RETURNS real AS $$
```

```
DECLARE subtotal ALIAS FOR $1;
BEGIN RETURN subtotal * 0.06; END;
$$ LANGUAGE plpgsql;
```

Пример 1. Описать процедуру, которая заносит данные в таблицу LECTURER в зависимости от параметров процедуры, вызвать эту процедуру.

```
CREATE OR REPLACE FUNCTION insert_lecturer(
  p_lect_id lecturer.lect_id%TYPE,
  p_surname lecturer.surname%TYPE,
  p_name lecturer.name%TYPE,
  p_city lecturer.city%TYPE,
  p_univ_id lecturer.univ_id%TYPE
)
RETURNS VOID AS $$
BEGIN
  BEGIN
     INSERT INTO lecturer(lect_id, surname, name, city,
                          univ_id)
    VALUES(p_lect_id, p_surname, p_name, p_city,
           p_univ_id);
    RAISE NOTICE 'Data inserted successfully.';
  EXCEPTION
    WHEN OTHERS THEN
      RAISE NOTICE 'Error: %', SQLERRM;
  END;
END;
$$ LANGUAGE plpgsql;
```

```
-- Выполнение
SELECT insert_lecturer(10, 'John', 'Dorian', 'Липецк',
22);
```

Пример 2. В информационной системе клуба любителей скачек должна быть представлена информация об участвующих в скачках лошадях (кличка, пол, возраст), их владельцах (имя, адрес, телефон) и жокеях (имя, адрес, возраст, рейтинг). Необходимо сформировать таблицы для хранения информации по каждому состязанию: дата, время и место проведения скачек (ипподром), название состязаний (если таковое имеется), клички участвующих в заездах лошадей и имена жокеев, занятые ими места и показанное в заезде время.

Требуется:

– сформировать структуру таблиц базы данных;
– подобрать подходящие имена таблицам и их полям;
– обеспечить требования нормализации таблиц базы данных;
– сформировать SQL-запросы для создания таблиц базы данных с указанием первичных и внешних ключей и требуемых ограничений. Создать пакет с курсором и триггеры.

Решение:

```
-- Создание таблицы HORSES
CREATE TABLE HORSES (
  horse_id SERIAL PRIMARY KEY,
  horse_name VARCHAR(50) NOT NULL,
  gender VARCHAR(10),
  birth_date DATE
);
-- Создание таблицы OWNERS
CREATE TABLE OWNERS (
```

```
  owner_id SERIAL PRIMARY KEY,
  owner_name VARCHAR(50) NOT NULL,
  address VARCHAR(100),
  phone VARCHAR(20)
);
-- Создание таблицы JOCKEYS
CREATE TABLE JOCKEYS (
  jockey_id SERIAL PRIMARY KEY,
  jockey_name VARCHAR(50) NOT NULL,
  address VARCHAR(100),
  birth_date DATE,
  rating INTEGER
);
-- Создание таблицы HIPPODROMES
CREATE TABLE HIPPODROMES (
  hippodrome_id SERIAL PRIMARY KEY,
  hippodrome_name VARCHAR(50) NOT NULL,
  length DECIMAL(10, 2),
  address VARCHAR(100)
);
-- Создание таблицы RACES
CREATE TABLE RACES (
  race_id SERIAL PRIMARY KEY,
  race_date DATE,
  race_time TIME,
  hippodrome_id INTEGER REFERENCES HIPPODROMES
    (hippodrome_id),
  race_name VARCHAR(100)
);
```

```
-- Создание таблицы RACE_RESULTS
CREATE TABLE RACE_RESULTS (
  result_id SERIAL PRIMARY KEY,
  race_id INTEGER REFERENCES RACES(race_id),
  horse_id INTEGER REFERENCES HORSES(horse_id),
  jockey_id INTEGER REFERENCES JOCKEYS(jockey_id),
  place INTEGER,
  race_time TIME
);
-- Добавление ограничения на уникальность клички
лошади
ALTER TABLE HORSES ADD CONSTRAINT UK_HORSES_NAME
UNIQUE (horse_name);

-- Создание функции для добавления новой лошади
CREATE OR REPLACE FUNCTION add_horse(
  horse_name VARCHAR,
  gender VARCHAR,
  birth_date DATE
) RETURNS VOID AS $$
BEGIN
  INSERT INTO horses (h_name, gender, h_birth)
  VALUES (horse_name, gender, birth_date);
END;
$$ LANGUAGE plpgsql;
-- Создание функции для добавления нового владельца
CREATE OR REPLACE FUNCTION add_owner(
  owner_name VARCHAR,
  address VARCHAR,
```

```
  phone VARCHAR
) RETURNS VOID AS $$
BEGIN
  INSERT INTO owners (o_name, o_address, o_phone)
  VALUES (owner_name, address, phone);
END;
$$ LANGUAGE plpgsql;
-- Создание функции для добавления нового жокея
CREATE OR REPLACE FUNCTION add_jockey(
  jockey_name VARCHAR,
  address VARCHAR,
  birth_date DATE,
  rating INTEGER
) RETURNS VOID AS $$
BEGIN
  INSERT INTO jockeys (j_name, j_address, j_birth,
    rating)
  VALUES (jockey_name, address, birth_date, rating);
END;
$$ LANGUAGE plpgsql;
-- Создание функции для добавления нового ипподрома
CREATE OR REPLACE FUNCTION add_hippodrome(
  hippodrome_name VARCHAR,
  length DECIMAL,
  address VARCHAR
) RETURNS VOID AS $$
BEGIN
  INSERT INTO hippodromes (hi_name, length,
    hi_address)
```

```
  VALUES (hippodrome_name, length, address);
END;
$$ LANGUAGE plpgsql;
-- Создание функции для добавления нового состязания
CREATE OR REPLACE FUNCTION add_race(
  race_date DATE,
  race_time TIME,
  hippodrome_id INTEGER,
  race_name VARCHAR
) RETURNS VOID AS $$
BEGIN
  INSERT INTO competitions (c_date, c_time, hi_id,
    race_name)
  VALUES (race_date, race_time, hippodrome_id,
    race_name);
END;
$$ LANGUAGE plpgsql;
-- Создание функции для добавления результата
состязания
CREATE OR REPLACE FUNCTION add_race_result(
  race_id INTEGER,
  horse_id INTEGER,
  jockey_id INTEGER,
  place INTEGER,
  race_time TIME
) RETURNS VOID AS $$
BEGIN
  INSERT INTO race_results (c_id, h_id, j_id, place,
    result_time)
```

```
  VALUES (race_id, horse_id, jockey_id, place,
    race_time);
END;
$$ LANGUAGE plpgsql;

-- Вызовы функций
SELECT add_horse('Лошадь1', 'Кобыла', '2022-01-01');
SELECT add_owner('Владелец1', 'Адрес1', '123-456-
7890');
SELECT add_jockey('Жокей1', 'Адрес2', '2000-02-02',
5);
SELECT add_hippodrome('Ипподром1', 10.5, 'Адрес3');
SELECT add_race('2022-05-01', '14:30:00', 1,
'Состязание1');
SELECT add_race_result(1, 1, 1, 1, '00:55:30');
```

Задания.

1. Описать процедуру, которая заносит данные в таблицу STUDENT в зависимости от параметров процедуры, вызвать эту процедуру.
2. Описать процедуру, которая заносит данные в таблицы STUDENT и EXAM_MARKS с соблюдением ссылочной целостности, воспользоваться ей в зависимости от параметров.
3. Описать процедуру, которая заносит данные в таблицы STUDENT и SUBJECT с соблюдением ссылочной целостности, воспользоваться ей в зависимости от параметров.
4. Описать процедуру, которая заносит данные в таблицы STUDENT и UNIVERSITY с соблюдением ссылочной целостности, воспользоваться ей в зависимости от параметров.

5. Описать процедуру, которая заносит данные в таблицу LECTURER в зависимости от параметров процедуры, вызвать эту процедуру.
6. Описать и вызвать функцию, которая вычисляет сумму баллов для студентов, номера которых находятся в заданном интервале в таблице EXAM_MARKS.
7. Описать и вызвать функцию, которая вычисляет для студентов, номера которых находятся в заданном диапазоне, максимальный балл по заданному предмету (с некоторым номером) в таблице EXAM_MARKS.
8. Описать и вызвать функцию, которая определяет для студентов с заданной фамилией сумму баллов по заданному предмету в таблице EXAM_MARKS.
9. Описать и вызвать функцию, которая вычисляет для студентов, номера которых находятся в данном диапазоне, максимальный балл по заданному названию предмета.
10. Описать и вызвать функцию, которая вычисляет для студентов, номера которых находятся в заданном диапазоне, средний балл по заданному названию предмета.
11. Описать и вызвать функцию, которая вычисляет для преподавателей, номера которых находятся в заданном диапазоне, максимальное количество часов по заданному названию предмета.
12. Описать и вызвать функцию, которая вычисляет для студентов, номера которых находятся в заданном диапазоне, максимальное количество часов по заданному названию предмета.
13. Описать и вызвать функцию, которая вычисляет для университета, номера которых находятся в заданном диапазоне, максимальное количество преподавателей, которые его читают, по заданному названию предмета.

4.3. Обработка ошибок

По умолчанию любая возникающая ошибка прерывает выполнение функции на PL/pgSQL и транзакцию, в которая она выполняется. Использование в блоке секции EXCEPTION позволяет перехватывать и обрабатывать ошибки. Синтаксис секции EXCEPTION расширяет синтаксис обычного блока:

```
[ <<метка>> ]
[ DECLARE объявления ]
BEGIN операторы
EXCEPTION
    WHEN условие [ OR условие ... ] THEN
        операторы_обработчика
    [ WHEN условие [ OR условие ... ] THEN
        операторы_обработчика
    ... ]
END;
```

Если ошибок не было, то выполняются все операторы блока и управление переходит к следующему оператору после END. Но если при выполнении оператора происходит ошибка, то дальнейшая обработка прекращается и управление переходит к списку исключений в секции EXCEPTION. В этом списке ищется первое исключение, условие которого соответствует ошибке. Если исключение найдено, то выполняются соответствующие операторы_обработчика и управление переходит к следующему оператору после END. Если исключение не найдено, то ошибка передаётся наружу, как будто секции EXCEPTION не было. При этом ошибку можно перехватить в секции EXCEPTION внешнего блока.

Если ошибка так и не была перехвачена, то обработка функции прекращается.

Если задаётся имя категории, ему соответствуют все ошибки в данной категории. Специальному имени условия OTHERS (другие) соответствуют все типы ошибок, кроме QUERY_CANCELED и ASSERT_FAILURE. (И эти два типа ошибок можно перехватить по имени, но часто это неразумно.) Имена условий воспринимаются без учёта регистра. Условие ошибки также можно задать кодом SQLSTATE; например, эти два варианта равнозначны:

```
WHEN division_by_zero THEN ...
WHEN SQLSTATE '22012' THEN ...
```

Если при выполнении операторов_обработчика возникнет новая ошибка, то она не может быть перехвачена в этой секции EXCEPTION. Ошибка передаётся наружу и её можно перехватить в секции EXCEPTION внешнего блока.

При выполнении команд в секции EXCEPTION локальные переменные функции на PL/pgSQL сохраняют те значения, которые были на момент возникновения ошибки. Однако все изменения в базе данных, выполненные в блоке, будут отменены. В качестве примера рассмотрим следующий фрагмент:

```
INSERT INTO mytab (firstname, lastname)
  VALUES('Tom', 'Jones');
BEGIN
  UPDATE mytab SET firstname = 'Joe'
    WHERE lastname = 'Jones';
  x := x + 1; y := x / 0;
  EXCEPTION
    WHEN division_by_zero THEN
```

```
        RAISE NOTICE
         'перехватили ошибку division_by_zero';
        RETURN x;
END;
```

При присваивании значения переменной y произойдёт ошибка division_by_zero. Она будет перехвачена в секции EXCEPTION. Оператор RETURN вернёт значение x, увеличенное на единицу, но изменения, сделанные командой UPDATE, будут отменены. Изменения, выполненные командой INSERT, которая предшествует блоку, не будут отменены. В результате база данных будет содержать 'Tom Jones', а не 'Joe Jones'.

Подсказка. Наличие секции EXCEPTION значительно увеличивает накладные расходы на вход/выход из блока, поэтому не используйте EXCEPTION без надобности.

Пример. Обработка исключений для команд UPDATE/INSERT.

В этом примере обработка исключений помогает выполнить либо команду UPDATE, либо INSERT, в зависимости от ситуации. Однако в современных приложениях вместо этого приёма рекомендуется использовать INSERT с ON CONFLICT DO UPDATE. Данный пример предназначен в первую очередь для демонстрации управления выполнением PL/pgSQL:

```
CREATE TABLE db (a INT PRIMARY KEY, b TEXT);
CREATE FUNCTION merge_db(key INT, data TEXT)
RETURNS VOID AS $$
BEGIN
  LOOP
    -- сначала попытаться изменить запись по ключу
```

```
    UPDATE db SET b = data WHERE a = key;
    IF found THEN RETURN;
    END IF;
    -- записи с таким ключом нет, поэтому её нужно
    -- добавить, если параллельно будет вставлена
    -- запись с таким же ключом, произойдёт ошибка
    -- уникальности
    BEGIN
      INSERT INTO db (a,b) VALUES (key, data);
      RETURN;
    EXCEPTION WHEN unique_violation THEN
    -- здесь не нужно ничего делать,
    -- просто продолжить цикл, чтобы повторить UPDATE.
    END;
  END LOOP;
END;
$$
LANGUAGE plpgsql;
SELECT merge_db(1, 'david');
SELECT merge_db(1, 'dennis');
```

В этом коде предполагается, что ошибка unique_violation вызывается самой командой `INSERT`, а не, скажем, внутренним оператором `INSERT` в функции триггера для этой таблицы.

Поведение также возможно, если в таблице будет несколько уникальных индексов; тогда операция будет повторяться вне зависимости от того, нарушение какого индекса вызвало ошибку. Используя средства, рассмотренные далее, можно сделать код более надёжным, проверяя, что перехвачена именно ожидаемая ошибка.

Получение информации об ошибке. При обработке исключений часто бывает необходимым получить детальную информацию о произошедшей ошибке. Для этого в PL/pgSQL есть два способа: использование специальных переменных и команда `GET STACKED DIAGNOSTICS`. Внутри секции `EXCEPTION` специальная переменная `SQLSTATE` содержит код ошибки, для которой было вызвано исключение. Специальная переменная `SQLERRM` содержит сообщение об ошибке, связанное с исключением. Эти переменные являются неопределёнными вне секции `EXCEPTION`. Также в обработчике исключения можно получить информацию о текущем исключении командой `GET STACKED DIAGNOSTICS`, которая имеет вид:

```
GET STACKED DIAGNOSTICS переменная { = | := }
     элемент [ , ... ];
```

Каждый элемент представляется ключевым словом, указывающим, какое значение состояния нужно присвоить заданной переменной (она должна иметь подходящий тип данных, чтобы принять его).

4.4. Курсоры

Доступ к курсорам в PL/pgSQL осуществляется через переменные курсора, которые имеют специальный тип данных `REFCURSOR`.

Существует два способа создания курсорной переменной: объявление ее как переменной типа `REFCURSOR` или использование синтаксиса объявления курсора. Синтаксис объявления курсора имеет следующий общий формат:

имя `[ [ NO ] SCROLL ] CURSOR [` (аргументы) `] FOR` запрос;

(`FOR` может быть заменен на `IS` для совместимости с Oracle).

Указание SCROLL позволяет прокручивать курсор назад, в то время как NO SCROLL запрещает прокрутку назад. Если параметр прокрутки не указан, возможность прокрутки назад зависит от запроса. Если указаны аргументы, они должны быть парами имени и типа данных, разделенными запятыми. Эти пары определяют имена, которые будут заменены значениями параметров в запросе. Фактические значения для замены этих имен будут предоставлены при открытии курсора.

Примеры:

```
DECLARE curs1 REFCURSOR;
curs2 CURSOR FOR SELECT * FROM tenk1;
curs3 CURSOR (key integer)
FOR SELECT * FROM tenk1 WHERE unique1 = key;
```

Все три переменные имеют тип данных REFCURSOR. Первая переменная может использоваться с любым запросом, вторая связана (bound) с полностью сформированным запросом, а последняя - с параметризованным запросом (key будет заменен на целочисленное значение параметра при открытии курсора). Переменная curs1 считается несвязанной (unbound), поскольку с ней не связано ни одного запроса. Реализация SCROLL полагается на использование FOR UPDATE/SHARE в запросе курсора. Кроме того, при использовании запроса, включающего волатильные (изменчивые) функции, рекомендуется использовать NO SCROLL. Реализация SCROLL предполагает, что повторное прочтение вывода запроса даст согласованные результаты, что не может быть гарантировано при использовании изменяемых функций.

Открытие курсора

Перед получением данных из курсора его необходимо открыть, аналогично SQL-команде DECLARE CURSOR.

В PL/pgSQL существует три формы оператора `OPEN`: две для несвязанных переменных курсора и одна для связанных переменных.

Пример функции, которая открывает курсор и извлекает строки на основе заданного года:

```
CREATE OR REPLACE FUNCTION get_film_titles(p_year integer)
RETURNS TEXT AS $$
DECLARE
 titles TEXT DEFAULT '';
 rec_film RECORD;
 cur_films CURSOR (p_year INTEGER) FOR
     SELECT title, release_year
     FROM film
     WHERE release_year = p_year;
BEGIN
 -- Open the cursor
 OPEN cur_films(p_year);

 LOOP
     -- Fetch row into the film
     FETCH cur_films INTO rec_film;
     -- Exit when no more rows to fetch
     EXIT WHEN NOT FOUND;

     -- Build the output
     IF rec_film.title LIKE '%ful%' THEN
```

```
            titles    :=    titles    ||    ','    ||
rec_film.title || ':' || rec_film.release_year;
        END IF;
    END LOOP;

    -- Close the cursor
    CLOSE cur_films;

    RETURN titles;
   END;
   $$ LANGUAGE plpgsql;

   SELECT get_film_titles(2006);
```

Пример открытия двух разных курсоров

```
   DECLARE
       cur_films CURSOR FOR SELECT * FROM film;
       cur_films2 CURSOR (year integer) FOR SELECT
* FROM film WHERE release_year = year;
   BEGIN
       OPEN cur_films;
       OPEN cur_films2(year:=2005);
   END;
```

С исключительной ситуацией

```
   DO
```

```
$$
DECLARE
 rec record;
 v_length int = 90;
BEGIN
 -- Select a film
 SELECT film_id, title
INTO STRICT rec
FROM film
WHERE length = v_length;

 -- Catch exceptions
 EXCEPTION
    WHEN sqlstate 'P0002' THEN
       RAISE EXCEPTION 'Film with length % not found', v_length;
    WHEN sqlstate 'P0003' THEN
       RAISE EXCEPTION 'The film with length % is not unique', v_length;
END;
$$
LANGUAGE plpgsql;
```

Оператор FETCH.

FETCH [направление { FROM | IN }] курсор INTO цель;

FETCH извлекает строки с помощью ранее заданного курсора в цель. В качестве цели может быть строковая переменная, переменная типа record или список простых переменных, разделенных запятой, как и в SELECT INTO. Курсор имеет ассоциированную позицию, которая используется командой FETCH. Позиция курсора может быть перед первой строкой результата запроса, на любой конкретной строке результата или после последней строки результата. При создании курсор позиционируется перед первой строкой. После выборки нескольких строк курсор позиционируется на последней полученной строке. Если FETCH заканчивается на конце доступных строк, то курсор остается позиционированным после последней строки или перед первой строкой, если выборка идет в обратном направлении. FETCH ALL или FETCH BACKWARD ALL всегда оставляют курсор расположенным после последней или перед первой строкой.

Параметры NEXT, PRIOR, FIRST, LAST, ABSOLUTE, RELATIVE выполняют выборку одного ряда после соответствующего перемещения курсора. Если такой строки нет, возвращается пустой результат, а курсор остается установленным перед первой строкой или после последней строки, в зависимости от ситуации.

Параметры, использующие FORWARD и BACKWARD, извлекают указанное количество строк, двигаясь в прямом или обратном направлении, оставляя курсор на последней возвращенной строке (или после/перед всеми строками, если счетчик превышает количество доступных строк). Если требуется перемещение назад, курсор должен быть объявлен или открыт с SCROLL.

RELATIVE 0, FORWARD 0 и BACKWARD 0 запрашивают выборку текущего ряда без перемещения курсора, то есть повторную выборку последнего найденного ряда. Это будет успешным, если курсор не

расположен перед первой строкой или после последней; в этом случае ни одна строка не будет возвращена.

Примеры:

```
FETCH curs1 INTO rowvar;
FETCH curs2 INTO foo, bar, baz;
FETCH LAST FROM curs3 INTO x, y;
FETCH RELATIVE -2 FROM curs4 INTO x;
```

Оператор MOVE

MOVE [направление { FROM | IN }] курсор;

MOVE перемещает курсор без извлечения данных. MOVE работает аналогично FETCH, но только перемещает курсор без извлечения строки. Успешность перемещения можно проверить с помощью специальной переменной FOUND.

Примеры:

```
MOVE curs1;
MOVE LAST FROM curs3;
MOVE RELATIVE -2 FROM curs4;
MOVE FORWARD 2 FROM curs4;
```

Команды UPDATE/DELETE

UPDATE таблица SET ... WHERE CURRENT OF курсор;

DELETE FROM таблица WHERE CURRENT OF курсор;

Когда курсор позиционируется на строку таблицы, эту строку можно изменить или удалить с помощью курсора. Существуют ограничения на запрос курсора (в частности, нельзя использовать группировки), и

рекомендуется указывать FOR UPDATE. Для получения дополнительной информации обратитесь к документации DECLARE.

Пример:

```
UPDATE foo SET dataval = myval WHERE CURRENT OF curs1;
```

Команда CLOSE

CLOSE курсор;

CLOSE закрывает связанный с курсором портал. Используется для освобождения ресурсов до завершения транзакции или для повторного открытия курсорной переменной.

Пример:

```
CLOSE curs1;
```

Пример курсора для таблиц учебной базы. Сделать выборку из таблицы SUBJ_LECT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о преподавателях, читающих предмет с данным номером с использованием пользовательской исключительной ситуации.

```
DO $$
    DECLARE
        v_lect_id subj_lect.lect_id%TYPE;
        v_subj_id subj_lect.subj_id%TYPE;
        v_subject_found boolean := FALSE;
        v_message text;
        c_subj_lect CURSOR FOR SELECT lect_id, subj_id
                               FROM subj_lect;
```

```
    BEGIN
        OPEN c_subj_lect;
    LOOP
        FETCH c_subj_lect INTO v_lect_id, v_subj_id
        EXIT WHEN NOT FOUND;
        RAISE NOTICE 'Lecturer ID: %, Subject ID: %',
                     v_lect_id, v_subj_id;
        IF v_subj_id = 10 THEN
            v_subject_found := TRUE;
            EXIT;
        END IF;
    END LOOP;
    CLOSE c_subj_lect;
    IF v_subject_found THEN
        v_message := 'Found';
    ELSE
        v_message := 'Not Found';
    END IF;
    RAISE NOTICE '%', v_message;
END $$;
```

Задания.

1. Сделать выборку из таблицы STUDENT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о студентах, живущих в Москве с использованием пользовательской исключительной ситуации.
2. Сделать выборку из таблицы SUBJECT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о предметах во 2-м семестре с использованием пользовательской

исключительной ситуации.

3. Сделать выборку из таблицы UNIVERSITY с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных об университетах с рейтингом большим 200 с использованием пользовательской исключительной ситуации.
4. Сделать выборку из таблицы EXAM_MARKS с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных об экзаменах, сданных в данную дату с использованием пользовательской исключительной ситуации.
5. Сделать выборку из таблицы SUBJ_LECT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о преподавателях, читающих предмет с данным номером с использованием пользовательской исключительной ситуации.
6. Сделать выборку из таблицы STUDENT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о студентах, живущих в Москве и использовать пользовательскую исключительную ситуацию среди выбранных студентов тех, у которых стипендия меньше повышенной.
7. Сделать выборку из таблицы SUBJECT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о предметах во 2-м семестре с использованием пользовательской исключительной ситуации с количеством часов 72.
8. Сделать выборку из таблицы UNIVERSITY с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных об университетах с рейтингом большим 400 и с помощью пользовательской исключительной ситуации исключить из вывода данные об университетах, расположенных в Воронеже.
9. Сделать выборку из таблицы EXAM_MARKS с использованием курсора

и цикла с методом %FOUND или %NOTFOUND для получения данных об экзаменах, сданных в данную дату с помощью пользовательской исключительной ситуации.

10. Сделать выборку из таблицы SUBJ_LECT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о преподавателях, читающих предмет с данным номером с помощью пользовательской исключительной ситуации.
11. Сделать выборку из таблицы STUDENT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о студентах, получающих повышенную стипендию с использованием пользовательской исключительной ситуации.
12. Сделать выборку из таблицы EXAM_MARKS с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных об экзаменах, сданных на первых двух курсах с помощью пользовательской исключительной ситуации.
13. Сделать выборку из таблицы UNIVERSITY с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных об университетах, расположенных в Москве и с помощью пользовательской исключительной ситуации исключить из вывода данные о МГУ.
14. Сделать выборку из таблицы SUBJECT с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных о предметах на 4-м курсе с использованием пользовательской исключительной ситуации с количеством часов 36.
15. Сделать выборку из таблицы UNIVERSITY с использованием курсора и цикла с методом %FOUND или %NOTFOUND для получения данных об университетах с максимальным рейтингом и с помощью пользовательской исключительной ситуации исключить из вывода

данные об университетах, расположенных в Воронеже.

4.5. Триггеры

Триггер в PostgreSQL объявляется как функция без аргументов и с типом результата `trigger`. Важно отметить, что эта функция должна быть объявлена без аргументов, даже если предполагается, что она получит аргументы, указанные в команде `CREATE TRIGGER` — такие аргументы передаются через `TG_ARGV`, как описано ниже.

Когда функция PL/pgSQL запускается как триггер, в блоке верхнего уровня автоматически создается несколько специальных переменных:

- `NEW` (тип данных `RECORD`): Эта переменная содержит данные новой строки для команд `INSERT/UPDATE` в триггерах на уровне строк. В триггерах уровня оператора и для команды `DELETE` эта переменная имеет значение `NULL`.
- `OLD` (тип данных `RECORD`): Эта переменная содержит данные старой строки для команд `UPDATE/DELETE` в триггерах уровня строки. В триггерах уровня оператора и для команды `INSERT` эта переменная имеет значение `NULL`.
- `TG_NAME` (тип данных `name`): Переменная содержит имя сработавшего триггера.
- `TG_WHEN` (тип данных `text`): Строка, содержащая `BEFORE`, `AFTER` или `INSTEAD OF`, в зависимости от определения триггера.
- `TG_LEVEL` (тип данных `text`): Строка, содержащая `ROW` или `STATEMENT`, в зависимости от определения триггера.
- `TG_OP` (тип данных `text`): Строка, содержащая `INSERT`, `UPDATE`, `DELETE` или `TRUNCATE`, в зависимости от того, для какой операции сработал триггер.

- TG_RELID (тип данных oid): OID таблицы, для которой сработал триггер.
- TG_RELNAME (тип данных name): Имя таблицы, для которой сработал триггер. Эта переменная устарела и может стать недоступной в будущих релизах. Вместо неё нужно использовать TG_TABLE_NAME.
- TG_TABLE_NAME (тип данных name): Имя таблицы, для которой сработал триггер.
- TG_TABLE_SCHEMA (тип данных name): Имя схемы, содержащей таблицу, для которой сработал триггер.
- TG_NARGS (тип данных integer): Число аргументов в команде CREATE TRIGGER, которые передаются в триггерную функцию.
- TG_ARGV[] (тип данных массив text): Аргументы из команды CREATE TRIGGER. Индекс массива начинается с 0. Для недопустимых значений индекса (< 0 или >= tg_nargs) возвращается NULL.

Триггерная функция должна вернуть либо NULL, либо запись/строку, соответствующую структуре таблицы, для которой сработал триггер. Если триггер уровня строки BEFORE возвращает NULL, то все дальнейшие действия с этой строкой прекращаются (т.е. последующие триггеры не запускаются, и команда INSERT/UPDATE/DELETE не выполняется над строкой). Если возвращаемое значение не NULL, то дальнейшая обработка продолжается именно с этой строкой. Возвращение строки отличной от начальной NEW, изменяет строку, которая будет вставлена или изменена. Поэтому, если в триггерной функции нужно выполнить некоторые действия и не менять саму строку, то нужно возвратить переменную NEW (или её эквивалент). Для того чтобы изменить

сохраняемую строку, можно поменять отдельные значения в переменной NEW и затем её вернуть. Либо создать и вернуть полностью новую переменную.

В случае строчного триггера BEFORE для команды DELETE само возвращаемое значение не имеет прямого эффекта, но оно должно быть отличным от NULL, чтобы не прерывать обработку строки. Обратите внимание, что переменная NEW всегда NULL в триггерах на DELETE, поэтому возвращать её не имеет смысла. Традиционной идиомой для триггеров DELETE является возврат переменной OLD.

Триггеры INSTEAD OF (которые всегда являются триггерами уровня строки и могут использоваться только с представлениями) могут возвращать NULL, чтобы показать, что они не выполняли никаких изменений, так что обработку этой строки можно не продолжать (то есть, не вызывать последующие триггеры и не считать строку в числе обработанных строк для окружающих команд INSERT/UPDATE/DELETE). В противном случае должно быть возвращено значение, отличное от NULL, показывающее, что триггер выполнил запрошенную операцию.

Для операций INSERT и UPDATE возвращаемым значением должно быть NEW, которое триггерная функция может модифицировать для поддержки предложений INSERT RETURNING и UPDATE RETURNING (это также повлияет на значение строки, передаваемое последующим триггерам, или доступное под специальным псевдонимом EXCLUDED в операторе INSERT с предложением ON CONFLICT DO UPDATE).

Для операций DELETE возвращаемым значением должно быть OLD. Возвращаемое значение для строчного триггера AFTER и триггеров уровня оператора (BEFORE или AFTER) всегда игнорируется. Это может быть и NULL. Однако в этих триггерах по-прежнему можно прервать вызвавшую

их команду, для этого нужно явно вызвать ошибку. Пример показывает пример триггерной функции в PL/pgSQL.

Пример: Триггерная функция на `PL/pgSQL`.

Триггер, показанный в этом примере, при любом добавлении или изменении строки в таблице сохраняет в этой строке информацию о текущем пользователе и отметку времени. Кроме того, он требует, чтобы было указано имя сотрудника, и зарплата задавалась положительным числом.

```
CREATE TABLE emp (
      empname text,
      salary integer,
      last_date timestamp,
      last_user text
);

CREATE FUNCTION emp_stamp()
RETURNS trigger AS $emp_stamp$
BEGIN
-- Проверить, что указаны имя сотрудника и зарплата
      IF NEW.empname IS NULL THEN EXCEPTION
      'empname cannot be null';
      END IF;
      IF NEW.salary IS NULL THEN RAISE EXCEPTION
      '% cannot have null salary', NEW.empname;
      END IF;
-- Кто будет работать, если за это надо будет платить?
      IF NEW.salary < 0 THEN RAISE EXCEPTION
      '%cannot have a negative salary', NEW.empname;
```

```
    END IF;
-- Запомнить, кто и когда изменил запись
NEW.last_date := current_timestamp;
NEW.last_user := current_user;
RETURN NEW;
END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp
BEFORE INSERT OR UPDATE ON emp
FOR EACH ROW EXECUTE FUNCTION emp_stamp();
```

Другой вариант ведения журнала изменений для таблицы предполагает создание новой таблицы, которая будет содержать отдельную запись для каждой выполненной команды `INSERT`, `UPDATE`, `DELETE`. Этот подход можно рассматривать как протоколирование изменений таблицы для аудита.

Пример: Триггерная функция для аудита в `PL/pgSQL`

Показанный в этом примере триггер гарантирует, что любое добавление, изменение или удаление строки в таблице `emp` будет зафиксировано в таблице `emp_audit` (для аудита). Также он фиксирует текущее время, имя пользователя и тип выполняемой операции.

```
CREATE TABLE emp(
    empname text NOT NULL,
    salary integer
);
CREATE TABLE emp_audit(
    operation char(1) NOT NULL,
```

```
      stamp timestamp NOT NULL,
      userid text NOT NULL,
      empname text NOT NULL,
      salary integer
);
CREATE OR REPLACE FUNCTION process_emp_audit()
RETURNS TRIGGER AS $emp_audit$
BEGIN
      -- Добавление строки в emp_audit,
      -- которая отражает операцию, выполняемую в emp;
      -- для определения типа операции применяется
      -- специальная переменная TG_OP.
      IF (TG_OP = 'DELETE') THEN
            INSERT INTO emp_audit
            SELECT 'D', now(), user, OLD.*;
      ELSIF (TG_OP = 'UPDATE') THEN
            INSERT INTO emp_audit
            SELECT 'U', now(), user, NEW.*;
      ELSIF (TG_OP = 'INSERT') THEN
            INSERT INTO emp_audit
            SELECT 'I', now(), user, NEW.*;
      END IF;

      RETURN NULL; -- возвращаемое значение для
      -- триггера AFTER игнорируется
END;
$emp_audit$ LANGUAGE plpgsql;

CREATE TRIGGER emp_audit
```

```
AFTER INSERT OR UPDATE OR DELETE ON emp
FOR EACH ROW
EXECUTE FUNCTION process_emp_audit();
```

Пример триггера для учебной базы данных. Создать триггер, который считает средний рейтинг университетов и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута в зависимости от величины среднего рейтинга, при этом происходит заполнение некоторой таблицы.

Создаем функцию, которая вычисляет средний рейтинг университетов и проверяет превышение порога отклонения:

```
CREATE OR REPLACE FUNCTION check_average_rating()
  RETURNS TRIGGER AS
$$
DECLARE
  avg_rating FLOAT;
  deviation_threshold FLOAT := 1.5;
  -- Значение порога отклонения
BEGIN
  -- Вычисляем средний рейтинг университетов
  SELECT AVG(rating) INTO avg_rating FROM university;
  -- Проверяем, превышает ли отклонение
  -- пороговое значение
  IF abs(NEW.rating-avg_rating) > deviation_threshold
  THEN
    -- Возбуждаем исключение
    -- с диагностическим сообщением
```

```
    RAISE EXCEPTION 'Превышено пороговое значение
отклонения: Средний рейтинг составляет %, введенный
рейтинг %',
      avg_rating, NEW.rating;
  END IF;

  -- Возвращаем новую строку
  RETURN NEW;
END;
$$
LANGUAGE plpgsql;

-- Создаем триггер, который вызывает функцию при
-- вставке или обновлении строки в таблице university
CREATE TRIGGER check_rating_trigger
  BEFORE INSERT OR UPDATE ON university
  FOR EACH ROW
  EXECUTE FUNCTION check_average_rating();

-- Выполнение
-- Вставим несколько записей в таблицу university
INSERT INTO university
  (univ_id, univ_name, rating, city)
VALUES
  (1, 'Университет 1', 4.5, 'Город 1'),
  (2, 'Университет 2', 3.7, 'Город 2'),
  (3, 'Университет 3', 4.2, 'Город 3');

-- Обновим рейтинг университета 2
UPDATE university SET rating = 4.8 WHERE univ_id = 2;
```

```
-- Вставим новый университет, который
-- не превышает пороговое значение отклонения
INSERT INTO university
  (univ_id, univ_name, rating, city)
  VALUES (4, 'Университет 4', 4, 'Город 4');
```

Задания.

1. Создать триггер, который считает среднюю стипендию и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута в зависимости от средней стипендии, при этом происходит заполнение некоторой таблицы.
2. Создать триггер, который считает средний балл в заданный день и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута в зависимости от среднего балла, при этом происходит заполнение некоторой таблицы.
3. Создать триггер, который определяет границы изменения номеров предметов и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута, при этом происходит заполнение некоторой таблицы.
4. Создать триггер, который определяет границы изменения номеров преподавателей и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута, при этом происходит заполнение некоторой таблицы.
5. Создать триггер, который считает средний рейтинг университетов и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута в зависимости от величины среднего рейтинга, при этом происходит заполнение некоторой таблицы.

6. Создать триггер, который определяет границы изменения номеров лекторов в зависимости от номеров читаемых курсов и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута, при этом происходит заполнение некоторой таблицы.
7. Создать триггер, который считает максимальный стипендию и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута от максимальной стипендии, при этом происходит заполнение некоторой таблицы.
8. Создать триггер, который считает средний балл заданного студента и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута в зависимости от среднего балла, при этом происходит заполнение некоторой таблицы.
9. Создать триггер, который определяет границы изменения номеров студентов и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута, при этом происходит заполнение таблицы.
10. Создать триггер, который определяет границы изменения номеров предметов и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута, при этом происходит заполнение некоторой таблицы.
11. Создать триггер, который считает минимальный рейтинг университетов и выдает диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута в зависимости от величины минимального рейтинга, при этом происходит заполнение некоторой таблицы.
12. Создать триггер, который определяет границы изменения номеров лекторов в зависимости от рейтинга университета и выдает

диагностическое сообщение при превышении заданного порога уклонения вводимого значения атрибута номера лектора, при этом происходит заполнение некоторой таблицы.

4.6. Последовательности

```
-- Создание таблицы EXAM_MARKS
CREATE TABLE EXAM_MARKS (
  exam_id INT PRIMARY KEY,
  stud_id INT,
  subj_id INT,
  mark INT,
  univ_id INT,
 FOREIGN KEY (stud_id) REFERENCES student(stud_id),
  FOREIGN KEY (subj_id) REFERENCES subject(subj_id),
  FOREIGN KEY (univ_id) REFERENCES university(univ_id)
);

-- Создание последовательности для генерации значений
столбца exam_id
CREATE SEQUENCE exam_id_sequence START WITH 1
INCREMENT BY 1;
-- Заполнение таблицы EXAM_MARKS с использованием
последовательности
INSERT INTO EXAM_MARKS (exam_id, stud_id, subj_id,
mark, univ_id)
VALUES (NEXTVAL('exam_id_sequence'), 10, 10, 85, 16);

-- Выполнение:
SELECT * FROM EXAM_MARKS;
```

```
GRANT SELECT ON university TO lab;

-- Переключаемся на пользователя lab
SELECT univ_id FROM university;
```

Задания.

1. Создать таблицу STUDENT и заполнить ее первичный ключ с использованием последовательностей.
2. Создать таблицу SUBJECT и заполнить ее первичный ключ с использованием последовательностей.
3. Создать таблицу LECTURER и заполнить ее первичный ключ с использованием последовательностей.
4. Создать таблицу UNIVERSITY и заполнить ее первичный ключ с использованием последовательностей.
5. Создать таблицу EXAM_MARKS и заполнить ее первичный ключ с использованием последовательностей.
6. Создать таблицу SUBJ_LECT и заполнить ее первичный ключ с использованием последовательностей.

4.7. Пакеты

`CREATE PACKAGE` — создать пакет

`CREATE [ OR REPLACE ] PACKAGE` *`имя_пакета`*
`элемент_пакета` `[ ... ]`

`CREATE PACKAGE` создаёт пакет в текущей базе данных.

Пакет — это, по сути, схема, которая помогает организовать взаимосвязанные именованные объекты, поэтому его также можно создать командой `CREATE SCHEMA`, и с ним можно выполнять те же действия, что и с обычной схемой. Однако пакет может содержать только функции, процедуры и составные типы.

CREATE OR REPLACE PACKAGE можно использовать как для создания нового пакета, так и для замены существующего определения пакета. При замене пакета можно изменять только функции и типы, в то время как другие объекты необходимо предварительно удалить. Если сигнатуры функций остаются неизменными, содержимое пакета будет заменено, а зависимые объекты будут сохранены. Однако если сигнатура функции изменится, будет создана новая функция. Создание новой функции будет успешным только в том случае, если нет зависимых объектов. Важно убедиться, что никакие другие сеансы не используют заменяемые пакеты, остановив соответствующее приложение.

То же ограничение действует при замене типов. При замене пакетов убедитесь, что никакие другие сеансы не используют их (другими словами, перед заменой пакетов остановите использующее их приложение).

Параметры пакета:

- имя_пакета: Имя создаваемого пакета. Имя пакета должно быть уникальным и отличаться от любого существующего имени пакета или схемы в текущей базе данных. Обратите внимание, что имена, начинающиеся с pg_, зарезервированы для системных схем.
- элемент_пакета: Оператор SQL, определяющий объект, который будет создан в пакете. В настоящее время в CREATE PACKAGE разрешены только операторы CREATE FUNCTION, CREATE TYPE и CREATE PROCEDURE. Подкоманды обрабатываются аналогично отдельным командам, выполняемым после создания пакета. Переменные, объявленные в функции инициализации пакета, являются глобальными и могут быть доступны функциям других пакетов используя запись с точкой. Модификатор #import, указанный для функции инициализации, влияет на все функции пакета.

Чтобы создать пакет, пользователь должен быть суперпользователем или иметь разрешение CREATE для текущей базы данных.

Пример создания пакета counter с функциями:

```
CREATE PACKAGE counter
    CREATE FUNCTION __init__() RETURNS void AS $$
     -- package initialization
    DECLARE
        n int := 1;
        k int := 3;
    BEGIN
        FOR i IN 1..10 LOOP
            n := n + n;
        END LOOP;
    END;
    $$

    CREATE FUNCTION inc() RETURNS int AS $$
    BEGIN
        n := n + 1;
        RETURN n;
    END;
    $$
;
```

Обратите внимание, что отдельные подкоманды внутри пакета не заканчиваются точкой с запятой, подобно оператору CREATE SCHEMA. Кроме того, при создании функций не указывается язык.

Следующий пример показывает, как можно использовать описанный выше пакет.

```
DO $$
#import counter
BEGIN
    RAISE NOTICE '%', counter.n;
    RAISE NOTICE '%', counter.inc();
END;
$$;

-- Вывод:
NOTICE:  1024
NOTICE:  1025
```

Другой вариант предыдущего примера:

```
CREATE PACKAGE foo
  CREATE TYPE footype AS (a int, b int)

  CREATE FUNCTION __init__() RETURNS void AS $$
  DECLARE
    x int := 1;
  BEGIN
    RAISE NOTICE 'foo initialized';
  END;
  $$

  CREATE FUNCTION get() RETURNS int AS $$
  BEGIN
    RETURN x;
  END;
  $$
```

```
  CREATE FUNCTION inc() RETURNS void AS $$
  BEGIN
    x := x + 1;
  ENG;
  $$
;
```

Ниже показано, как можно добиться того же результата командой CREATE SCHEMA:

```
CREATE SCHEMA foo;
CREATE TYPE foo.footype AS (a int, b int);
CREATE FUNCTION foo.__init__() RETURNS void AS $$
DECLARE x int := 1;
BEGIN
  RAISE NOTICE 'foo initialized';
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION foo.get() RETURNS int AS $$
#package
BEGIN
  RETURN x;
END;
$$ LANGUAGE plpgsql;

CREATE FUNCTION foo.inc() RETURNS void AS $$
#package
BEGIN
```

```
  x := x + 1;
END;
$$ LANGUAGE plpgsql;
```

Пример пакета. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество преподавателей, работающих в университете, заданным параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, номер университета, количество преподавателей в новую таблицу, созданную заранее.

```
CREATE SCHEMA lecturer_package;
CREATE SCHEMA lecturer_package;
CREATE OR REPLACE FUNCTION
  teacher_package.count_teachers_by_university
  (p_univ_id university.univ_id%TYPE)
  RETURNS INTEGER AS $$
DECLARE
  v_teacher_count INTEGER;
BEGIN
  SELECT COUNT(*) INTO v_teacher_count FROM lecturer
  WHERE univ_id = p_univ_id;
  RETURN v_teacher_count;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION
  teacher_package.insert_call_count
  (p_univ_id university.univ_id%TYPE,
```

```
   p_teacher_count INTEGER)
RETURNS VOID AS $$
BEGIN
  INSERT INTO teacher_package.call_count_table
    (univ_id, call_count, teacher_count)
  SELECT p_univ_id, COALESCE(MAX(call_count), 0) + 1,
         p_teacher_count
  FROM teacher_package.call_count_table
  WHERE univ_id = p_univ_id;
  RAISE NOTICE 'Вызов внесения значения успешен.';
END;
$$ LANGUAGE plpgsql;

CREATE TABLE teacher_package.call_count_table (
  univ_id INTEGER,
  call_count INTEGER,
  lecturer_count INTEGER
);

-- Выполнение
SELECT teacher_package.count_teachers_by_university(1)
CALL teacher_package.insert_call_count(1, 10)
```

Задания.

1. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество студентов, получающих стипендию, заданную параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, величину стипендии, количество студентов, получающих в ее в новую таблицу, созданную

заранее.

2. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество студентов, живущих в городе, заданном параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, название города, количество студентов, живущих в этом городе в новую таблицу, созданную заранее.
3. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество предметов, по которым получено оценка, заданная параметром, более чем у 20 человек. Процедура подсчитывает количество обращений к функции и заносит это количество, величину оценки и количество предметов в новую таблицу, созданную заранее.
4. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество университетов с рейтингом, заданным параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, величину рейтинга, количество университетов с этим рейтингом в новую таблицу, созданную заранее.
5. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество преподавателей, работающих в университете, заданным параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, номер университета, количество преподавателей в новую таблицу, созданную заранее.
6. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество предметов, прочитанных в семестре, номер которого задан параметром, с количеством часов,

заданным другим параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, номер семестра, количество часов и количество предметов в новую таблицу, созданную заранее.

7. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество предметов, прочитанных в семестре, номер которого задан параметром и средний балл, полученный по этому предмету. Процедура подсчитывает количество обращений к функции и заносит это количество, номер семестра, средний балл и количество предметов в новую таблицу, созданную заранее.
8. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество студентов, учащихся в университете, заданным параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, номер университета, количество студентов в новую таблицу, созданную заранее.
9. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество университетов с рейтингом, заданным параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, величину рейтинга, количество университетов с рейтингом меньше, чем заданный, в новую таблицу, созданную заранее.
10. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество предметов, по которым получено оценка, заданная параметром, более чем у 20 человек. Процедура подсчитывает количество обращений к функции и заносит это количество, величину оценки и количество предметов в новую

таблицу, созданную заранее.

11. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество студентов заданного года рождения в городе, заданном параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, название города, количество студентов заданного года рождения, в новую таблицу, созданную заранее.
12. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество студентов, получивших оценки выше заданной параметром. Процедура подсчитывает количество обращений к функции и заносит эту оценку, количество студентов, получающих ее, в новую таблицу, созданную заранее.
13. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество предметов, по которым получено оценка, выше средней по этому предмету, более чем у 10 человек. Процедура подсчитывает количество обращений к функции и заносит это количество, величину оценки и количество предметов в новую таблицу, созданную заранее.
14. Создать пакет, состоящий из функции с параметрами, процедуры без параметров. Функция подсчитывает количество студентов, получивших положительную оценку по предмету, заданном параметром. Процедура подсчитывает количество обращений к функции и заносит это количество, название предмета, количество студентов, получивших положительную оценку по предмету, в новую таблицу, созданную заранее.

4.8. Портирование из Oracle PL/SQL

В этом разделе рассматриваются различия между языками Postgres Pro PL/pgSQL и Oracle PL/SQL, чтобы помочь разработчикам, переносящим приложения из Oracle в Postgres Pro.

PL/pgSQL – это блочно-структурированный, императивный язык, во многом похожий на PL/SQL. Оба языка требуют объявления переменных, поддерживают присваивания, циклы и условные операторы. Однако при переносе с PL/SQL на PL/pgSQL необходимо учитывать несколько ключевых различий:

- В PL/pgSQL имя, используемое в команде SQL, может относиться либо к столбцу таблицы, либо к переменной функции. Однако PL/SQL считает его именем столбца таблицы, тогда как PL/pgSQL по умолчанию сообщает об ошибке из-за этой неоднозначности. Установив plpgsql.variable_conflict = use_column, вы можете изменить поведение в соответствии с PL/SQL. Обычно рекомендуется избегать таких неоднозначностей, но, если вам нужно перенести большой объем кода, который зависит от такого поведения, настройка variable_conflict может быть лучшим решением.
- В Postgres Pro тело функции должно быть записано в виде строки. Поэтому нужно использовать знак доллара в качестве кавычек или экранировать одиночные кавычки в теле функции.
- Имена типов данных часто требуют корректировки при переносе. Например, в Oracle строковые значения часто объявляются с типом varchar2, который не является стандартным типом SQL. В Postgres Pro вместо него рекомендуется использовать varchar или text. Аналогично, тип number следует заменить на numeric или другой подходящий числовой тип.

- В Postgres Pro для группировки функций используются схемы вместо пакетов.
- Поскольку в Postgres Pro нет пакетов, пакетные переменные не существуют. Вместо этого вы можете хранить состояние каждого сеанса во временных таблицах.
- Целочисленные циклы FOR с указанием REVERSE задают разное поведение. В PL/SQL значение счётчика уменьшается от второго числа к первому, в то время как в PL/pgSQL счётчик уменьшается от первого ко второму. Поэтому при портировании нужно менять местами границы цикла. Хотя такое решение вызывает сожаление, оно вряд ли будет изменено
- Циклы FOR в запросах (не курсорам) также работают по-разному. В PL/pgSQL переменная цикла должна быть объявлена явно, тогда как в PL/SQL она объявляется неявно. Преимущество PL/pgSQL в том, что значения переменных остаются доступными после выхода из цикла.
- Существуют некоторые отличия в нотации при использовании курсорных переменных.

Примеры портирования.

Пример: Портирование простой функции из PL/SQL в PL/pgSQL.

Функция Oracle PL/SQL:

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version (
      v_name varchar2,
      v_version varchar2
)
RETURN varchar2 IS
BEGIN
      IF v_version IS NULL THEN
```

```
            RETURN v_name;
      END IF;
RETURN v_name || '/' || v_version;
END;
/
show errors;
```

Сравним эту функцию и посмотрим различия по сравнению с PL/pgSQL. Тип данных varchar2 в Oracle следует сменить на varchar или text для PL/pgSQL. В этом разделе для примеров мы будем использовать varchar, но в целом лучше выбрать text, если не требуется ограничивать длину строк.

Ключевое слово RETURN в прототипе функции (не в теле функции) в Postgres Pro заменяется на RETURNS. Кроме того, IS становится AS, и необходимо добавить оператор LANGUAGE, потому что PL/pgSQL – не единственный возможный язык.

В Postgres Pro тело функции является строкой, поэтому нужно использовать кавычки или знаки доллара. Это заменяет завершающий / в подходе Oracle.

Команда show errors не существует в Postgres Pro и не требуется, так как ошибки будут выводиться автоматически.

Функция после портирования на PL/pgSQL:

```
CREATE OR REPLACE FUNCTION cs_fmt_browser_version (
      v_name varchar,
      v_version varchar
)
RETURNS varchar AS $$
      BEGIN
```

```
            IF v_version IS NULL THEN
                 RETURN v_name;
            END IF;
RETURN v_name || '/' || v_version;
END;
$$ LANGUAGE plpgsql;
```

Пример. Портирование функции, создающей другую функцию, из PL/SQL в PL/pgSQL, обработка проблем с кавычками.

Следующая процедура получает строки из `SELECT` и строит большую функцию, в целях эффективности возвращающую результат в операторах `IF`.

Версия `Oracle`:

```
CREATE OR REPLACE PROCEDURE
cs_update_referrer_type_proc IS
  CURSOR referrer_keys IS
  SELECT * FROM cs_referrer_keys ORDER BY try_order;
  func_cmd VARCHAR(4000);
BEGIN
  func_cmd := 'CREATE OR REPLACE FUNCTION
    cs_find_referrer_type(
    v_host IN VARCHAR2,
    v_domain IN VARCHAR2,
    v_url IN VARCHAR2)
    RETURN VARCHAR2 IS BEGIN';
  FOR referrer_key IN referrer_keys LOOP
    func_cmd := func_cmd || ' IF v_' ||
      referrer_key.kind || ' LIKE ''' ||
      referrer_key.key_string || ''' THEN RETURN '''
```

```
      || referrer_key.referrer_type || '''; END IF;';
  END LOOP;
  func_cmd := func_cmd || ' RETURN NULL; END;';
  EXECUTE IMMEDIATE func_cmd;
END;
/
show errors;
```

В Postgres Pro эта функция может выглядеть так:

```
CREATE OR REPLACE PROCEDURE
cs_update_referrer_type_proc() AS $func$
DECLARE
  referrer_keys CURSOR IS SELECT *
    FROM cs_referrer_keys ORDER BY try_order;
  func_body text;
  func_cmd text;
BEGIN
  func_body := 'BEGIN';
  FOR referrer_key IN referrer_keys LOOP
    func_body := func_body || 'IF v_' ||
      referrer_key.kind || ' LIKE ' ||
      quote_literal(referrer_key.key_string) ||
      ' THEN RETURN ' ||
      quote_literal(referrer_key.referrer_type) ||
      '; END IF;';
  END LOOP;
  func_body := func_body || ' RETURN NULL; END;';
  func_cmd := 'CREATE OR REPLACE FUNCTION
    cs_find_referrer_type(v_host varchar, v_domain
```

```
      varchar, v_url varchar) RETURNS varchar AS
      ' || quote_literal(func_body) ||
      ' LANGUAGE plpgsql;';
   EXECUTE func_cmd;
END;
$func$ LANGUAGE plpgsql;
```

Обратите внимание, что тело функции строится отдельно, с использованием `quote_literal` для дублирования кавычек. Эта техника необходима, потому что мы не можем безопасно использовать знаки доллара при определении новой функции: мы не знаем наверняка, какие строки будут вставлены из `referrer_key.key_string`. (Мы предполагаем, что `referrer_key.kind` всегда имеет значение из списка: `host`, `domain` или `url`, но `referrer_key.key_string` может быть чем угодно, в частности, может содержать знаки доллара). На самом деле, в версии функции для `PL/pgSQL` есть улучшение по сравнению с оригиналом `Oracle`, потому что не будет генерироваться неправильный код, когда `referrer_key.key_string` или `referrer_key.referrer_type` содержат кавычки.

Этот пример демонстрирует, как перенести функцию с выходными параметрами `(OUT)` и манипулирующую строками.

В `Postgres Pro` нет встроенной функции `instr`, но её можно создать, используя комбинацию других функций.

5. Расширенные возможности группировки в СУБД PostgreSQL

5.1. База данных

Во всех примерах этого раздела используется демонстрационная БД «Авиаперевозки» (https://postgrespro.ru/education/demodb), содержащая следующие таблицы:

- Бронирования (Bookings);
- Билеты (Tickets);
- Перелеты (Ticket_flights);
- Рейсы (Flights);
- Посадочные талоны (Boarding_passes);
- Аэропорты (Airports);
- Самолеты (Aircrafts);
- Места (Seats).

5.2. Примеры простых SELECT-запросов

Первый тип запросов – **запросы без группировки и без агрегации**.

Задание: выдать информацию обо всех бронированиях, сделанных на сумму более 500000 рублей.

```
SELECT book_ref, book_date, total_amount
FROM Bookings
WHERE total_amount > 500000;
```

Никакой группировки и агрегации данных нет: каждая строка ответа содержит данные только из одной строки таблицы-источника.

Задание: выдать информацию обо всех бронированиях, сделанных на сумму более 500000 рублей, включая информацию о включенных в них билетах.

```
SELECT B.book_ref, B.book_date, B.total_amount,
       T.ticket_no, T.passenger_id, T.passenger_name,
       T.contact_data
FROM Bookings B JOIN Tickets T
     ON B.book_ref = T.book_ref
WHERE total_amount > 500000;
```

В случае нескольких таблиц-источников каждая строка ответа содержит данные только из одной строки декартова произведения таблиц-источников.

Второй тип запросов – **запросы с группировкой**.

Задание: выдать список бронирований, сделанных на сумму не менее 500000 рублей и включающих не менее 4 билетов, указав для каждого из них количество включенных билетов.

```
SELECT B.book_ref, B.book_date, B.total_amount,
       COUNT(T.ticket_no) as ticket_cnt
FROM Bookings B JOIN Tickets T
     ON B.book_ref = T.book_ref
WHERE B.total_amount > 500000
GROUP BY B.book_ref HAVING COUNT(T.ticket_no) >= 4;
```

Строки, полученные простым запросом (он выделен жирным шрифтом), разбиваются на группы по указанному критерию. Каждая строка ответа соответствует одной такой группе. Конструкция HAVING позволяет накладывать условия на эти группы.

В ответ могут непосредственно попадать только те столбцы исходной таблицы, по которым делается группировка. В случае, если группировка делается только по первичному ключу некоторой таблицы (как в данном случае), в ответ также могут непосредственно попадать и другие столбцы

этой таблицы. Остальные столбцы могут попадать в ответ только как аргументы агрегатных функций.

Третий тип запросов – **запросы с агрегацией без группировки**.

Задание: выдать общее количество бронирований, каждое из которых сделано на сумму не менее 500000 рублей, и общую сумму этих бронирований.

```
SELECT COUNT(book_ref), SUM(total_amount)
FROM Bookings
WHERE total_amount > 500000;
```

Здесь вся выборка, полученная простым запросом (который выделен жирным шрифтом), представляет собой одну группу, по которой делается агрегация. Запрос с агрегацией без группировки всегда выдает одну строку.

Ограничения запросов простых типов:

- Агрегация для всех столбцов и строк ответа идет по одному и тому же множеству признаков.
- В одной и той же строке ответа все данные должны быть либо общими для группы, либо полученными в результате агрегации.
- Эти ограничения можно обойти, используя в качестве источников не таблицы, а подзапросы с группировкой. Но такой подход ведет к усложнению запросов.
- Никак нельзя обойти только невозможность рекурсии на произвольное число уровней (т.к. в этом случае нужна отдельная таблица-источник для каждого уровня).

5.3. SELECT-запросы с группировкой по разным критериям

Задание: выдать число рейсов – общее, для каждого города, в котором расположен аэропорт вылета, и для каждого аэропорта вылета. Без специального синтаксиса решение получается примерно таким:

```
SELECT P.airport_name, P.city, COUNT(*)
FROM Flights F JOIN Airports P
     ON F.departure_airport = P.airport_code
GROUP BY P.airport_code, P.airport_name, P.City
UNION ALL
SELECT NULL, P.City, COUNT(*)
FROM Flights F JOIN Airports P
     ON F.departure_airport = P.airport_code
GROUP BY P.city
UNION ALL
SELECT NULL, NULL, COUNT(*)
FROM Flights F JOIN Airports P
     ON F.departure_airport = P.airport_code;
```

Налицо объединение однотипных запросов, с одним и тем же источником и одними и теми же столбцами результата. NULL-значения везде указывают отсутствие группировки по соответствующему критерию (агрегацию по всем значениям этого критерия). Напрашивается упрощение синтаксиса.

```
SELECT P.airport_name, P.city, COUNT(*)
FROM Flights F JOIN Airports P
     ON F.departure_airport = P.airport_code
GROUP BY GROUPING SETS ((P.airport_name, P.City),
         (P.City), ())
ORDER BY city, airport_name;
```

Синтаксис с использованием GROUPING SETS гораздо проще, чем с UNION ALL, но есть и еще более простые варианты для стандартных комбинаций критериев. Синтаксис ROLLUP применяется для группировки по всем уровням вложенных критериев.

Задание то же: выдать число рейсов – общее, для каждого города, в котором расположен аэропорт вылета, и для каждого аэропорта вылета.

```
SELECT P.airport_name, P.city, COUNT(*)
FROM Flights F JOIN Airports P
     ON F.departure_airport = P.airport_code
GROUP BY ROLLUP ((P.City), (P.airport_name));
```

ROLLUP (A, B, C) – это то же самое, что и GROUPING SETS ((A, B, C), (A, B), (A), ()). ROLLUP ((A, B), (C, D)) – то же самое, что и GROUPING SETS ((A, B, C, D), (A, B), ()).

Синтаксис CUBE применяется для группировки по всем сочетаниям независимых критериев.

Задание: выдать общее количество рейсов – всего, для каждого аэропорта вылета, для каждого аэропорта прилета и для каждой комбинации аэропорта вылета и аэропорта прилета.

```
SELECT P1.airport_name, P2.airport_name, COUNT(*)
FROM Flights F JOIN Airports P1
     ON F.departure_airport = P1.airport_code
JOIN Airports P2
     ON F.arrival_airport = P2.airport_code
GROUP BY CUBE ((P1.airport_name), (P2.airport_name));
```

CUBE (A, B, C) – это то же самое, что и GROUPING SETS ((A, B, C), (A, B), (A, C), (B, C), (A), (B), (C), ()). CUBE ((A, B), (C, D)) – то же самое, что и GROUPING SETS ((A, B, C, D), (A, B), (C, D), ()).

Результат ответа на запрос со сложной группировкой может содержать NULL-значения разного происхождения.

Задание: выдать общее количество рейсов – всего, для каждого часа реального отправления, для каждого часа реального прибытия и для каждой комбинации часов реального отправления и реального прибытия.

```
WITH q AS
(SELECT flight_no,
    EXTRACT(HOUR FROM actual_departure) departure,
    EXTRACT (HOUR FROM actual_arrival) arrival
FROM flights)
SELECT departure, arrival, COUNT(*)
FROM q
GROUP BY CUBE (departure, arrival)
ORDER BY departure, arrival;
```

Здесь может возникнуть некоторая путаница, т.к. в таблице Flights есть рейсы, для которых не указано время реального отправления или время реального прибытия (NULL-значения в соответствующих столбцах). Эти NULL-значения в результате запроса легко спутать с NULL-значениями, указывающими агрегацию по всем значениям соответствующего критерия. В каких строках результата NULL-значение означает NULL-значение в исходных данных и в каких – агрегацию по всем значениям соответствующего критерия, сразу не очевидно.

Для того, чтобы отличить NULL-значения, полученные из исходных таблиц (означающие “неизвестно/неприменимо”), от NULL-значений, полученных в результате группировки по всем значениям критерия (означающие “всего”), используется функция **GROUPING.**

Функция **GROUPING (выражение_группировки)** возвращает значение 1 для некоторой строки, если при формировании этой строки использовалась агрегация по всем значениям указанного выражения, и 0 в противном случае.

```
WITH q AS
(SELECT flight_no,
    EXTRACT(HOUR FROM actual_departure) departure,
    EXTRACT (HOUR FROM actual_arrival) arrival
FROM flights)
SELECT CASE WHEN departure IS NULL
       THEN CASE WHEN GROUPING(departure) = 1
       THEN 'По всем' ELSE 'Нет' END
       ELSE CAST (departure AS CHAR(2)) END,
       CASE WHEN arrival IS NULL
       THEN CASE WHEN GROUPING(arrival) = 1
       THEN 'По всем' ELSE 'Нет' END
       ELSE CAST (arrival AS CHAR(2)) END,
       COUNT(*) FROM q
GROUP BY CUBE (departure, arrival)
ORDER BY departure, arrival;
```

Функция **GROUPING (столбец1, … столбецN)** возвращает значение, в котором каждый бит является признаком наличия или отсутствия для данной строки агрегации по всем значениям соответствующего столбца. Последний столбец в списке соответствует самому младшему биту (0), первый – самому старшему биту (2^{N-1}).

6. Задачи на проектирование схем баз данных и работу с ними

6.1. Пример выполнения задачи

Задача 1. Летопись острова Санта-Белинда

Примерно посередине воображаемого великого океана лежит воображаемый остров Санта-Белинда. Вот уже триста лет ведется подробная летопись острова. В летопись заносятся и данные обо всех людях, которые хоть какое-то время жили на острове. Записываются их имена, пол, даты рождения и смерти. Хранятся там и имена их родителей, если известно, кто они. У некоторых отсутствуют сведения об отце, у некоторых – о матери, а часть людей, судя по записям, – круглые сироты. Из летописи можно узнать, когда был построен каждый дом, стоящий на острове (а если сейчас его уже нет, то когда он был снесен), точный адрес и подробный план этого дома, кто и когда в нем жил.

Точно так же, как и столетия назад, на острове действуют предприниматели, занимающиеся, в частности, ловлей рыбы, заготовкой сахарного тростника и табака. Большинство из них делают все сами, а некоторые нанимают работников, заключая с ними контракты разной продолжительности. Имеются записи и о том, кто кого нанимал, на какую работу, когда начался и закончился контракт. Собственно, круг занятий жителей острова крайне невелик и не меняется веками. Неудивительно поэтому, что в летописи подробно описывается каждое дело, будь то рыбная ловля или выпечка хлеба. Все предприниматели – уроженцы острова. Некоторые объединяются в кооперативы, и по записям можно установить, кто участвовал в деле, когда вступил и когда вышел из него, каким паем владел. Имеются краткие описания деятельности каждого частного предпринимателя или кооператива, сообщающие, в том числе,

когда было начато дело, когда и почему прекращено.

Написать пакет, состоящий из процедур и функций, которые позволили выполнить указанные действия с базой данных:

1) Функция считает количество жителей острова, мужчин, женщин и детей по отдельности;
2) Найти количество предпринимателей, которые использую наемный труд, и указать поле их деятельности;
3) Найти количество людей, которые работают в наем;
4) Найти количество и название кооперативов, которые созданы в указанный диапазон дат.
5) Добавить нового жителя острова в базу данных.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

```
-- Создание базы данных
CREATE DATABASE annals_of_santa_belinda;

-- Создание таблицы Islanders
CREATE TABLE Islanders (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    sex VARCHAR(10) NOT NULL,
    date_of_birth DATE NOT NULL,
    date_of_death DATE,
    father_id INT,
     mother_id INT,
     FOREIGN KEY (father_id) REFERENCES Islanders(id),
     FOREIGN KEY (mother_id) REFERENCES Islanders(id)
);
```

```
-- Создание таблицы Houses
CREATE TABLE Houses (
    id SERIAL PRIMARY KEY,
    address VARCHAR(100) NOT NULL,
    build_date DATE NOT NULL,
    demolish_date DATE
);

-- Создание таблицы Contracts
CREATE TABLE Contracts (
    id SERIAL PRIMARY KEY,
    employer_id INTEGER NOT NULL,
    work_type VARCHAR(100) NOT NULL,
    contract_start_date DATE NOT NULL,
    contract_end_date DATE,
    FOREIGN KEY (employer_id) REFERENCES Islanders(id)
           ON DELETE CASCADE
           ON UPDATE CASCADE
);

-- Создание таблицы Entrepreneurs
CREATE TABLE Entrepreneurs (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    activity VARCHAR(100) NOT NULL
);

-- Создание таблицы Cooperatives
CREATE TABLE Cooperatives (
```

```
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    start_date DATE NOT NULL,
    end_date DATE
);

-- Схема для инкапсуляции объектов пакета:
CREATE SCHEMA island_management;

-- Функция для подсчета количества островитян, мужчин,
женщин и детей отдельно:
CREATE OR REPLACE FUNCTION count_islanders()
RETURNS TABLE (
    total_count BIGINT,
    men_count BIGINT,
    women_count BIGINT,
    children_count BIGINT
) AS $$
BEGIN
    RETURN QUERY SELECT
        COUNT(*) AS total_count,
        COUNT(*) FILTER (WHERE sex = 'Male') AS
men_count,
        COUNT(*) FILTER (WHERE sex = 'Female') AS
women_count,
        COUNT(*) FILTER (WHERE date_of_birth >
CURRENT_DATE - INTERVAL '18 years') AS children_count
    FROM Islanders;
END;
```

```
$$ LANGUAGE plpgsql;

-- Функция для определения количества
предпринимателей, использующих наемный труд, и
указания их сферы деятельности:
CREATE TYPE hiring_entrepreneurs_result AS (
    total_count INT,
    activities TEXT
);
CREATE OR REPLACE FUNCTION
count_hiring_entrepreneurs()
RETURNS hiring_entrepreneurs_result AS $$
DECLARE
    result hiring_entrepreneurs_result;
BEGIN
    SELECT COUNT(*) AS total_count,
string_agg(DISTINCT activity, ', ') AS activities
    INTO result
    FROM Entrepreneurs
    WHERE EXISTS (
        SELECT 1
        FROM Contracts
        WHERE Contracts.employer_id = Entrepreneurs.id
    );
    RETURN result;
END;
$$ LANGUAGE plpgsql;
```

```
-- Функция для определения количества людей,
работающих по найму:
CREATE OR REPLACE FUNCTION count_hired_workers()
RETURNS INT AS $$
BEGIN
    RETURN (
        SELECT COUNT(DISTINCT employee_id)
        FROM Contracts
    );
END;
$$ LANGUAGE plpgsql;

-- Функция для определения количества и названия
кооперативов, созданных в указанном диапазоне дат:

CREATE OR REPLACE FUNCTION count_cooperatives(
    start_date_param DATE,
    end_date_param DATE
)
RETURNS TABLE (
    total_count INT,
    cooperative_names TEXT
) AS $$
BEGIN
    RETURN QUERY
    SELECT COUNT(*) AS total_count, STRING_AGG(name,
', ') AS cooperative_names
    FROM Cooperatives
```

```
    WHERE start_date <= end_date_param AND end_date >=
start_date_param;
END;
$$ LANGUAGE plpgsql;

-- Процедура для добавления нового островитянина в
базу данных:
CREATE OR REPLACE PROCEDURE add_islander(
    p_name VARCHAR,
    p_sex VARCHAR,
    p_date_of_birth DATE,
    p_date_of_death DATE,
    p_father_id INTEGER,
    p_mother_id INTEGER
)
AS $$
BEGIN
    INSERT INTO Islanders (name, sex, date_of_birth,
       date_of_death, father_id, mother_id)
    VALUES (p_name, p_sex, p_date_of_birth,
       p_date_of_death, p_father_id, p_mother_id);
END;
$$ LANGUAGE plpgsql;

-- Создание триггеров и их функций:
CREATE OR REPLACE FUNCTION update_demolish_date()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Houses
```

```
    SET demolish_date = CURRENT_DATE
    WHERE id = OLD.id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER islander_delete_trigger
BEFORE DELETE ON Islanders
FOR EACH ROW
EXECUTE FUNCTION update_demolish_date();

CREATE OR REPLACE FUNCTION update_contract_end_date()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Contracts
    SET contract_end_date = CURRENT_DATE
    WHERE employer_id = OLD.id;
    RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER islander_delete_trigger_date
BEFORE DELETE ON Islanders
FOR EACH ROW
EXECUTE FUNCTION update_contract_end_date();

-- Эти триггеры гарантируют, что при удалении
островитянина будет автоматически обновлена
demolish_date в таблице Houses и contract_end_date в
таблице Contracts.
```

```
-- Пример наполнения таблиц:
INSERT INTO Islanders (name, sex, date_of_birth,
date_of_death, father_id, mother_id) VALUES
('John Doe', 'Male', '1980-01-01', NULL, NULL, NULL),
('Jane Smith', 'Female', '1990-05-15', NULL, NULL,
NULL),
('Michael Johnson', 'Male', '2007-08-20', NULL, NULL,
NULL),
('John Doerk', 'Male', '1980-01-01', NULL, NULL,
NULL),
('Ja Smith', 'Female', '1990-05-15', NULL, NULL,
NULL),
('Michael John', 'Male', '2009-08-20', NULL, NULL,
NULL),
('John Doels', 'Male', '1986-01-01', NULL, NULL,
NULL),
('Jane Smithy', 'Female', '1991-05-15', NULL, NULL,
NULL),
('Michael Son', 'Male', '2005-08-20', NULL, NULL,
NULL);

INSERT INTO Contracts (employer_id, work_type,
contract_start_date, contract_end_date) VALUES
   (1, 'Fishing', '2022-01-01', '2023-12-31'),
   (2, 'Harvesting', '2022-03-15', '2023-06-30'),
   (3, 'Baking', '2022-02-01', '2023-11-30'),
   (4, 'Cooking', '2022-01-01', '2023-12-31'),
   (5, 'Harvesting', '2022-03-15', '2023-06-30'),
```

```
   (9, 'Fishing', '2022-02-01', '2023-11-30');

INSERT INTO Houses (address, build_date,
demolish_date) VALUES
    ('123 Main Street', '2000-01-01', NULL),
    ('456 Elm Street', '1995-06-15', NULL),
    ('789 Oak Avenue', '1985-12-31', '2010-05-20');

INSERT INTO Entrepreneurs (name, activity) VALUES
    ('John Smith', 'Fishing'),
    ('Jane Johnson', 'Harvesting'),
    ('Michael Brown', 'Baking'),
    ('Michael Son', 'Fishing'),
    ('Jane Smithy', 'Harvesting'),
    ('John Doels', 'Cooking'),
    ('Michael John', 'Baking'),
    ('Ja Smith', 'Cooking'),
    ('John Doerk', 'Cooking'),
    ('Michael Johns', 'Fishing'),
    ('Jane Smith', 'Baking'),
    ('John Doe', 'Harvesting');

INSERT INTO Cooperatives (name, start_date, end_date)
VALUES
    ('Cooperative A', '2022-01-01', '2023-12-31'),
    ('Cooperative B', '2022-03-15', '2023-08-30'),
    ('Cooperative C', '2022-02-01', '2023-11-30');

-- Примеры использования функций, процедур и триггеров
из пакета:
```

```
-- Использование функции count_islanders() для
подсчета количества островитян, мужчин, женщин и
детей:
SELECT * FROM island_management.count_islanders();
Результат:
total_count | men_count | women_count | children_count
-------------------------------------------------
100         | 40        | 50          | 20

-- Использование функции count_hiring_entrepreneurs()
для подсчета количества предпринимателей, использующих
наемный труд, и их сферы деятельности:
SELECT * FROM count_hiring_entrepreneurs();
Результат:
total_count | activities
----------------------
50          | Fishing, Harvesting

-- Использование функции count_hired_workers() для
подсчета количества людей, работающих по найму:
SELECT island_management.count_hired_workers();
Результат:
25

-- Использование функции count_cooperatives() для
подсчета количества и названий кооперативов, созданных
в указанном диапазоне дат:
SELECT * FROM count_cooperatives('2023-01-01', '2023-
12-31');
```

```
Результат:
total_count | cooperative_names
-----------------------------
3           | Cooperative A, Cooperative B,
Cooperative C

-- Использование процедуры add_islander() для
добавления нового островитянина в базу данных:
CALL add_islander('Captain Jack Sparrow', 'Male',
'2000-05-10', NULL, NULL, NULL);
-- Процедура добавит нового островитянина с именем
"Captain Jack Sparrow", мужского пола, датой рождения
"2000-05-10" и без информации о родителях.

-- Использование триггеров update_contract_end_date и
update_contract_end_date для автоматического
обновления demolish_date и update_contract_end_date в
таблицах Houses и Contracts при удалении
островитянина:
DELETE FROM Islanders WHERE id = 5;
```

6.2.Задачи для решения

Задача 2. База данных хроники восхождений в альпинистском клубе.

В базе данных должны записываться даты начала и завершения каждого восхождения, имена и адреса участвовавших в нем альпинистов, название и высота горы, страна и район, где эта гора расположена. Дайте выразительные имена таблицам и полям, в которые могла бы заноситься указанная информация. Написать пакет, состоящий из процедур и функций,

которые позволили выполнить следующие действия с базой данных:

Для каждой горы показать список групп, осуществлявших восхождение, в хронологическом порядке.

Предоставить возможность добавления новой вершины, с указанием названия вершины, высоты и страны местоположения.

Предоставить возможность изменения данных о вершине, если на нее не было восхождения.

Показать список альпинистов, осуществлявших восхождение в указанный интервал дат.

Предоставить возможность добавления нового альпиниста в состав указанной группы.

Показать информацию о количестве восхождений каждого альпиниста на каждую гору.

Показать список восхождений (групп), которые осуществлялись в указанный пользователем период времени.

Предоставить возможность добавления новой группы, указав ее название, вершину, время начала восхождения.

Предоставить информацию о том, сколько альпинистов побывали на каждой горе.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 3. База данных медицинского кооператива.

Базу данных использует для работы коллектив врачей. В таблицы должны быть занесены имя, пол, дата рождения и домашний адрес каждого их пациента. Всякий раз, когда врач осматривает больного, явившегося к нему на прием, или сам приходит к нему на дом, он записывает дату и место, где проводится осмотр, симптомы, диагноз и предписания больному, проставляет имя пациента, а также свое имя. Если врач прописывает

больному какое-либо лекарство, в таблицу заносится название лекарства, способ его приема, словесное описание предполагаемого действия и возможных побочных эффектов. Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

1) По заданной дате определить количество вызовов в этот день.
2) Определить количество больных, заболевших данной болезнью.
3) По заданному лекарству определить его побочный эффект.
4) Предоставить возможность добавления нового лекарства с описанием его свойств в базе данных.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 4. База данных Городской Думы.

В базе хранятся имена, адреса, домашние и служебные телефоны всех членов Думы. В Думе работает порядка сорока комиссий, все участники которых являются членами Думы. Каждая комиссия имеет свой профиль, например вопросы образования, проблемы, связанные с жильем и так далее. Данные по каждой из комиссий включают: ее нынешний состав и председатель, прежние председатели и члены этой комиссии, участвовавшие в ее работе за прошедшие 10 лет, даты включения и выхода из состава комиссии, избрания ее председателей. Члены Думы могут заседать в нескольких комиссиях. В базу заносятся время и место проведения каждого заседания комиссии с указанием депутатов и служащих Думы, которые участвуют в его организации. Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Показать список комиссий, для каждой ее состав и председателя.

Предоставить возможность добавления нового члена комиссии.

Показать список членов муниципалитета, для каждого из них список комиссий, в которых он участвовал и/или был председателем.

Предоставить возможность добавления новой комиссии, с указанием председателя.

Для указанного интервала дат и комиссии выдать список ее членов с указанием количества пропущенных заседаний.

Предоставить возможность добавления нового заседания, с указанием присутствующих.

По каждой комиссии показать количество проведенных заседаний в указанный период времени.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 5. База данных рыболовной фирмы.

Фирме принадлежит небольшая флотилия рыболовных катеров. Каждый катер имеет «паспорт», куда занесены его название, тип, водоизмещение и дата постройки. Фирма регистрирует каждый выход на лов, записывая название катера, имена и адреса членов команды с указанием их должностей (капитан, боцман и т. д.), даты выхода и возвращения, а также вес пойманной рыбы отдельно по сортам (например, трески). За время одного рейса катер может посетить несколько банок. Фиксируется дата прихода на каждую банку и дата отплытия, качество выловленной рыбы (отличное, хорошее, плохое). На борту улов не взвешивается. Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Предоставить возможность добавления выхода катера в море с указанием команды.

Для указанного интервала дат вывести для каждого сорта рыбы список катеров с наибольшим уловом.

Для указанного интервала дат вывести список банок, с указанием среднего улова за этот период.

Предоставить возможность добавления новой банки с указанием данных о ней.

Для заданной банки вывести список катеров, которые получили улов выше среднего.

Вывести список сортов рыбы и для каждого сорта список рейсов с указанием даты выхода и возвращения, количества улова.

Для выбранного пользователем рейса и банки добавить данные о сорте и количестве пойманной рыбы.

Предоставить возможность пользователю изменять характеристики выбранного катера.

Для указанного интервала дат вывести в хронологическом порядке список рейсов в этот период времени. Для каждого их них вывести список сортов рыбы с указанием пойманного количества.

Предоставить возможность добавления нового катера.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 6. База данных фирмы, проводящей аукционы.

Фирма занимается продажей с аукциона антикварных изделий и произведений искусства. Владельцы вещей, выставляемых на проводимых фирмой аукционах, юридически являются продавцами. Лица, приобретающие эти вещи, именуются покупателями. Получив от продавцов партию предметов, фирма решает, на котором из аукционов выгоднее представить конкретный предмет. Перед проведением очередного аукциона каждой из выставляемых на нем вещей присваивается отдельный номер лота, играющий ту же роль, что и введенный ранее шифр товара. Две вещи,

продаваемые на различных аукционах, могут иметь одинаковые номера лотов.

В книгах фирмы делается запись о каждом аукционе. Там отмечаются дата, место и время его проведения, а также специфика (например, выставляются картины, написанные маслом и не ранее 1900 г.). Заносятся также сведения о каждом продаваемом предмете: аукцион, на который он заявлен, номер лота, продавец, отправная цена и краткое словесное описание. Продавцу разрешается выставлять любое количество вещей, а покупатель имеет право приобретать любое количество вещей. Одно и то же лицо или фирма может выступать и как продавец, и как покупатель. После аукциона служащие фирмы, проводящей аукционы, записывают фактическую цену, уплаченную за проданный предмет, и фиксируют данные покупателя.

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Для указанного интервала дат вывести список аукционов в хронологическом порядке с указанием наименования, даты и места проведения.

Добавить на указанный пользователем аукцион на продажу предмет искусства с указанием начальной цены.

Вывести список аукционов, с указанием суммарного дохода от продажи, отсортированных по доходу.

Для указанного интервала дат вывести список предметов, которые были проданы на аукционах в этот период времени.

Предоставить возможность добавления факта продажи на указанном аукционе заданного предмета.

Для указанного интервала дат вывести список продавцов с указанием общей суммы, полученной от продажи предметов в этот промежуток времени.

Вывести список покупателей, которые сделали приобретение в казанный интервал дат.

Предоставить возможность добавления записи о проводимом аукционе (место, время).

Для указанного места вывести список аукционов.

Для указанного интервала дат вывести список продавцов, которые принимали участие в аукционах, проводимых в этот период времени.

Предоставить возможность добавления и изменения информации о продавцах и покупателях.

Вывести список покупателей с указанием количества приобретенных предметов в указанный период времени.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 7. База данных музыкального магазина.

Таблицы базы данных содержат информацию о музыкантах, музыкальных произведениях и обстоятельствах их исполнения. Нескольких музыкантов, образующих единый коллектив, называются ансамблем. Это может быть классический оркестр, джазовая группа, квартет, квинтет и т. д. К музыкантам причисляют исполнителей (играющих на одном или нескольких инструментах), композиторов, дирижеров и руководителей ансамблей.

Кроме того, в базе данных хранится информация о пластинках, которыми магазин торгует. Каждая пластинка, а точнее, ее наклейка, идентифицируется отдельным номером, так что всем копиям, отпечатанным с матрицы в разное время, присвоены одинаковые номера. На пластинке

может быть записано несколько исполнений одного и того же произведения – для каждого из них в базе заведена отдельная запись. Когда выходит новая пластинка, регистрируется название выпустившей ее компании (например, EM1), а также адрес оптовой фирмы, у которой магазин может приобрести эту пластинку. Не исключено, что компания–производитель занимается и оптовой продажей своих пластинок. Магазин фиксирует текущие оптовые и розничные цены на каждую пластинку, дату ее выпуска, количество экземпляров, проданных за прошлый год и в нынешнем году, а также число еще не распроданных пластинок.

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

1) Посчитать количество музыкальных произведений заданного альбома.
2) Вывести название всех пластинок заданного альбома.
3) Показать лидеров продаж текущего года, то есть вывести название пластинок, наиболее часто продаваемых в текущем году.
4) Предусмотреть изменение данных о пластинках и ввод новых данных.
5) Предусмотреть ввод новых данных об ансамблях.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 8. База данных кегельной лиги.

Ставится задача спроектировать базу данных для секретаря кегельной лиги небольшого городка, расположенного на Среднем Западе. В ней секретарь будет хранить всю информацию, относящуюся к кегельной лиге, а средствами СУБД – формировать еженедельные отчеты о состоянии лиги. Специальный отчет предполагается формировать в конце сезона.

Секретарю понадобятся имена-фамилии, телефонные номера, и адреса всех игроков лиги. Так как в лигу могут входить только жители

городка, нет необходимости хранения для каждого игрока названия города и почтового индекса. Интерес представляют число набранных очков каждым игроком в еженедельной серии из трех встреч, в которых он принял участие, и его текущая результативность (среднее число набираемых очков в одной встрече). Секретарю необходимо знать для каждого игрока название команды, за которую он выступает, и имя-фамилию капитана каждой команды. Помимо названия, секретарь планирует назначить каждой команде уникальный номер.

Исходные значения результативности каждого игрока необходимы как при определении в конце сезона игрока, достигшего наибольшего прогресса в лиге, так и при вычислении гандикапа для каждого игрока на первую неделю нового сезона. Лучшая игра каждого игрока и лучшие серии потребуются при распределении призов в конце сезона.

Секретарь планирует включать в еженедельные отчеты информацию об общем числе набранных очков и общем числе проведенных игр каждым игроком, эта информация используется при вычислении их текущей результативности и текущего гандикапа. Используемый в лиге гандикап составляет 75% от разности между 200 и результативностью игрока, при этом отрицательный гандикап не допускается. Если результатом вычисления гандикапа является дробная величина, то она усекается. Перерасчет гандикапа осуществляется каждую неделю.

На каждую неделю каждой команде требуется назначать площадку, на которой она будет выступать. Эту информацию не нужно хранить в БД (Соперники выступают на смежных площадках).

Наконец, в БД должна содержаться вся информация, необходимая для проведения вычислений по получению положения команд. Команде засчитывается одна победа за каждую игру, в которой ей удалось набрать больше очков (выбить больше кеглей) (с учетом гандикапа), чем команде

соперников. Точно также команде засчитывается одно поражение за каждую встречу, в которой эта команда выбила меньшее число кеглей, чем команда соперников. Команде также засчитывается одна победа (поражение) в случае, если по сравнению с командой соперников ею набрано больше (меньше) очков за три встречи, состоявшиеся на неделе. Таким образом, на каждой неделе разыгрывается 4 командных очка (побед или поражений). В случае ничейного результата каждая команда получает 1/2 победы и 1/2 поражения. В случае неявки более чем двух членов команды, их команде автоматически засчитывается 4 поражения, а команде соперников – 4 победы. В общий результат команде, которой засчитана неявка, очки не прибавляются, даже если явившиеся игроки в этой встрече выступили, однако, в индивидуальные показатели – число набранных очков и проведенных встреч – будут внесены соответствующие изменения.

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Для указанного интервала дат показать список выступающих команд.

Предоставить возможность добавления новой команды.

Вывести список игровых площадок с указанием количества проведенных игр на каждой их них.

Для указанного интервала дат вывести список игровых площадок, с указанием списка игравших на них команд.

Предоставить возможность заполнения результатов игры 2-х команд на указанной площадке.

Вывести список площадок с указанием суммарной результативности игроков на каждой из них.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 9. База данных библиотеки.

Разработать информационную систему обслуживания библиотеки, которая содержит следующую информацию: названия книг, ФИО авторов, наименования издательств, год издания, количество страниц, количество иллюстраций, стоимость, название филиала библиотеки или книгохранилища, в которых находится книга, количество имеющихся в библиотеке экземпляров конкретной книги, количество студентов, которым выдавалась конкретная книга, названия факультетов, в учебном процессе которых используется указанная книга.

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

1) Для указанного филиала посчитать количество экземпляров указанной книги, находящихся в нем.
2) Для указанной книги посчитать количество факультетов, на которых она используется.
3) Предоставить возможность добавления и изменения информации о книгах в библиотеке.
4) Предоставить возможность добавления и изменения информации о филиалах.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 10. База данных по учету успеваемости студентов.

База данных должна содержать данные о контингенте студентов (фамилия, имя, отчество, год поступления, форма обучения (дневная/вечерняя/заочная), номер или название группы; об учебном плане (название специальности, дисциплина, семестр, количество отводимых на дисциплину часов, форма отчетности: экзамен/зачет); о журнале успеваемости студентов (год/семестр, студент, дисциплина, оценка).

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

1) Для указанной формы обучения вывести количество студентов этой формы.
2) Для указанной дисциплины получить количество часов и формы отчетности по этой дисциплине.
3) Предоставить возможность добавления и изменения информации о студентах, об учебных планах, о журнале успеваемости при этом предусмотреть курсоры, срабатывающие на пользовательские исключительные ситуации.
4) Предоставить возможность добавления и изменения информации о журнале успеваемости.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 11. База данных для учета аудиторного фонда университета.

База данных должна содержать следующую информацию об аудиторном фонде университета. Наименование корпуса, в котором расположено помещение, номер комнаты, расположение комнаты в корпусе, ширина и длина комнаты в метрах, назначение и вид помещения, подразделение университета, за которым закреплено помещение. В базе данных также должна быть информация о высоте потолков в помещениях в зависимости от места расположения помещений в корпусе. Следует также учитывать, что структура подразделений университета имеет иерархический вид, когда одни подразделения входят в состав других (факультет, кафедра, лаборатория, …).

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Рассчитать данные о площадях и объемах каждого помещения.

Для указанного корпуса получить количество факультетов, их названия и структуру, находящихся в этом корпусе.

Предоставить возможность добавления и изменения информации о корпусах в университете, при этом предусмотреть курсоры, срабатывающие на некоторые пользовательские исключительные ситуации.

Предоставить информацию о комнатах в корпусах университета, при этом предусмотреть курсоры, срабатывающие на некоторые пользовательские исключительные ситуации.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 12. База данных для регистрации происшествий.

Необходимо создать базу данных для регистрации происшествий. База данных должна содержать данные для регистрации сообщений о происшествиях (регистрационный номер сообщения, дата регистрации, краткая фабула (тип происшествия)); информацию о принятом по происшествию решении (отказано в возбуждении дел, удовлетворено ходатайство о возбуждении уголовного дела с указанием регистрационный номера заведенного дела, отправлено по территориальному признаку); информацию о лицах, виновных или подозреваемых в совершении происшествия (регистрационный номер лица, фамилия, имя, отчество, адрес, количество судимостей), отношение конкретных лиц к конкретным происшествиям (виновник, потерпевший, подозреваемый, свидетель, ...).

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Рассчитать данные о количестве происшествий в указанный промежуток времени.

Для указанного лица получить количество происшествий, в которых он зарегистрирован.

Предоставить возможность добавления и изменения информации о происшествиях, при этом предусмотреть курсоры, срабатывающие на некоторые пользовательские исключительные ситуации.

Предусмотреть возможность добавления и изменения информации о лицах, участвующих в происшествиях, при этом предусмотреть курсоры, срабатывающие на некоторые пользовательские исключительные ситуации.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 13. Базу данных для обслуживания работы конференции.

База данных должна содержать справочник персоналий участников конференции (фамилия, имя, отчество, ученая степень, ученое звание, научное направление, место работы, кафедра (отдел), должность, страна, город, почтовый индекс, адрес, рабочий телефон, домашний телефон, e-mail), и информацию, связанную с участием в конференции (докладчик или участник, дата рассылки первого приглашения, дата поступления заявки, тема доклада, отметка о поступлении тезисов, дата рассылки второго приглашения, дата поступления оргвзноса, размер поступившего оргвзноса, дата приезда, дата отъезда, потребность в гостинице).

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Для указанной даты первой рассылки вывести список приглашенных и посчитать их количество.

Предоставить возможность добавления приглашенных на конференцию с указанием оргвзноса и даты его уплаты.

Вывести список приглашенных с указанием даты оплаты оргвзноса.

Для указанного интервала дат вывести список участников, оплативших оргвзнос в этом диапазоне.

Для указанного города вывести название тезисов докладов, поступивших из этого города.

Для указанного города вывести список нуждающихся в гостинице из этого города.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 14. База данных для обслуживания склада.

База данных должна обеспечить автоматизацию складского учета. В ней должны содержаться следующие данные:

информация о “единицах хранения” – номер ордера, дата, код поставщика, балансный счет, код сопроводительного документа по справочнику документов, номер сопроводительного документа, код материала по справочнику материалов, счет материала, код единицы измерения, количество пришедшего материала, цена единицы измерения);

информация о хранящихся на складе материалах (справочник материалов – код класса материала, код группы материала, наименование материала);

информация о единицах измерения конкретных видов материалов – код материала, единица измерения (метры, килограммы, литры и т. д.).

информация о поставщиках материалов – код поставщика, его наименование, ИНН, юридический адрес (индекс, город, улица, дом), адрес банка (индекс, город, улица, дом), номер банковского счета.

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

1) Посчитать количество поставщиков данного материала.

2) Предоставить возможность добавления единицы хранения с указанием всех реквизитов.
3) Вывести список поставщиков заданного материала на склад.
4) Для указанного адреса банка посчитать количество поставщиков склада, пользующихся услугами этого банка.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 15. База данных фирмы.

Фирма отказалась от приобретения некоторых товаров у своих поставщиков, решив самостоятельно наладить их производство. С этой целью она организовала сеть специализированных цехов, каждый из которых принимает определенное участие в технологическом процессе.

Каждому виду выпускаемой продукции присваивается, как обычно, свой шифр товара, под которым он значится в файле товарных запасов. Этот же номер служит и шифром продукта. В записи с этим шифром указывается, когда была изготовлена последняя партия этого продукта, какова ее стоимость, сколько операций потребовалось.

Операцией считается законченная часть процесса производства, которая целиком выполняется силами одного цеха в соответствии с техническими требованиями, перечисленными на отдельном чертеже. Для каждого продукта и для каждой операции в базе данных фирмы заведена запись, содержащая описание операции, ее среднюю продолжительность и номер чертежа, по которому можно отыскать требуемый чертеж. Кроме того, указывается номер цеха, обычно производящего данную операцию.

В запись, связанную с конкретной операцией, заносятся потребные количества расходуемых материалов, а также присвоенные им шифры товара. Расходуемыми называют такие материалы, как, например, электрический кабель, который нельзя использовать повторно. Когда,

готовясь к выполнению операции, расходуемый материал забирают со склада, регистрируется фактически выданное количество, соответствующий шифр товара, номер служащего, ответственного за выдачу, дата и время выдачи, номер операции и номер наряда на проведение работ, который будет обсуждаться ниже. Реально затраченное количество материала может не совпадать с расчетным, из-за того, например, что часть изготовленной продукции бракуется.

Каждый из цехов располагает многочисленными инструментами и приспособлениями. При выполнении некоторых операций их все же не хватает, и цех вынужден обращаться в центральную инструментальную за недостающими. Каждый тип инструмента снабжен отдельным номером и на него заведена запись со словесным описанием. Кроме того, там отмечено, какое количество инструментов этого типа выделено цехам и какое осталось в инструментальной. Экземпляры инструмента конкретного типа, например гаечные ключи одного размера, различаются по своим индивидуальным номерам. На фирме для каждого типа инструмента имеется запись, содержащая перечень всех индивидуальных номеров. Кроме того, указаны даты их поступления на склад.

По каждой операции в фирме отмечают типы и количества инструментов этих типов, которые должны использоваться при ее выполнении. Когда инструменты действительно берутся со склада, фиксируется индивидуальный номер каждого экземпляра, указываются номер заказавшего их цеха и номер наряда на проведение работ. И в этом случае затребованное количество не всегда совпадает с заказанным.

Наряд на проведение работ по форме напоминает заказ на приобретение товаров, но, в отличие от последнего, он направляется не поставщику, а в один из цехов. Оформляется этот наряд после того, как руководство фирмы сочтет необходимым выпустить партию некоторого

продукта. В наряд заносятся шифр продукта, дата оформления наряда, срок, к которому должен быть выполнен заказ, а также требуемое количество продукта.

Написать пакет, состоящий из процедур и функций, которые позволили выполнить следующие действия с базой данных:

Для выбранного цеха выдать список операций, выполняемых им. Для каждой операции список расходуемых материалов, с указанием количества.

Показать список инструментов и предоставить возможность добавления нового.

Выдать список используемых инструментов.

Для указанного интервала дат вывести список нарядов.

Показать список операций и предоставить возможность добавления новой операции.

Выдать список расходуемых материалов в различных нарядах.

Выдать список товаров, с указанием используемых инструментов.

Показать список нарядов и предоставить возможность добавления нового.

Выдать отчет о производстве товаров различными цехами, указав наименование цеха, название товара и его количество.

Предусмотреть разработку триггеров, обеспечивающие каскадные изменения в связанных таблицах.

Задача 16. Магазин подержанных автомобилей.

Торговец подержанными автомобилями содержит штат служащих, в который входят агенты по продаже, секретари и механики. Агенты по продаже получают оклад плюс комиссионные, в то время как все остальные служащие получают почасовую оплату. Комиссионные составляют 5% для тех агентов по продаже, стаж работы которых менее трех лет, и 8% для тех, чей стаж составляет 3 и более лет.

Информация об имеющихся в наличии автомобилях включает в себя дату покупки, оценочную стоимость, объем ремонтных работ, которые должны быть выполнены до выставления на продажу, приблизительную стоимость этих работ, марку, модель, год выпуска и основной цвет.

Некоторые механики выполняют специальные виды работ, например, капитальный ремонт двигателя, исправление вмятин и др. Добавьте любые уместные по вашему мнению атрибуты.

Задача 17. Музыкальная коллекция.

Эту базу данных можно использовать для хранения и обработки информации о музыкальных записях, а также для автоматизации управления магазином «Музыка».

В БД необходимо хранить информацию о характеристиках альбома: название, дата издания, стиль музыкальных произведений, число дорожек (tracks), общее время звучания в минутах, тип носителя (компакт-диск, магнитофонная лента, грампластинка).

Интерес представляет информация и о песнях, представленных в альбоме (название песни, дата написания, стиль музыки, длительность песни в минутах).

Информация об исполнителях песен должна включать следующие данные: фамилия, имя, дата рождения, место рождения, дата смерти, основной стиль музыки. Кроме того, потребуются данные о песнях исполнителя (является ли исполнитель певцом или певцом и автором песни, дата записи или исполнения).

Надо учесть, что исполнитель может входить в состав одной или нескольких музыкальных групп. В этом случае в БД должна храниться информация о дате, когда исполнитель присоединился к данной группе и о дате, когда исполнитель покинул группу.

Информация о группах должна содержать следующее: название группы, дата основания, дата распада, основной стиль музыки.

Наконец, БД должна хранить информацию о стилях музыки.

Задача 18. Поваренная книга.

Эту базу данных можно использовать для хранения кулинарных рецептов или для ведения дел в ресторане.

Прежде всего, в БД должны храниться информация о блюдах: полное описание блюда, категория блюда (завтрак, ленч, обед, мясо, домашняя птица, рыбное блюдо, салат, закуска и пр.), цена блюда, если оно подается в ресторане.

Рецепт приготовления блюда должен содержать следующую информацию:

- ингредиенты – количество ингредиента для данного рецепта, единица измерения (столовая ложка, чайная ложка, грамм, чашка и т. д.);
- описание рецепта, категорию приготовляемого блюда, картинку готового блюда;
- этапы приготовления – номер этапа, описание того, что нужно выполнить.

Необходимо также хранить информацию об ингредиентах (описание ингредиента, наличие ингредиента, единица измерения, поставщик ингредиента) и их поставщиках (имя поставщика, почтовый индекс, город, адрес, номер телефона, факс, сумма текущей задолженности поставщику, дата следующей оплаты, сумма следующего платежа).

Задача 19. Книжный магазин.

Эта база данных может быть основой приложения, разрабатываемого для ведения дел в книжном магазине, либо для хранения данных о каждой книге личной библиотеки.

Прежде всего, представляет интерес информация об авторах: фамилия автора, дата рождения, дата смерти, место рождения, основная тематика работ.

Информация о книгах: код книги (ISBN), название, число страниц, тип книги (брошюра, в твердом переплете, в мягком переплете), издательство, дата выхода издания, цена, наличие книги, число проданных книг на определенную дату, код прохода между книжными стеллажами, номер стеллажа.

Информация об издательствах: название, адрес, телефон, факс.

Задача 20. Топночская атлетическая универсиада

Уполномоченный определил следующие параметры как имеющие наибольшее значение:

Для каждого учебного заведения, входящего в лигу:

- официальное название колледжа;
- число студентов;
- все виды спорта, культивируемые учебным заведением;
- логотип;
- название стадиона и его вместимость;
- название футбольного поля и его размеры;
- фамилия, адрес, номер домашнего телефона и номер служебного телефона каждого из тех, кто перечислен: президент колледжа, спортивный директор, заведующий отделом спортивной информации, представитель от факультета по вопросам спорта.

Список всех одобренных лигой судей, содержащий следующую информацию:

- фамилия и номер страхового полиса;
- домашний адрес;
- номер домашнего телефона;

- вид спорта, который обслуживает судья;
- рейтинг по результатам предыдущего сезона;
- соревнования наступающего сезона, на которые назначается данный судья.

Список всех студентов-спортсменов, содержащий следующую информацию:

- фамилию и номер страхового полиса;
- домашний адрес;
- адрес колледжа;
- номер телефона колледжа;
- текущая профилирующая дисциплина;
- средняя оценка;
- число учебных часов, оставшихся до окончания колледжа;
- число соревновательных сезонов по всем видам спорта, которыми студент занимался;
- дата поступления в колледж;
- число текущих зачетных часов;
- число сданных зачетных часов за два последних семестра.

Расписание лиги на текущий год, содержащее следующую информацию:

- команда, в данной встрече выступающая в роли хозяина;
- команда, в данной встрече выступающая в роли гостя;
- дата и время каждой встречи;
- назначенные судьи;
- вид спорта, по которому предполагается проведение соревнований.

По каждому виду спорта, культивируемому в лиге, имеется комитет по правилам, и один главный тренер назначается лигой в качестве председателя этого комитета.

Необходимые допущения сделать самим.

Задача 21. Круизы.

Даны три таблицы.

Первая содержит информацию о рейсах пассажирских судов: НОМЕР_РЕЙСА, ДАТА_ОТПЛЫТИЯ, ЧИСЛО_ПАССАЖИРОВ.

Суда, совершающие эти рейсы, имеют атрибуты НАЗВАНИЕ_СУДНА, ВМЕСТИМОСТЬ_СУДНА, НОМЕРА_РЕЙСОВ.

Есть информация о портах, в которые заходит судно, выполняющее конкретный рейс: ПОРТ, ПЛАТА_ЗА_СТОЯНКУ, НОМЕРА_РЕЙСОВ.

Приведите эти таблицы к пятой нормальной форме и выберите подходящие имена для новых таблиц и их полей.

Задача 22. Фирма «Тик-так».

Фирма «Тик-так» решила самостоятельно наладить производство некоторых товаров. С этой целью она организовала сеть специализированных цехов, каждый из которых принимает определенное участие в технологическом процессе. Каждому виду продукции, выпускаемой «Тик-так», присваивается свой шифр товара, под которым он значится в таблице товарных запасов. Этот же номер служит и шифром продукта. В записи с этим шифром указывается, когда была изготовлена последняя партия этого продукта, какова ее стоимость, сколько операций потребовалось.

Операцией считается законченная часть процесса производства, которая целиком выполняется силами одного цеха в соответствии с техническими требованиями, перечисленными на отдельном чертеже. Для каждого продукта и для каждой операции в «Тик-так» заведена запись, содержащая описание операции, ее среднюю продолжительность и номер чертежа, по которому можно отыскать требуемый чертеж. Кроме того, указывается номер цеха, обычно производящего данную операцию.

В запись, связанную с конкретной операцией, заносятся использованные количества расходуемых материалов, а также присвоенные им шифры товара. Расходуемыми называют такие материалы, как электрический кабель, который нельзя использовать повторно. Когда, готовясь к выполнению операции, расходуемый материал забирают со склада, регистрируется фактически выданное количество, соответствующий шифр товара, номер служащего, ответственного за выдачу, дата и время выдачи, номер операции и номер наряда на проведение работ, который будет обсуждаться ниже. Реально затраченное количество материала может не совпадать с требуемым, из-за того, например, что часть изготовленной продукции бракуется.

Каждый из цехов располагает многочисленными инструментами и приспособлениями. При выполнении некоторых операций их все же не хватает, и цех вынужден обращаться в центральную инструментальную за недостающими. В «Тик-так» каждый тип инструмента снабжен отдельным номером и на него заведена запись со словесным описанием. Кроме того, там отмечено, какое количество инструментов этого типа выделено цехам и какое осталось и инструментальной. Экземпляры инструмента конкретного типа, например гаечные ключи одного размера, различаются по своим индивидуальным номерам. На фирме для каждого типа инструмента имеется запись, содержащая перечень всех индивидуальных номеров. Кроме того, указаны даты их поступления па склад.

По каждой операции в «Тик-так» отмечают типы и количества инструментов этих типов, которые должны использоваться при ее выполнении. Когда инструменты действительно берутся со склада, фиксируется индивидуальный номер каждого экземпляра, указываются номер заказавшего их цеха и номер наряда на проведение работ. И в этом случае потребное количество не всегда совпадает с заказанным.

Наряд на проведение работ направляется в один из цехов. Оформляется этот наряд после того, как руководство «Тик-так» сочтет необходимым выпустить партию некоторого продукта. В наряд заносятся шифр продукта, дата оформления наряда, срок, к которому должен быть выполнен заказ, а также требуемое количество продукта.

Подберите имена полей и таблиц, в которых могла бы разместиться вся эта информация. Разработанная система таблиц должна обеспечивать возможность получения разнообразных справок в ответ на запросы, например, следующего характера:

Каковы количества и шифры всех расходуемых материалов, фактически потраченных при выполнении работ по наряду номер 6531?

Располагает ли фирма всеми инструментами, необходимыми для выпуска партии продукта номер 421729?

Где находится (на складе или в цехе с таким-то номером) каждый инструмент типа 321? В каких операциях, и при какой продукции он применяется?

Литература

1. ANSI/X3/SPARC Study Group on Data Base Management Systems. Interim Report // FDT (ACM SIGMOD bulletin). – 1975. – 7, № 2.
2. Elmasri R. Fundamentals of Database System. / R. Elmasri, S. B. Navathe – Redwood City: Publishing Company, 1998. – 873 p.
3. Астахова И.Ф. Практикум по информационным системам. ORACLE. / И.Ф. Астахова, А.С. Потапов, В.А. Чулюков и др. – Киев: Юниор, 2004. – 172 с.
4. Астахова И.Ф. СУБД: Язык SQL в примерах и задачах. / И.Ф. Астахова, А.П. Толстобров, В.М. Мельников и др. – М.: ФИЗМАТЛИТ, 2009 – 172 с.
5. Астахова И.Ф. Распределенные базы данных. ORACLE. / И.Ф. Астахова, Б.М. Дубов, О.А. Сидорова, Ю.В. Хицкова – Воронеж: Издат. дом ВГУ, 2022. – 158 с.
6. Вейскас Д. Эффективная работа с Microsoft Access 2/Перев. с англ. / Д. Вейскас – СПб: Питер, 1995. – 864 с.: ил.
7. Вендров А. М. CASE технологии современные методы и средства проектирования информационных систем. / А.М. Вендров – М.: Финансы и статистика, 1998. – 176 с.
8. Дейт, К.Дж. Введение в системы баз данных. 8-е издание.: Пер. с англ. / К.Дж. Дейт – М.: «Издательский дом Вильямс», 2005. – 1328 с.: ил.
9. Калянов Г.Н. CASE – структурный системный анализ (автоматизация и применение). / Г.Н. Калянов – М.: Лори, 1996. – 242 с.
10. Петрова И.Ю. Организация баз данных: уч. пособие для спец. «Автоматизированные системы обработки информации и управления» / И.Ю. Петрова, Е.А. Лазуткина – Астрахань: Изд-во АГТУ, 1999. – 340 с.
11. Ульман Дж. Базы данных на Паскале: Пер. с англ. М.В. Сергиевского, А.В. Шалашова; Под ред. Ю.И. Топчеева. / Дж. Ульман – М.:

Машиностроение, 1990. – 368 с., ил.

12. Фридман А. Л. Основы объектно-ориентированной разработки программных систем. / А.Л. Фридман – М.: Финансы и статистика, 2000. – 192 с. ил. – (Прикладные информационные технологии)
13. Документация к Postgres Pro Standard 14.2.1 ООО «Постгрес Профессиональный» Авторские права © 2016–2022 Постгрес Профессиональный»
14. Лузанов П. В., Рогов Е. В., Лёвшин И. В. Postgres. Первое знакомство. 8-е издание, переработанное и дополненное. – Москва: ООО «ППГ», 2022 https://postgrespro.ru/education/books/introbook.

Учебное издание

Астахова Ирина Федоровна

Борисенков Дмитрий Васильевич

Маковий Катерина Александровна

Хицкова Юлия Владимировна

УЧЕБНОЕ ПОСОБИЕ ПО РАБОТЕ С СУБД POSTGRESQL

Учебное пособие

Издано в авторской редакции