Министерство науки и высшего образования РФ
ФГБОУ ВО «Ульяновский государственный университет»
Факультет математики, информационных и авиационных технологий

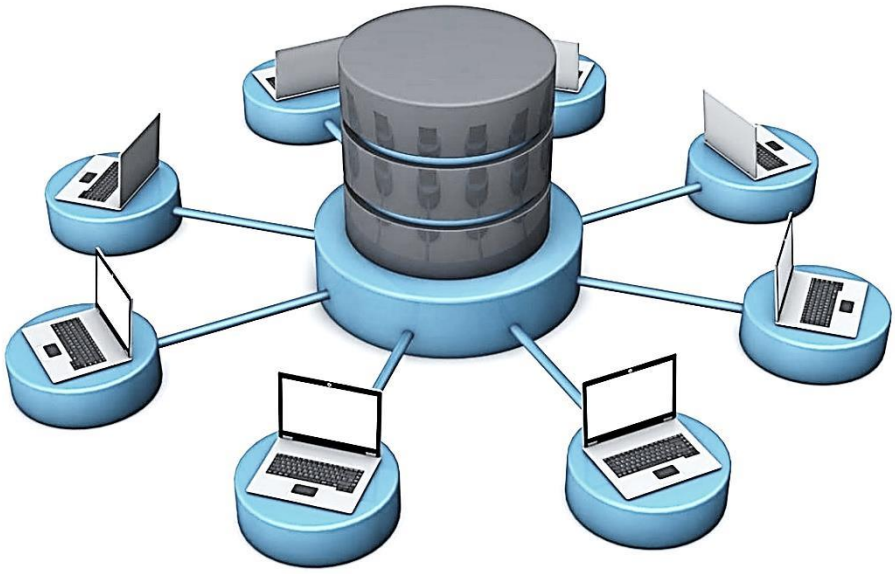
Кафедра телекоммуникационных технологий и сетей

Липатова Светлана Валерьевна

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
для семинарских (практических) занятий, лабораторного практикума
и самостоятельной работы
по дисциплине

«Базы данных»

для студентов факультета математики, информационных и авиационных технологий



Ульяновск
2019

Методические рекомендации для семинарских (практических) занятий, лабораторного практикума и самостоятельной работы по дисциплине «Базы данных» / составитель: С.В. Липатова - Ульяновск: УлГУ, 2022 – 148 с.

Настоящие методические рекомендации предназначены для студентов для студентов факультета математики, информационных и авиационных технологий. В работе приведены литература по дисциплине, темы дисциплины и вопросы в рамках каждой темы, рекомендации по изучению теоретического материала, контрольные вопросы для самоконтроля, задания для самостоятельной работы, задачи и упражнения для самостоятельной подготовки к семинарам или полностью самостоятельного освоения практических навыков, задания для лабораторного практикума и рекомендации по их выполнению.

Студентам всех форм обучения следует использовать данные методические рекомендации при подготовке к семинарам, самостоятельной подготовке, а также промежуточной аттестации по дисциплине «Базы данных».

Рекомендованы к введению в образовательный процесс

Учёным советом факультета математики, информационных и авиационных технологий УлГУ

протокол № 3/**22** от «19» апреля 2022 г.

СОДЕРЖАНИЕ

ОБЩИЕ ВОПРОСЫ

В результате изучения дисциплины «Базы данных» студенты должны изучить:

- модели структур данных;
- способы классификации СУБД в зависимости от реализуемых моделей данных и способов их использования;
- способы хранения данных на физическом уровне, типы и способы организации файловых систем;
- способы реляционной модели данных и СУБД, реализующих эту модель, языка запросов SQL;
- проблемы и основные способы их решения при коллективном доступе к данным;
- возможности СУБД, поддерживающих различные модели организации данных, преимущества и недостатки этих СУБД при реализации различных структур данных, средствами этих СУБД;
- этапы жизненного цикла базы данных, поддержки и сопровождения;
- специализированные аппаратные и программные средства ориентированных на построение баз данных больших объёмов хранения.

Методические рекомендации для семинарских (практических) занятий, лабораторного практикума и самостоятельной работы по дисциплине «Базы данных» направлены на повышение эффективности освоения знаний, умений, навыков и компетенций, связанных с:

- проектированием реляционных баз данных,
- программированием на языке SQL,
- способами хранения и обработки разнородных данных и т.д..

Методические рекомендации предлагают указания по всем темам дисциплины «Базы данных». Методические рекомендации разбиты по темам и содержат набор вопросов для систематизации теоретического материала, полученного на лекционных занятиях, и самостоятельного изучения теории, вопросы (тесты) для текущего контроля на практических занятиях (семинарах), задачи для усвоения практических навыков. Для лабораторного практикума приведены задания, варианты и рекомендации по выполнению лабораторных работ.

Список литературы и информационного обеспечения, приведённый в конце методических указаний, может служить основой для изучения всех рассматриваемых тем. Дополнительная и учебно-методическая литература могут быть использованы обучающимися для закрепления изучаемого материала.

РЕКОМЕНДАЦИИ ПО ОТДЕЛЬНЫМ ТЕМАМ ДИСЦИПЛИНЫ

Тема 1. Данные и базы данных. Эволюция концепций баз данных.

Основные вопросы темы

1. Определение данных и информации.
2. Классификация наборов данных, баз данных.

Рекомендации по изучению темы

Вопрос 1 изложен в учебнике [1] на с. 14-39.

Вопрос 1 изложен в учебнике [1] на с. 39-43.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Какие модели представления данных относят к первым (ранним)?
2. Как можно классифицировать наборы данных?
3. Какая СУБД считается первой?
4. Какой первый стандарт был принят по моделям представления данных?
5. Что подразумевает понятие Big Data?
6. Что подразумевает понятие noSQL?
7. Когда появилась реляционная модель представления данных?

Контрольные тесты

1. Первая реализация SQL была в:

Выберите один ответ:

- a. CODASYL (Conference of Data System Languages)
- b. нет правильного ответа
- c. СУБД Ingres
- d. СУБД System R

2. СУБД IBM System R является:

Выберите один ответ:

- a. иерархической
- b. многомерной
- c. серевой

d. реляционной

3. **Из какого проекта развивалась СУБД PostgreSQL?**

Выберите один ответ:

a. System R

b. Postgres95

c. POSTGRES

d. UniSQL

4. **Создали машины, работающие с перфокартами:**

Выберите один или несколько ответов:

a. Кодд

b. Лум

c. Жаккар

d. Стоунбрейкер

5. **К какому поколению относят реляционные базы данных?**

Выберите один ответ:

a. 3

b. 4

c. 1

d. 2

e. 5

6. **Автоматизированно обрабатываемые данных хранились ранее или могут хранится сейчас на:**

Выберите один или несколько ответов:

a. Жесткий диск

b. Перфокарта

c. Магнитная лента

d. Картотека

e. Оперативная память

Тема 2. Системы управления базами данных. СУБД PostgreSQL

Основные вопросы темы

1. Функции и структура СУБД.
2. Управление данными, управление транзакциями, журнализация изменений базы данных, восстановление после сбоев.
3. Особенности СУБД PostgreSQL.

Рекомендации по изучению темы

Вопросы 1,2 изложены в учебнике [1] на с. 39-43.

Вопросы 3 изложен в учебнике [5] в главе 2.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Какие основные пять функций СУБД?
2. Чем отличается мягкий сбой от жесткого и какие действия по восстановлению данных после них требуется предпринять?
3. Какие языки поддерживают СУБД?
4. Как выполняется запись в журнал СУБД?
5. Какую модель представления данных поддерживает PostgreSQL?
6. По каким параметрам можно классифицировать СУБД?
7. Какие утилиты требуются для работы с PostgreSQL?

Контрольные тесты

1. **PostgreSQL является специализированной СУБД (верно или нет)**

Выберите один ответ:

- Верно
- Неверно

2. **После восстановления после жесткого сбоя достаточно журнала изменений базы данных.**

Выберите один ответ:

- Верно
- Неверно

3. **Одна БД может находиться под управлением нескольких СУБД одновременно.**

Выберите один ответ:

- Верно
- Неверно

4. **Какие СУБД обладают свойствами объектно-реляционных СУБД?**

Выберите один или несколько ответов:

- a. Postgre95
- b. POSTGRES
- c. Oracle Database
- d. UniSQL
- e. Microsoft SQL Server
- f. PostgreSQL
- g. MySQL
- h. СУБД ЛИНТЕР
- i. Informix Universal Server
- j. SQLite

5. **К жесткому сбою могут привести следующие проблемы:**

Выберите один ответ:

- a. выход из стоя частично или полностью оперативной памяти
- b. неисправность жесткого диска
- c. ошибка операционной системы

6. **Система управления базами данных — это**

Выберите один ответ:

- a. прикладная программа для обработки текстов и различных документов
- b. оболочка операционной системы, позволяющая более комфортно работать с файлами

c. набор программ, обеспечивающий работу всех аппаратных устройств компьютера и доступ пользователя к ним

d. программная система, поддерживающая наполнение и манипулирование данными в файлах баз данных

7. Функциями СУБД являются:

Выберите один или несколько ответов:

- a. поддержка различных диалектов SQL
- b. управление данными во внешней памяти
- c. журнализация
- d. управление файлами
- e. управление буферами оперативной памяти
- f. распределение вычислительных ресурсов
- g. управление дисками
- h. поддержка языков БД
- i. управление взаимодействием одновременно работающих задач
- j. компиляция
- k. авторизация доступа к объектам

8. Язык для работы с СУБД должен обеспечивать:

Выберите один или несколько ответов:

- a. возможность описания объектов базы данных (схемы)
- b. возможность управления файлами
- c. возможность манипулирования данными
- d. возможность построения пользовательских отчетов

9. Одна БД может находиться под управлением нескольких СУБД одновременно.

Выберите один ответ:

- Верно
- Неверно

Тема 3. Модели представления данных

Основные вопросы темы

1. Понятие модели представления данных, классификация моделей. Инфологические, даталогические, физические модели.
2. Иерархическая модель, сетевая модель, реляционная, постреляционная, многомерная, объектно-ориентированная.

Рекомендации по изучению темы

Вопрос 1 изложен в учебнике [1] на с. 64-69.

Вопрос 1 изложен в учебнике [1] на с. 151-171 и [2] в лекции 2.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Какие недостатки у иерархической и сетевой модели представления данных?
2. В чем преимущества и недостатки реляционной модели представления данных?
3. Для чего используется инфологическая (концептуальная) модель представления данных?
4. В зависимости от объекта описания на какие классы делятся даталогические модели представления данных?
5. Какие модели представления данных относятся к теоретико-графовым?
6. Что означает свойство независимости данных?
7. Как связаны инфологическая, даталогическая и физические модели представления данных?

Контрольные тесты

1. **Даталогические модели:**

Выберите один или несколько ответов:

- a. диаграммы Бахмана
- b. основанные на странично-сегментной организации
- c. иерархическая
- d. дескриптивные

2. **Модель данных - это некоторая абстракция, которая будучи применительно к конкретным данным, позволяет пользователям и разработчикам трактовать**

их как информацию, то есть сведения, содержащие не только данные, но и взаимосвязь между ними.

Выберите один ответ:

- Верно
- Неверно

3. Иерархическая база данных может быть определена как

Выберите один ответ:

- a. БД, в которой информация организована в виде прямоугольных таблиц
- b. БД, в которой записи расположены в произвольном порядке
- c. БД, в которой существует возможность устанавливать дополнительно к вертикальным иерархическим связям горизонтальные связи
- d. БД, в которой элементы в записи упорядочены, т.е. один элемент считается главным, остальные подчиненными

4. Наиболее распространенными в практике являются:

Выберите один ответ:

- a. сетевые базы данных
- b. реляционные базы данных
- c. иерархические базы данных
- d. многомерные базы данных

5. Сетевая база данных – это

Выберите один ответ:

- a. БД, в которой элементы в записи упорядочены, т.е. один элемент считается главным, остальные подчиненными
- b. БД, в которой информация организована в виде прямоугольных таблиц
- c. БД, в которой принята свободная связь между элементами разных уровней
- d. БД, в которой записи расположены в произвольном порядке

Тема 4. Реляционная модель представления данных

Основные вопросы темы

1. Основные понятия и термины реляционной модели (n-арные отношение, схема отношения, кортеж, домен, ключ, первичный ключ, внешний ключ). Фундаментальные свойства отношений.
2. Реляционная алгебра. Операции реляционной алгебры (объединение, пересечение, разность, декартово произведение, проекция, ограничение, соединение, эквисоединение, деление).
3. Понятие целостности. Классификация ограничений целостности. Причины, вызывающие нарушение ограничений целостности. Способы задания ограничений целостности в современных СУБД. Ограничения целостности в стандартах SQL.

Рекомендации по изучению темы

Вопрос 1 изложен в учебнике [1] на с. 109-137.

Вопрос 2 изложен в учебнике [1] на с. 81-93.

Вопрос 3 изложен в учебнике [1] на с. 218-219.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Какие виды целостности существуют и какими механизмами они поддерживаются?
2. Что такое домен в реляционной модели данных?
3. Какие виды ограничений можно наложить на поле таблицы?
4. Какое обязательное ограничение для первичного ключа?
5. Что означает требование атомарности значений?
6. Что такое кортеж?
7. Что из себя представляет составной первичный ключ?

Контрольные тесты

1. **Требование целостности сущностей заключается в следующем: каждый кортеж любого отношения должен отличатся от любого другого кортежа этого отношения.**

Выберите один ответ:

- Верно
- Неверно

2. **Требование целостности по ссылкам состоит в следующем:**

для каждого значения внешнего ключа, появляющегося в родительском отношении, в дочернем отношении должен найтись кортеж с таким же значением первичного ключа.

Выберите один ответ:

- Верно
- Неверно

3. **Какое поле можно считать уникальным**

Выберите один ответ:

- a. поле, значение которого имеют свойство наращивания
- b. поле, которое носит уникальное имя
- c. поле, значения в котором не могут повторятся

4. **Домен в реляционных базах данных это...?**

Выберите один или несколько ответов:

- a. множество всех допустимых атомарных значений одного атрибута
- b. имя-символ, помогающее находить адреса интернет-серверов
- c. административная часть распределённой системы или домен управления службой каталогов
- d. собрание участников безопасности
- e. метаданные, абстрактно описывающие столбец таблицы БД, включая проверки и ограничения

5. **Кортеж – это**

Выберите один ответ:

- a. Строка таблицы
- b. Некоторый показатель, который характеризует числовым, текстовым или иным значением
- c. Совокупность однотипных данных
- d. Столбец таблицы

6. **Наиболее точным аналогом реляционной базы данных может служить**

Выберите один ответ:

- a. неупорядоченное множество данных

b. двумерная таблица

c. вектор

d. генеалогическое дерево

Тема 5. Проектирование реляционных баз данных

Основные вопросы темы

1. Концептуальные модели данных. Семантические модели данных. Семантическая модель Entity-Relationship (сущность-связь). Связи: один к одному, один ко многим, многие ко многим.
2. Понятие нормальной формы. Первая нормальная форма. Функциональная зависимость и вторая нормальная форма. Полная функциональная зависимость, транзитивная зависимость, третья нормальная форма. Нормальная форма Бойса-Кодда. Четвертая нормальная форма. Пятая нормальная форма. Необходимость нормализации. Использование нормальных форм при проектировании приложений в реляционных СУБД.
3. Средства автоматизации проектирования баз данных. Построение логической модели данных с использованием CASE-средств.

Рекомендации по изучению темы

Вопрос 1 изложен в учебнике [1] на с. 50-63 и [2] в лекциях 6-8.

Вопрос 2 изложен в учебнике [2] в лекции 5.

Вопрос 3 изложен в учебнике [1] на с. 69-79.

Дополнительные материалы по теме с примерами представлены в [4].

Справочные материалы по проектированию приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. В чем отличия в семантическом подходе к проектированию и нормализации?
2. На какой нормальной форме останавливаются на практике и почему?
3. Что такое контролируемая избыточность?
4. Приведите пример функциональной зависимости?
5. Какие свойства у нормальных форм?

6. Приведите пример транзитивной функциональной зависимости?
7. Если таблица имеет первичный ключ, то к какой нормальной форме она уже приведена?

Контрольные тесты

1. Обязательные требования к первичному ключу:

Выберите один или несколько ответов:

- ☐ a. значения первичного ключа не должны повторяться
- ☐ b. должен быть определен как UNIQUE
- ☐ c. должен быть числовым
- ☐ d. должен быть определен как NOT NULL
- ☐ e. состоит из одного поля
- ☐ f. уникальность
- ☐ c. должен состоять только из одного поля

2. В каком порядке создавать связанные таблицы?

Выберите один или несколько ответов:

- ☐ a. вначале ключи, а потом в любом таблицы
- ☐ b. в любом но без внешнего ключа, ключ создать после с помощью оператора alter table
- ☐ c. вначале дочернюю с внешним ключом затем родительскую с первичным
- ☐ d. вначале родительскую с первичным ключом затем дочернюю с внешним

Тема 6. SQL. Таблицы

Основные вопросы темы

1. Язык баз данных SQL. Средства управления и изменения схемы базы данных, определения ограничений целостности.
2. Операторы CRETE TABLE, ALTER TABLE, DROP TABLE.

Рекомендации по изучению темы

Вопрос 1 изложен в [6] на с. 3-11 и [3] глава 3.

Вопрос 2 изложен в [6] на с. 12-16.

Справочные материалы по операторам языка SQL для СУБД PosgreSQL приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Как можно использовать наследование таблиц?
2. Какие ограничения можно задавать для столбца, какие для таблицы?
3. Каким оператором можно переименовать таблицу?
4. Для каких изменений таблицы надо использовать CONSTRAINT?
5. Что необходимо учитывать при изменении типа столбца и каким оператором это можно сделать?
6. Что подразумевает использование CASCADE?
7. При наличии внешних ключей в каком порядке требуется применять операторы DROP?

Контрольные тесты

1. С помощью оператора alter table можно переименовать

Выберите один или несколько ответов:

- a. поле
- b. внешний ключ
- c. первичный ключ
- d. таблицу
- e. условие проверки поля
- f. домен

2. Выберете операторы, которые создадут таблицу с первичным ключом

Выберите один или несколько ответов:

- a. create table t1(p1 int not null, p2 int, primary key (p1))
- b. create table t1(p1 int, p2 int, primary key (p1))
- c. create table t1(p1 int not null , p2 int); alter table t1 add constraint pk1 primary key (p1);
- d. create table t1(p1 int not null , p2 int); alter table t1 add constraint fk1 foreign key (p1);

e. create table t1(p1 int not null primary key, p2 int)

3. **Что необходимо указать после REFERENCES**

Выберите один ответ:

a. родительскую таблицу и первичный ключ в ней

b. поле - первичный ключ и родительской таблице

c. досернюю таблицу и внешний ключ в ней

d. поле - внешний ключ в текущей таблице

4. **CHECK применяется только к одному полю при его определении (после задания названия и типа)**

Выберите один ответ:

Верно

Неверно

Тема 7. SQL. Операторы модификации данных

Основные вопросы темы

1. Средства ведения данных в таблицах базы данных.
2. Операторы INSERT, UPDATE, DELETE, TRUNCATE, COPY.

Рекомендации по изучению темы

Вопросы 1-2 изложены в [6] на с. 17-18.

Справочные материалы по операторам языка SQL для СУБД PosgreSQL приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Можно ли вставить несколько строк в таблицу одним оператором?
2. Для чего используется оператор COPY?
3. В чем отличия операторов DELETE и TRUNCATE TABLE?
4. Обязательно ли писать название столбцов в операторе INSERT?
5. Что делает оператор UPDATE?
6. Зачем часть WHERE в операторах DELETE и UPDATE?

7. Что делает оператор MERGE и есть ли он в PostgreSQL?

Контрольные тесты

1. **Что можно использовать для удаления таблицы t1?**

Выберите один ответ:

- a. drop from t1
- b. drop table t1
- c. delete table t1
- d. truncate table t1
- e. delete from t1

2. **Для вставки или изменения значения даты следует ее вводить**

Выберите один ответ:

- a. в квадратных скобках [12.05.2015]
- b. без всего 12.05.2015
- c. в апострофах `12.05.2015`
- d. в одинарных кавычках '12.05.2015'
- e. в круглых скобках (12.05.2015)
- f. в двойных кавычках "12.05.2015"

3. **Вставить значение по умолчанию в таблицу t1(p1 int, t2 int default 5) можно таким образом:**

Выберите один или несколько ответов:

- a. insert into t1 default values
- b. insert into t1(t1, t2) values (5, DEFAULT)
- c. insert into t1(t1, t2) values (5, 6)
- d. insert into t1 values default
- e. insert into t1(t1) values (4)

4. **Можно ли изменить значения столбцов на значения данных из других таблиц с помощью оператора update PostgreSQL?**

Выберите один или несколько ответов:

- a. можно с помощью FROM

- b. нет
- c. можно с помощью подзапроса

5. С помощью оператора insert можно вставить:

Выберите один или несколько ответов:

- a. изменить данные в поле или полях
- b. одну или несколько строк
- c. только одну строку
- d. данные из других таблиц
- e. вставить новую таблицу

6. Операторы модификации данных к каким объектам базы данных могут применяться?

Выберите один или несколько ответов:

- a. таблицы
- b. Курсоры
- c. Схемы
- d. Просмотры (виды)
- e. Функции
- f. Процедуры

7. Какие из перечисленных операторов SQL удалят из таблицы spr_student строчки в которых фамилия (столбец FIO), начинается с буквы А?

Выберите один ответ:

- a. Drop table spr_student where FIO like 'A%'
- b. Нет правильного ответа
- c. Delete from spr_student where FIO like 'A%'
- d. Все ответы приведут к желаемому результату
- e. Delete from spr_student where FIO like '%A%'

8. Как можно удалить данные из таблицы t1?

Выберите один или несколько ответов:

- a. truncate t1

- b. delete from t1
- c. drop from t1
- d. truncate table t1

9. Выберете операторы, которые выполнят вставку строк в таблицу, созданную оператором create table Product (id int, name varchar(50))?

Выберите один или несколько ответов:

- a. insert into Product (id, name) values (2, 'sugar')
- b. insert into Product values (1, 'soup')
- c. insert into Product select * from Product
- d. insert into Product values ('soup', 1)
- e. insert into Product (id) values (3)

10. Если в конце оператора модификации PostgeSQL добавить RETURNING *, то изменяемые данные будут выведены как результат запросы. В insert - новые данные, в delete - удаленные. Что выведется при update?

Выберите один ответ:

- a. новое содержимое изменённых строк
- b. и старое и новое содержимое изменённых строк
- c. старое содержимое изменённых строк

Тема 8. SQL. Запросы

Основные вопросы темы

1. Оператор выбора SELECT: предложения FROM, WHERE, GROUP BY, HAVING,ORDER BY.
2. Объединения результатов запросов, соединение таблиц.
3. Использование агрегатных и оконных функций.

Рекомендации по изучению темы

Вопрос 1 изложен в [6] на с. 20-26.

Вопрос 2 изложен в [6] на с. 26-31.

Вопрос 3 изложен в [6] на с. 33-35.

Справочные материалы по операторам языка SQL для СУБД PosgreSQL приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Как можно выполнить рекурсивный запрос?
2. Что такое подзапрос и какие подзапросы (в каких частях оператора) бывают?
3. Какое соотношение между полями часть после SELECT, часть ORDER и части GROUP?
4. Чем отличаются WHERE и HAVING?
5. Как происходит агрегация строк при использовании WHERE, GROUP, HAVING и без них?
6. Какие виды объединений наборов данных предоставляет SQL?
7. Какие операторы соответствуют пересечению, объединению и исключению множества кортежей?

Контрольные тесты

1. **Общие табличные выражения являются отдельно применяемым оператором и сохраняют данные во временную таблицу**

Выберите один ответ:

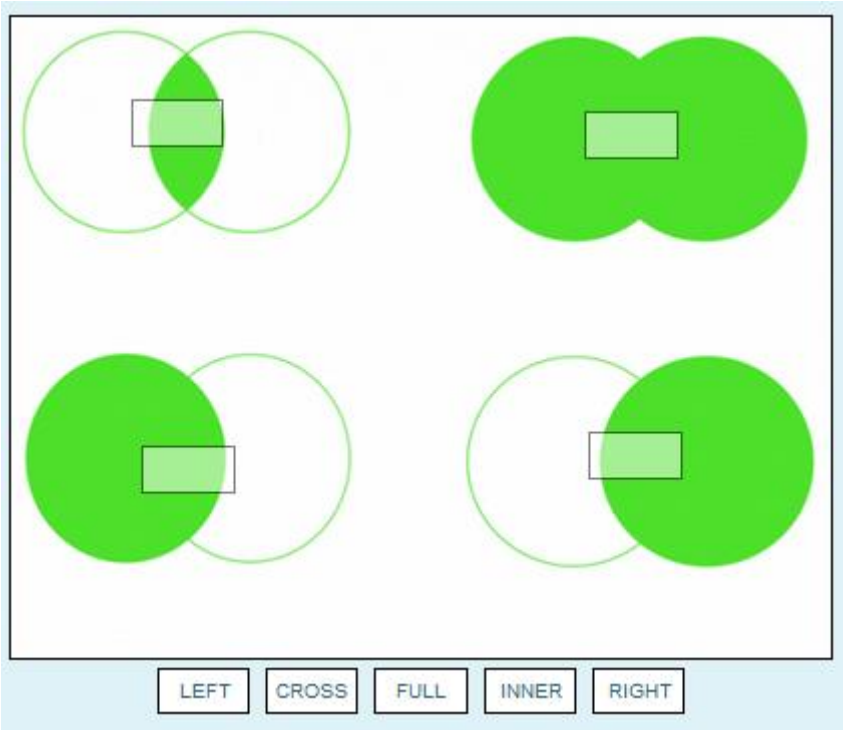
- Верно
- Неверно

2. **Какие из перечисленных операторов SQL возвратят список фамилий студентов (FIO), возраст (age) которых находится вне диапазона от 20 до 25 лет?**

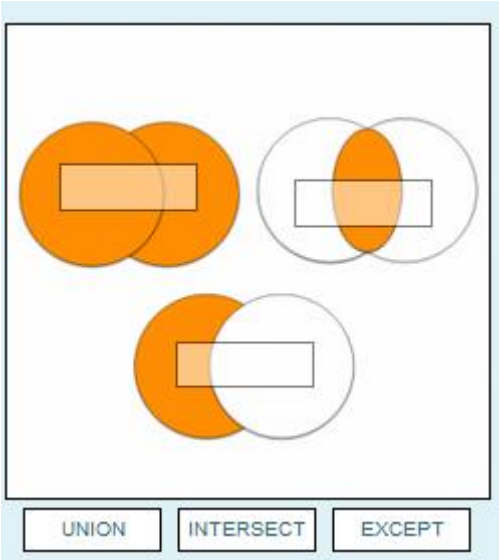
Выберите один ответ:

- a. Select FIO from Student where age not in (20,25)
- b. Select FIO from Student where (age > 20) and (age <25)
- c. Select FIO from Student where (age < 20) and (age > 25)
- d. Select FIO from Student where age not between 20 and 25
- e. Select FIO from Student where not (age > 20) and (age <25)
- f. Select FIO from Student where age not between (20,25)

3. **Сопоставьте соответствующий вид соединения таблиц (JOIN) на соответствующее отображение операции над множествами.**



4. **Сопоставьте операторы объединения результатов запроса схемам**



5. **Что выполняют оконные функции с их значениями (применяемые с OVER)**
   - last_value,
   - row_number
   - lead
   - first_value
   - Lag

Тема 9. SQL. Индексы, просмотры

Основные вопросы темы

1. Понятие индекса. Типы индексов. B-дерево. Операторы создания, изменения и удаления индекса.
2. Понятие и виды представления. Операторы работы с представлениями.

Рекомендации по изучению темы

Вопрос 1 изложен в [3] глава 11.

Вопрос 1 изложен в [3] глава 12.

Справочные материалы по операторам языка SQL для СУБД PosgreSQL приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Для чего используют индексы?
2. Для каких целей можно использовать представления?
3. Какие виды представлений бывают?
4. Какие виды индексов бывают?
5. Что из себя представляет B-дерево?
6. Можно ли создать индекс на часть данных таблицы?
7. Каким требованиям должно отвечать прибавление, чтобы через него можно было модифицировать данные в базовой таблице?

Контрольные тесты

1. **В настоящее время обеспечение уникальности поддерживают только индексы-B-деревья.**

Выберите один ответ:

- Верно
- Неверно

2. **К нематериализованному представлению можно создать индекс.**

Выберите один ответ:

- Верно

Неверно

3. Используя данное представление можно изменить данные в базовой таблице

```
CREATE VIEW Bonus
              AS SELECT DISTINCT snum, sname
                 FROM Elitesalesforce a
                 WHERE 10 < =
                    (SELECT COUNT (*)
                        FROM Elitesalestorce b
                        WHERE a.snum = b.snum)
```

Выберите один ответ:

Верно

Неверно

4. В каких запросах будет использован индекс (если он существует)?

Выберите один или несколько ответов:

a. select * from t1 where p1<>6

b. select * from t1 where p1>6

c. select * from t1 where p2 LIKE 'a%'

d. select * from t1 where p2 LIKE '%a'

e. select * from t1 where p1 is NULL

5. Можно ли создать несколько индексов к одной таблице?

Выберите один ответ:

a. Да

b. Только если наборы полей индексов не будут пересекаться

c. Нет

6. Выберете то, на основе чего можно построить индекс:

Выберите один или несколько ответов:

a. столбец таблицы возведенный в квадрат

b. запрос

c. столбец таблицы

d. курсор

- e. два столбца таблицы
- f. общее табличное выражение
- g. функция
- h. функция примененная к столбцу таблицы

7. Чего НЕ должно быть в запросе, на котором базируется изменяемое представление?

Выберите один или несколько ответов:

- a. where
- b. *
- c. group by
- d. order by
- e. as
- f. distinct
- g. CROSS JOIN

8. Использование WHERE в индексе превратит его в ...

Выберите один ответ:

- a. простой
- b. составной
- c. уникальный
- d. отсортированный по возрастанию
- e. частичный

9. Можно вставить, используя это представление, в таблицу Salespeople, людей, находящихся в Париже.

```
CREATE VIEW Londonstaff
            AS SELECT *
            FROM Salespeople
            WHERE                city                =                'London'
            WITH CHECH OPTION
```

Выберите один ответ:

- Верно

Неверно

10. **Значения столбцов, включенных в помощью INCLUDE, используются для построения B-дерева.**

Выберите один ответ:

Верно

Неверно

Тема 10. SQL. Транзакции

Основные вопросы темы

1. Понятие транзакции.
2. Способы организации транзакций и принципы блокировки доступа к данным.

Рекомендации по изучению темы

Вопрос 1 изложен в [3] глава 13.

Вопрос 2 изложен в [1] с. 241-245.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Какие 4 свойства у транзакций?
2. Что такое неявная транзакция?
3. Какой оператор позволяет откатить транзакцию?
4. Какие операторы должны входить в транзакцию, чтобы она успешно завершилась?
5. Как транзакция связана с многопользовательским доступом?
6. Какие режимы блокировок бывают?
7. Что такое взаимная блокировка транзакций?

Контрольные тесты

1. **Если пользователь не использует BEGIN и COMMIT, то транзакции не создаются.**

Выберите один ответ:

Верно

Неверно

2. **Если произведен откат вложенной транзакции может ли быть выполнена основная транзакция?**

Выберите один ответ:

- a. Нет
- b. Частично (операторы до вложенной транзакции)
- c. Да
- d. Частично (операторы после вложенной транзакции)

3. **Выберете, что может быть заблокировано (в зависимости от уровня блокировки)**

Выберите один или несколько ответов:

- a. функция
- b. страница
- c. таблица
- d. схема
- e. домен
- f. строка
- g. база данных

4. **ROLLBACK отменяет операторы транзакции после ее фиксации оператором COMMIT.**

Выберите один ответ:

- Верно
- Неверно

5. **Можно отменить часть транзакции, пока она не зафиксирована.**

Выберите один ответ:

- Верно
- Неверно

6. **Если пользователь использует оператор SELECT, то на соответствующие данные накладывается блокировка...**

Выберите один ответ:

- a. обе

- b. не накладывается
- c. чтения
- d. записи

Тема 11. SQL и PL/pgSQL. Процедуры, функции

Основные вопросы темы

1. Понятие процедуры и функции. Виды функций. Типы входных и выходных аргументов и способы их задания.
2. Основные операторы PL/pgSQL.

Рекомендации по изучению темы

Справочные материалы по операторам языка SQL для СУБД PosgreSQL приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Какие функции бывают?
2. На каких языках можно написать функцию для БД?
3. Что может возвращать функция в БД?
4. Какие операторы относятся к PL/SQL?
5. Есть ли отличия между процедурой и функцией в PostgreSQL?
6. Какие циклы можно использовать к теле функции?
7. Как вызывается функция?

Контрольные тесты

1. **Функций с одинаковыми именами в одной схеме быть не может.**

Выберите один ответ:

- Верно
- Неверно

EXECUTE в PostgreSQL используется для

Выберите один ответ:

- a. вызова функции

- b. выполнения запроса
- c. выполнения динамически сформированного кода
- d. вызова процедуры

2. **Определение функции содержит инструкцию LANGUAGE SQL. Выберете какие операторы могут использоваться в теле такой функции.**

Выберите один или несколько ответов:

- a. for
- b. select
- c. insert
- d. delete
- e. commit

3. **Вызвать функцию можно с помощью каких операторов?**

Выберите один или несколько ответов:

- a. переменная :=
- b. SELECT
- c. EXECUTE
- d. CALL
- e. PERFORM

4. **Выходную переменную можно определить в функции**

Выберите один или несколько ответов:

- a. после $$
- b. рядом с входной переменной написать OUT
- c. после RETURNS
- d. после RETURN
- e. после declare

5. **На каких языках можно написать функцию в PostgreSQL?**

Выберите один или несколько ответов:

- a. SQL

b. Pascal

c. C

d. PL/pgSQL

e. C++

6. **При вызове функции не обязательно передавать полный список аргументов.**

Выберите один ответ:

Верно

Неверно

Тема 12. SQL и PL/pgSQL. Курсоры

Основные вопросы темы

1. Понятие курсора, виды курсоров, алгоритм работы с курсором.
2. Операторы работы с курсорами, использование курсора в операторах модификации данных.

Рекомендации по изучению темы

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

3. Для чего используются курсоры?
4. В чем недостатки курсоров?
5. Как можно классифицировать курсоры?
6. Чем связанный курсор отличается от несвязанного?
7. Как можно использовать курсор совместно с операторами модификации данных?
8. Чем отличается FETCH от MOVE?
9. Чем отличается использование RELATIVE от ABSOLUTRE при движении по курсору?

Контрольные тесты

1. **Чем отличаются операторы FETCH и MOVE?**

Выберите один ответ:

a. FETCH не получат данные, а только переходит на строку

b. MOVE не получат данные, а только переходит на строку

c. Ничем, это синонимы

2. Какой синтаксис оператора открытия курсора соответствует курсору с динамически формируемым запросом?

Выберите один ответ:

a. OPEN переменная_курсора [[NO] SCROLL] FOR запрос;

b. OPEN переменная_курсора[([имя_аргумента:=] значение_аргумента[, ...])];

c. OPEN переменная_курсора[[NO] SCROLL] FOR EXECUTE строка_запроса[USING выражение[, ...]];

3. Какие операторы можно применять к открытому курсору?

Выберите один или несколько ответов:

a. DELETE

b. UPDATE

c. MOVE

d. FETCH

e. CLOSE

f. INSERT

4. Какой вид цикла ориентирован на работу с курсором (автоматически открывает и закрывает курсор при выходе из цикла)?

Выберите один ответ:

a. FOR

b. WHILE

c. LOOP

5. Можно ли применять к курсору, объявленному таким образом:

DECLARE curs1 NO SCROLL CURSOR refcursor;

следующий оператор

FETCH ABSOLUTE -1 FROM curs1?

Выберите один ответ:

Верно

Неверно

Тема 13. SQL и PL/pgSQL. Триггеры

Основные вопросы темы

1. Понятие триггера и триггерной функции.
2. Виды триггеров. Операторы работы с триггерами.

Рекомендации по изучению темы

Справочные материалы по операторам языка SQL для СУБД PosgreSQL приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Когда срабатывает триггер?
2. Можно ли отключить триггер?
3. Чем отличается триггер BEFORE от AFTER и INSTEAD OF?
4. Чем отличается триггер событий от триггера на изменения данных?
5. Приведите примеры событий для триггеров?
6. Что такое триггерная функция?
7. Можно ли один триггер установить на разные операторы модификации данных?

Контрольные тесты

1. **В PostgeSQL перед созданием триггера необходимо создать триггерную функцию.**

Выберите один ответ:

- Верно
- Неверно

2. **Специальные переменные OLD и NEW являются составными (по типу строки) и к конкретному полю можно обращаться New.поле / Old.поле.**

Выберите один ответ:

- Верно
- Неверно

3. **Какую команду можно использовать в теле триггерной функции для отмены оператора?**

Выберите один ответ:

- a. ROLLBACK
- b. RAISE EXCEPTION
- c. COMMIT
- d. RAISE NOTICE

4. В каком порядке будет выполняться триггеры, относящиеся к одному событию?

Выберите один ответ:

- a. По сортировке в обратном алфавитному порядке их имен
- b. По сортировке в порядке убывания их OID
- c. По сортировке в порядке возрастания их OID
- d. В случайном
- e. По сортировке в алфавитном порядке их имен

5. Каким объектам базы данных могут назначаться триггеры?

Выберите один или несколько ответов:

- a. таблица
- b. схема
- c. функция
- d. представление
- e. сторонняя таблица

6. Для каких событий можно определить триггеры на уровне строк:

Выберите один или несколько ответов:

- a. DROP
- b. INSERT
- c. UPDATE
- d. TRUNCATE
- e. DELETE
- f. ALTER

- [ ] g. CREATE

7. Во время выполнения триггера событий доступны специальные переменные:

Выберите один или несколько ответов:

- [ ] a. TG_OP
- [ ] b. TG_WHEN
- [ ] c. TG_TAG
- [ ] d. TG_NAME
- [ ] e. TG_EVENT

Тема 14. SQL. Роли, привилегии и операторы для работы с ними

Основные вопросы темы

1. Понятие роли, связь роли с понятиями пользователь, группа пользователей, схема базы данных. Предопределенные роли.
2. Операторы ведения ролей. Привилегии и операторы по назначению и отмене привилегий. Виды привилегий.

Рекомендации по изучению темы

Справочные материалы по операторам языка SQL для СУБД PosgreSQL приведены в соответствующей теме раздела «Лабораторный практикум» данного пособия.

Вопросы для самоподготовки

Рекомендуется после изучения материалов лекций и специальной литературы подготовить ответы на вопросы:

1. Чем роли отличаются от пользователей?
2. Как привилегии существуют?
3. Как назначить привилегию?
4. Как назначить одну роль другой роли?
5. Как отменить привилегию?
6. Что такое предопределенная роль и приведите ее примеры?
7. Одинаковы ли привилегии для таблиц и функций, приведите примеры?

Контрольные тесты

1. **Если подключился под определенной ролью, то не можешь ее сменить до конца сеанса.**

Выберите один ответ:

- Верно
- Неверно

2. **Для того чтобы исключить роль из другой роли (группы) надо использовать оператор:**

Выберите один ответ:

- a. DROP ROLE
- b. GRANT
- c. REVOKE
- d. DELETE ROLE
- e. CREATE ROLE

3. **Какие привилегии можно назначить пользовательской таблице?**

Выберите один или несколько ответов:

- a. TEMP
- b. EXECUTE
- c. TRUNCATE
- d. SELECT
- e. TRIGGER
- f. USAGE
- g. CREATE
- h. UPDATE
- i. CONNECT
- j. TEMPORARY
- k. REFERENCES
- l. ALL PRIVILEGES

☐ m. DELETE

☐ n. INSERT

4. **Какие из утверждений верные?**

Выберите один или несколько ответов:

☐ a. В других СУБД (например ORACLE, MS SQL Server) это разные операторы, понятия роли и пользователя там различны.

☐ b. Операторы CREATE ROLE, CREATE USER являются синонимами в PostgreSQL, а CREATE GROUP нет.

☐ c. Операторы CREATE ROLE, CREATE GROUP являются синонимами в PostgreSQL, а CREATE USER нет.

☐ d. Все три оператора создают разные объекты базы данных.

☐ e. Операторы CREATE ROLE, CREATE USER, CREATE GROUP являются синонимами в PostgreSQL.

5. **Выберете оператор который делает активным идентификатор пользователя сеанса:**

Выберите один или несколько ответов:

☐ a. SET ROLE user1

☐ b. RESET SESSION AUTHORIZATION

☐ c. SET ROLE NONE

☐ d. RESET ROLE

☐ e. SET SESSION AUTHORIZATION user1

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Порядок выполнения лабораторных работ может быть произвольным и определяется уровнем освоения компетенций обучающегося.

Тема 5: Проектирование реляционных баз данных

Задание лабораторной работы

Цель работы: Получение практических навыков проектирования реляционных баз данных и работы в CASE-средстве.

Задание: спроектировать согласно варианту реляционную базу данных, используя методологию IDEF1X, для этого необходимо:

- построить ER-модель (физическую) с помощью CASE-средства, например Open System Architect (не менее 7 сущностей),
- привести ER-модель к 3 нормальной форме,
- описать ограничения целостности,
- вывести программный код создания БД на языке целевой СУБД (прямое проектирование),
- перенести код в СУБД (выполнить),
- * внести изменения в схему БД и построить новую физическую модель измененной базы данных (обратное проектирование), если доступно создание ODBC источника данных.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Краткое описание базы данных (описываемую базой предметную область, выделенные сущности)
3. Модель базы данных
4. Код создания БД на языке целевой СУБД
5. Снимок экрана экспортированной базы в СУБД.
6. * Снимок измененной структуры БД
7. * Модель измененной БД

Методические указания по выполнению лабораторной работы

CASE-средства для проектирования баз данных

Термин CASE (Computer Aided Software Engineering) используется в настоящее время в весьма широком смысле. Первоначальное значение термина CASE, ограниченное

вопросами автоматизации разработки только лишь программного обеспечения (ПО), в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных информационных систем (ИС) в целом. Теперь под термином «CASE-средства» понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным программным обеспечением и техническими средствами образуют полную среду разработки информационной системы.

CASE-система для проектирования БД используется при моделировании и создании баз данных произвольной сложности на основе диаграмм «сущность-связь».

Информационная модель в системе представлена в виде диаграмм «сущность-связь» в нотации IDEF1x (Integrated DEFinition) или IE (Information Engineering), отражающих основные объекты предметной области и связи между ними. Дополнительно определяются атрибуты сущностей, характеристики связей, индексы и бизнес-правила, описывающие ограничения и закономерности предметной области. После создания ER-диаграммы система автоматически генерирует SQL-код для создания таблиц, индексов и других объектов базы данных. По заданным бизнес-правилам формируются стандартные триггеры БД для поддержки целостности данных; для сложных бизнес-правил можно создавать собственные триггеры, используя библиотеку шаблонов.

Система может осуществлять реинжиниринг существующих БД, генерируя ER-диаграммы по SQL-текстам. Таким образом, полностью поддерживается технология FRE (forward and reverse engineering – прямое и обратное построение), последовательность этапов которой приведена ниже:

- импорт с сервера существующей БД;
- автоматическая генерация модели БД;
- модификация модели;
- автоматическая генерация новой схемы и построение физической БД на том же самом или любом другом сервере.

Примером CASE-средств для работы с БД являются:

- ERWIN (прекращена поддержка ПО);
- Toad Data Modeler (http://www.casestudio.com/enu/default.aspx);
- SQL Power Architect (http://www.sqlpower.ca/page/architect);
- Open System Architect (http://www.codebydesign.com/SystemArchitect/downloads/);

- DB Designer (http://www.fabforce.net/dbdesigner4/).

Логическое проектирование реляционной базы данных

При проектировании любой базы данных всегда следует иметь в виду конечного пользователя. Логическое проектирование базы данных (также называемое построением ее *логической модели*) представляет собой процесс объединения данных в логически организованные группы объектов, которые можно легко поддерживать. Логическое проектирование базы данных должно приводить к уменьшению повторяющейся информации или даже полному ее устранению. В конце концов, зачем хранить одни и те же данные дважды? Кроме того, все используемые в базе данных соглашения о наименованиях также должны быть стандартными и логически обоснованными.

Проектирование схемы базы данных может быть выполнено двумя путями:

- путем декомпозиции (разбиения), когда исходное множество отношений, входящих в схему базы данных заменяется другим множеством отношений (число их при этом возрастает), являющихся проекциями исходных отношений;
- путем синтеза, то есть путем компоновки из заданных исходных элементарных зависимостей между объектами предметной области схемы базы данных.

При проектировании реляционной базы данных необходимо решить вопрос о наиболее эффективной структуре данных. Основные цели, которые при этом преследуются: обеспечить быстрый доступ к данным в таблицах; исключить ненужное повторение данных (избыточность), которая может явиться причиной ошибок при вводе и нерационального использования дискового пространства компьютера; обеспечить целостность данных таким образом, чтобы при изменении одних объектов автоматически происходили соответствующие изменения связанных с ними объектов. Корректной является схема базы данных, в которой отсутствуют нежелательные зависимости между атрибутами отношении.

Реляционная база данных - это набор отношений. Но не просто набор, а нормализованный набор отношений.

Нормализация базы данных

Нормализация представляет собой процесс, направленный на уменьшение избыточности информации в базе данных. Кроме самих данных, в базе данных также могут быть нормализованы различные наименования, имена объектов и выражения.

Нормализация — это процесс, направленный на уменьшение избыточности информации в реляционной базе данных.

Нормальная форма — это своеобразный показатель уровня, или глубины, нормализации базы данных. Уровень нормализации базы данных соответствует нормальной форме, в которой она находится.

Основные преимущества нормализации:

- Лучшая общая организация базы данных
- Сокращение избыточности информации
- Непротиворечивость информации внутри базы данных
- Более гибкий проект базы данных
- Большая безопасность данных

Процесс нормализации данных приводит к улучшению их общей организации, тем самым облегчая работу каждому — от пользователя, который обращается к таблицам, до администратора базы данных (DBA), ответственного за управление всеми объектами базы данных в целом. Снижение избыточности данных влечет за собой упрощение их структуры и способствует рациональному использованию дискового пространства. Вследствие минимизации дублирующей информации значительно уменьшается вероятность появления противоречивых данных.

Нормализация базы данных с разбиением ее на более мелкие таблицы дает большую гибкость при изменении существующих структур данных.

Нормализация способствует повышению безопасности информации в том смысле, что администратор базы данных может предоставлять некоторым пользователям доступ лишь к ограниченному числу таблиц. После проведения нормализации базы данных организация защиты хранимой в ней информации значительно упрощается.

Целостность данных связана с обеспечением непротиворечивости и достоверности информации, хранящейся внутри базы данных.

Нормальные формы

Классический подход - процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы - значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF);
- Доменно-ключевая нормальная форма (DKNF);
- Шестая нормальная форма (6NF).

На практике часто ограничиваются третьей.

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации, декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости. Для дальнейшего изложения потребуются несколько определений.

Определение. *Функциональная зависимость*

В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R.X (r) R.Y.

Определение. *Полная функциональная зависимость*

Функциональная зависимость R.X (r) R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

Определение. *Транзитивная функциональная зависимость*

Функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X -> R.Z и R.Z -> R.Y и отсутствует функциональная зависимость R.Z --> R.X. (При отсутствии последнего требования мы имели бы "неинтересные" транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

Определение. *Неключевой атрибут*

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа (в частности, перв
ичного).

Определение. *Взаимно независимые атрибуты*

Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

Определение. *Детерминант*

Детерминант - любой атрибут, от которого полностью функционально зависит некоторый другой атрибут.

Определение. *Многозначные зависимости*

В отношении R (A, B, C) существует многозначная зависимость R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

Определение. *Зависимость соединения*

Отношение R (X, Y, ..., Z) удовлетворяет зависимости соединения * (X, Y, ..., Z) в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, Y, ..., Z.

Функциональные зависимости определяют не текущее состояние базы данных, а все возможные ее состояния, то есть они отражают те связи между атрибутами, которые

присущи реальному объекту, который моделируется с помощью базы данных. Поэтому определить функциональные зависимости но текущему состоянию базы данных можно только в том случае, если экземпляр базы данных содержит абсолютно полную информацию (то есть никаких добавлений и модификации базы данных не предполагается). В реальной жизни это требование невыполнимо, поэтому набор функциональных зависимостей задает разработчик, системный аналитик, исходя из глубокого системного анализа предметной области.

Из-за небрежного проектирования базы данных появляется *избыточность*данных (повторение), которая приводит не только к потере лишнего места; она может вызвать нарушение целостности, то есть привести к пртиворечивости данных в базе данных. Несоотвествия между данными (противоречивость) называются аномалиями. Существут три вида аномалий:

- *аномалия обновления*–противоречивость данных, вызванная их избыточностью и их частичным обновлением;

- *аномалия удаления*–непреднамеренная потеря данных, вызванная удалением других данных;

-*аномалия ввода* – невозможность ввести данные в таблицу, вызванная отсутствием других данных.

Очевидно, что аномалии обновления, удаления и ввода нежелательны. Чтобы свести к минимуму проблемы, возникающие из-за аномалий, используется формальный метод, называемый разбиением. *Разбиение*–процесс разделения таблиц на несколько таблиц в целях избавления от аномалий и поддержания целостности данных. Для этого используются нормальные формы или правила структурирования таблиц.

Первая нормальная форма

Определение. *Первая нормальная форма*

Отношение, в котором на пересечении каждой строки и каждого столбца содержится только одно значение, находится в первой нормальной форме.

Не всегда легко определить, является ли атрибут скалярным. Например, дата состоит из трех различных компонентов: дня, месяца и года. Дату можно хранить как три различных атрибута или как единое целое. Выбор зависит от особенностей моделируемой предметной области. Если дата используется как отдельная величина, то она скалярная. Но если система работает с отдельными составляющими даты, лучше хранить ее в качестве набора из трех различных атрибутов. Аналогично можно по-разному моделировать имена: атрибут «Имя» может содержать полное имя (ФИО) элемента, а

можно каждую составляющую часть отнести к разным атрибутам; также и с адресом: полный адрес или нет.

Таблица находится в первой нормальной форме, если удовлетворяет следующими требованиям (требования реляционной модели):

- таблица не имеет повторяющихся записей;
- в таблице должны отсутствовать повторяющиеся группы полей;
- строки должны быть не упорядочены;
- столбцы должны быть не упорядочены.

Вторая нормальная форма

Рассмотрим следующий пример схемы отношения:

СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ

(СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Первичный ключ:

СОТР_НОМЕР, ПРО_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР -> СОТР_ЗАРП

СОТР_НОМЕР -> ОТД_НОМЕР

ОТД_НОМЕР -> СОТР_ЗАРП

СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН

Как видно, хотя первичным ключом является составной атрибут СОТР_НОМЕР, ПРО_НОМЕР, атрибуты СОТР_ЗАРП и ОТД_НОМЕР функционально зависят от части первичного ключа, атрибута СОТР_НОМЕР. В результате мы не сможем вставить в отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ кортеж, описывающий сотрудника, который еще не выполняет никакого проекта (первичный ключ не может содержать неопределенное значение). При удалении кортежа мы не только разрушаем связь данного сотрудника с данным проектом, но утрачиваем информацию о том, что он работает в некотором отделе. При переводе сотрудника в другой отдел мы будем вынуждены модифицировать все кортежи, описывающие этого сотрудника, или получим несогласованный результат. Такие неприятные явления называются аномалиями схемы отношения. Они устраняются путем нормализации.

Определение. *Вторая нормальная форма (в этом определении предполагается, что единственным ключом отношения является первичный ключ)* Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа.

Можно произвести следующую декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ в два отношения СОТРУДНИКИ-ОТДЕЛЫ и СОТРУДНИКИ-ПРОЕКТЫ:

СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР)

Первичный ключ:

СОТР_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР -> СОТР_ЗАРП

СОТР_НОМЕР -> ОТД_НОМЕР

ОТД_НОМЕР -> СОТР_ЗАРП

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Первичный ключ:

СОТР_НОМЕР, ПРО_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН

Каждое из этих двух отношений находится в 2NF, и в них устранены отмеченные выше аномалии (легко проверить, что все указанные операции выполняются без проблем).

Если допустить наличие нескольких ключей, то определение 6 примет следующий вид:

Определение. *Вторая нормальная форма*

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда оно находится в 1NF, и каждый неключевой атрибут полностью зависит от каждого ключа R.

Третья нормальная форма

Рассмотрим еще раз отношение СОТРУДНИКИ-ОТДЕЛЫ, находящееся в 2NF. Заметим, что функциональная зависимость СОТР_НОМЕР -> СОТР_ЗАРП является транзитивной; она является следствием функциональных зависимостей СОТР_НОМЕР -> ОТД_НОМЕР и ОТД_НОМЕР -> СОТР_ЗАРП. Другими словами, заработная плата сотрудника на самом деле является характеристикой не сотрудника, а отдела, в котором он работает (это не очень естественное предположение, но достаточное для примера).

В результате мы не сможем занести в базу данных информацию, характеризующую заработную плату отдела, до тех пор, пока в этом отделе не появится хотя бы один

сотрудник (первичный ключ не может содержать неопределенное значение). При удалении кортежа, описывающего последнего сотрудника данного отдела, мы лишимся информации о заработной плате отдела. Чтобы согласованным образом изменить заработную плату отдела, мы будем вынуждены предварительно найти все кортежи, описывающие сотрудников этого отдела. Т.е. в отношении СОТРУДИКИ-ОТДЕЛЫ по-прежнему существуют аномалии. Их можно устранить путем дальнейшей нормализации.

Определение. *Третья нормальная форма. (Снова определение дается в предположении существования единственного ключа.)*

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Можно произвести декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ в два отношения СОТРУДНИКИ и ОТДЕЛЫ:

СОТРУДНИКИ (СОТР_НОМЕР, ОТД_НОМЕР)

Первичный ключ:

СОТР_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР -> ОТД_НОМЕР

ОТДЕЛЫ (ОТД_НОМЕР, СОТР_ЗАРП)

Первичный ключ:

ОТД_НОМЕР

Функциональные зависимости:

ОТД_НОМЕР -> СОТР_ЗАРП

Каждое из этих двух отношений находится в 3NF и свободно от отмеченных аномалий.

Если отказаться от того ограничения, что отношение обладает единственным ключом, то определение 3NF примет следующую форму:

Определение. *Третья нормальная форма*

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 1NF, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается. Однако иногда полезно продолжить процесс нормализации.

Нормальная форма Бойса-Кодда

Рассмотрим следующий пример схемы отношения:

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, СОТР_ИМЯ, ПРО_НОМЕР, СОТР_ЗАДАН)

Возможные ключи:

СОТР_НОМЕР, ПРО_НОМЕР

СОТР_ИМЯ, ПРО_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР -> СОТР_ИМЯ

СОТР_НОМЕР -> ПРО_НОМЕР

СОТР_ИМЯ -> СОТР_НОМЕР

СОТР_ИМЯ -> ПРО_НОМЕР

СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН

СОТР_ИМЯ, ПРО_НОМЕР -> СОТР_ЗАДАН

В этом примере мы предполагаем, что личность сотрудника полностью определяется как его номером, так и именем (это снова не очень жизненное предположение, но достаточное для примера).

Отношение СОТРУДНИКИ-ПРОЕКТЫ находится в 3NF. Однако тот факт, что имеются функциональные зависимости атрибутов отношения от атрибута, являющегося частью первичного ключа, приводит к аномалиям. Например, для того, чтобы изменить имя сотрудника с данным номером согласованным образом, нам потребуется модифицировать все кортежи, включающие его номер.

Определение. *Нормальная форма Бойса-Кодда*

Отношение R находится в нормальной форме Бойса-Кодда (BCNF) в том и только в том случае, если каждый детерминант является возможным ключом.

Очевидно, что это требование не выполнено для отношения СОТРУДНИКИ-ПРОЕКТЫ. Можно произвести его декомпозицию к отношениям СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ:

СОТРУДНИКИ (СОТР_НОМЕР, СОТР_ИМЯ)

Возможные ключи:

СОТР_НОМЕР

СОТР_ИМЯ

Функциональные зависимости:

СОТР_НОМЕР -> СОТР_ИМЯ

СОТР_ИМЯ -> СОТР_НОМЕР

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Возможный ключ:

СОТР_НОМЕР, ПРО_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН

Возможна альтернативная декомпозиция, если выбрать за основу СОТР_ИМЯ. В обоих случаях получаемые отношения СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ находятся в BCNF, и им не свойственны отмеченные аномалии.

Четвертая нормальная форма

Рассмотрим пример следующей схемы отношения:

ПРОЕКТЫ (ПРО_НОМЕР,ПРО_СОТР, ПРО_ЗАДАН)

Отношение ПРОЕКТЫ содержит номера проектов, для каждого проекта список сотрудников, которые могут выполнять проект, и список заданий, предусматриваемых проектом. Сотрудники могут участвовать в нескольких проектах, и разные проекты могут включать одинаковые задания.

Каждый кортеж отношения связывает некоторый проект с сотрудником, участвующим в этом проекте, и заданием, который сотрудник выполняет в рамках данного проекта (мы предполагаем, что любой сотрудник, участвующий в проекте, выполняет все задания, предусмотренные этим проектом). По причине сформулированных выше условий единственным возможным ключем отношения является составной атрибут ПРО_НОМЕР, ПРО_СОТР, ПРО_ЗАДАН, и нет никаких других детерминантов. Следовательно, отношение ПРОЕКТЫ находится в BCNF. Но при этом оно обладает недостатками: если, например, некоторый сотрудник присоединяется к данному проекту, необходимо вставить в отношение ПРОЕКТЫ столько кортежей, сколько заданий в нем предусмотрено.

В отношении ПРОЕКТЫ существуют следующие две многозначные зависимости:

ПРО_НОМЕР -> -> ПРО_СОТР

ПРО_НОМЕР -> -> ПРО_ЗАДАН

Легко показать, что в общем случае в отношении R (A, B, C) существует многозначная зависимость R.A –> –> R.B в том и только в том случае, когда существует многозначная зависимость R.A –> –> R.C.

Дальнейшая нормализация отношений, подобных отношению ПРОЕКТЫ, основывается на следующей теореме:

Теорема Фейджина

Отношение R (A, B, C) можно спроецировать без потерь в отношения R1 (A, B) и R2 (A, C) в том и только в том случае, когда существует MVD A –> –> B | C.

Под проецированием без потерь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений.

Определение. *Четвертая нормальная форма*

Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости A –> –> B все остальные атрибуты R функционально зависят от A.

В нашем примере можно произвести декомпозицию отношения ПРОЕКТЫ в два отношения ПРОЕКТЫ-СОТРУДНИКИ и ПРОЕКТЫ-ЗАДАНИЯ:

ПРОЕКТЫ-СОТРУДНИКИ (ПРО_НОМЕР, ПРО_СОТР)

ПРОЕКТЫ-ЗАДАНИЯ (ПРО_НОМЕР, ПРО_ЗАДАН)

Оба эти отношения находятся в 4NF и свободны от отмеченных аномалий.

Пятая нормальная форма

Во всех рассмотренных до этого момента нормализациях производилась декомпозиция одного отношения в два. Иногда это сделать не удается, но возможна декомпозиция в большее число отношений, каждое из которых обладает лучшими свойствами.

Рассмотрим, например, отношение

СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ (СОТР_НОМЕР, ОТД_НОМЕР, ПРО_НОМЕР)

Предположим, что один и тот же сотрудник может работать в нескольких отделах и работать в каждом отделе над несколькими проектами. Первичным ключем этого отношения является полная совокупность его атрибутов, отсутствуют функциональные и многозначные зависимости.

Поэтому отношение находится в 4NF. Однако в нем могут существовать аномалии, которые можно устранить путем декомпозиции в три отношения.

Определение. *Пятая нормальная форма*

Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения - PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.

Введем следующие имена составных атрибутов:

СО = {СОТР_НОМЕР, ОТД_НОМЕР}

СП = {СОТР_НОМЕР, ПРО_НОМЕР}

ОП = {ОТД_НОМЕР, ПРО_НОМЕР}

Предположим, что в отношении СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ существует зависимость соединения:

* (СО, СП, ОП)

На примерах легко показать, что при вставках и удалениях кортежей могут возникнуть проблемы. Их можно устранить путем декомпозиции исходного отношения в три новых отношения:

СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, ОТД_НОМЕР)

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР)

ОТДЕЛЫ-ПРОЕКТЫ (ОТД_НОМЕР, ПРО_НОМЕР)

Пятая нормальная форма - это последняя нормальная форма, которую можно получить путем декомпозиции. Ее условия достаточно нетривиальны, и на практике 5NF не используется. Заметим, что зависимость соединения является обобщением как многозначной зависимости, так и функциональной зависимости.

Доменно-ключевая нормальная форма

Определение. *Доменно-ключевая нормальная форма*

Переменная отношения находится в ДКНФ тогда и только тогда, когда каждое наложенное на неё ограничение является логическим следствием ограничений доменов и ограничений ключей, наложенных на данную переменную отношения.

Ограничение домена – ограничение, предписывающее использовать для определённого атрибута значения только из некоторого заданного домена. Ограничение по своей сути является заданием перечня (или логического эквивалента перечня)

допустимых значений типа и объявлением о том, что указанный атрибут имеет данный тип.

Ограничение ключа – ограничение, утверждающее, что некоторый атрибут или комбинация атрибутов является потенциальным ключом.

Любая переменная отношения, находящаяся в ДКНФ, обязательно находится в 5НФ. Однако не любую переменную отношения можно привести к ДКНФ.

Шестая нормальная форма

Определение. *Шестая нормальная форма*

Переменная отношения находится в шестой нормальной форме тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ тогда и только тогда, когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6НФ, также находится и в 5НФ.

Введена К. Дейтом как обобщение пятой нормальной формы для хронологической базы данных.

Идея «декомпозиции до конца» выдвигалась до начала исследований в области хронологических данных, но не нашла поддержки. Однако для хронологических баз данных максимально возможная декомпозиция позволяет бороться с избыточностью и упрощает поддержание целостности базы данных.

Для хронологических баз данных определены U_операторы, которые распаковывают отношения по указанным атрибутам, выполняют соответствующую операцию и упаковывают полученный результат. В данном примере соединение проекций отношения должно производиться при помощи оператора U_JOIN.

Работники

Таб. №	**Время**	**Должность**	**Домашний адрес**
6575	[01-01-2000:10-02-2003]	слесарь	ул. Ленина, 10
6575	[11-02-2003:15-06-2006]	слесарь	ул. Советская, 22
6575	[16-06-2006:05-03-2009]	бригадир	ул. Советская, 22

Переменная отношения «Работники» не находится в 6НФ и может быть подвергнута декомпозиции на переменные отношения «Должности работников» и «Домашние адреса работников».

Должности работников

Таб. №	Время	Должность
6575	[01-01-2000:15-06-2006]	слесарь
6575	[16-06-2006:05-03-2009]	бригадир

Домашние адреса работников

Таб. №	Время	Домашний адрес
6575	[01-01-2000:10-02-2003]	ул. Ленина, 10
6575	[11-02-2003:05-03-2009]	ул. Советская, 22

Недостатки нормализации

Хотя наиболее удачные базы данных всегда в той или иной степени нормализованы, у нормализованной базы данных есть один существенный недостаток, связанный со снижением ее производительности. Чтобы понять, почему это происходит, необходимо знать, что при выполнении базой данных запросов или транзакций существенную роль начинают играть такие факторы, как использование центрального процессора и памяти, а также система ввода/вывода. Короче говоря, для обработки транзакций и запросов в случае нормализованной базы данных к центральному процессору, памяти и системе ввода/вывода предъявляются существенно большие требования, чем когда эта база данных ненормализована. Ведь для получения требуемой информации или обработки существующих данных нормализованная база данных должна сначала найти все необходимые таблицы, а затем объединить содержащуюся в них информацию.

Денормализация базы данных

Итак, денормализация базы данных — почему вообще она вам когда-нибудь может потребоваться? Ответ прост: единственной причиной денормализации базы данных является попытка улучшить ее производительность. Денормализованная база данных — это совсем не то же самое, что база данных, которая никогда не была нормализована. Денормализация базы данных представляет собой процесс, направленный на некоторое снижение уровня ее нормализации. Не забывайте, что нормализация базы данных на самом деле приводит к снижению ее производительности из-за частого выполнения операций, связанных с соединением таблиц. Процесс денормализации данных может включать в себя перепроектирование отдельных таблиц, а также дублирование информации с целью уменьшения числа таблиц, подлежащих объединению для поиска

необходимых данных; все эти меры приводят к снижению требований к системе ввода/вывода, а также уменьшают загрузку центрального процессора.

Денормализация — это процесс изменения структуры таблиц нормализованной базы данных, направленный на получение управляемой избыточности данных с целью повышения производительности системы.

Однако денормализация данных имеет и свои издержки. Повышенная избыточность информации, хранящейся в денормализованной базе данных, может улучшить ее производительность, но потребует от вас больших усилий при отслеживании связанных между собой данных. Написание приложений в этом случае становится более сложным делом, поскольку исходные данные были разбросаны по разным таблицам и их поиск может вызвать определенные затруднения. Кроме того, теперь больших усилий потребует и поддержка ссылочной целостности — ведь родственные данные были распределены среди большого числа таблиц. Как при нормализации данных, так и при их денормализации существует золотая середина; однако в любом случае от вас требуется глубокое понимание сущности реальных данных, а также особых требований, предъявляемых к деятельности вашей компании.

Варианты для выполнения лабораторной работы

Вариант	**Предметная область**
1	Научная лаборатория
2	Университет (учебный процесс)
3	Приемное отделение больницы
4	Диспетчерская МЧС
5	Аптека
6	Библиотека
7	Аэропорт (диспетчерская)
8	Таксопарк
9	Сотовая компания
10	Интернет-провайдер
11	Школа
12	Сеть ресторанов
13	Издательство журнала
14	Туроператор

15	Обработка результатов ЕГЭ
16	Железная дорога (продажа билетов)
17	Промышленное предприятие (сбыт продукции)
18	Промышленное предприятие (отдел кадров)
19	Промышленное предприятие (сборочное производство)
20	Промышленное предприятие (поставка материалов)

Тема 6: SQL. Таблицы

Задание лабораторной работы

Цель работы: Получение практических навыков работы с СУБД и языком SQL (создание и изменения таблиц).

Задание:

1) В созданной на предыдущей лабораторной работе базе данных дополните таблицы ограничениями CHECK, DEFAULT, NOT NULL, UNIQUE, PRIMARY KEY, FOREIGN KEY;

2) внести изменения в схему базы, используя операторы ALTER TABLE; и DROP TABLE;

3) создайте новую таблицу (не менее трех полей);

4) добавьте в нее новый столбец;

5) удалите второй столбец из новой таблицы;

6) удалите все таблицу;

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание
2. Коды операций
3. Принтскриты всех выполненных операторов

Методические указания по выполнению лабораторной работы

Оператор создания таблицы CREATE TABLE

Для создания таблицы нужно определить:

1) Название таблицы
2) Тип таблицы
3) Определить поля (структуру)
    a. Название полей
    b. тип полей

c. ограничения, наложенные на поля

4) Ограничения на таблицу

1. *Имя таблицы*

ИМЯ_СХЕМЫ.ИМЯ_ТАБЛИЦЫ

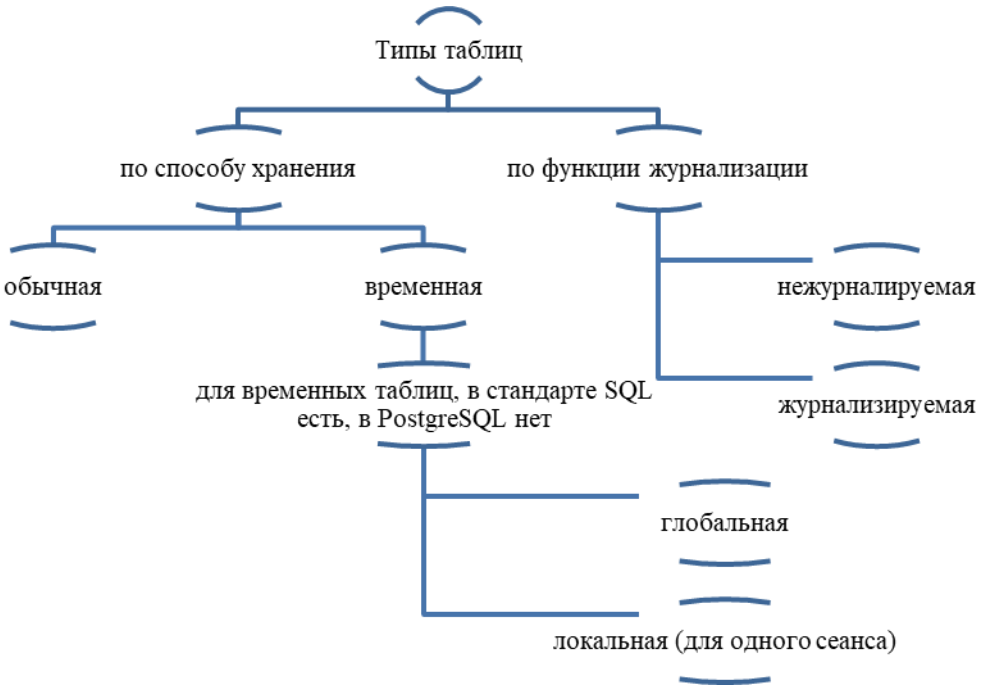
ИМЯ_СХЕМЫ – необязательно.

Если задано имя схемы (например, CREATE TABLE myschema.mytable ...), таблица создаётся в указанной схеме, в противном случае — в текущей.

Временные таблицы существуют в специальной схеме, так что при создании таких таблиц имя схемы задать нельзя. Имя таблицы должно отличаться от имён других таблиц, последовательностей, индексов, представлений или сторонних таблиц в этой схеме.

CREATE TABLE также автоматически создаёт составной тип данных, соответствующий одной строке таблицы. Таким образом, имя таблицы не может совпадать с именем существующего типа в этой же схеме.

2. *Типы таблицы*



TEMPORARY или TEMP

С таким указанием таблица создаётся как временная. Временные таблицы автоматически удаляются в конце сеанса или могут удаляться в конце текущей транзакции (см. описание ON COMMIT ниже). Существующая постоянная таблица с тем же именем не будет видна в текущем сеансе, пока существует временная, однако к ней можно обратиться, дополнив имя указанием схемы. Все индексы,

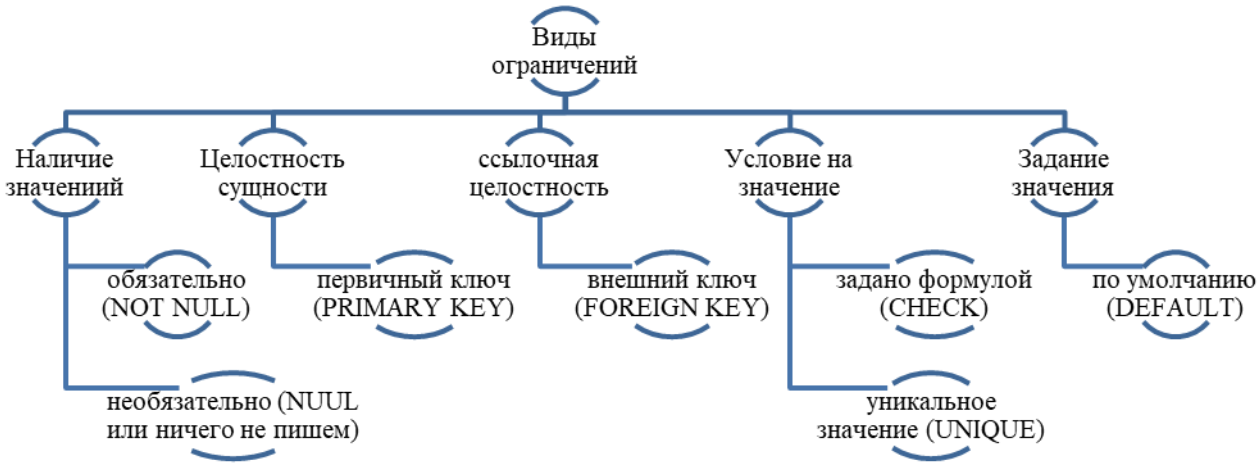
создаваемые для временной таблицы, так же автоматически становятся временными.

UNLOGGED

С этим указанием таблица создаётся как нежурналируемая. Данные, записываемые в нежурналируемые таблицы, не проходят через журнал предзаписи, в результате чего такие таблицы работают гораздо быстрее обычных. Однако, они не защищены от сбояКроме того, содержимое нежурналируемой таблицы не реплицируется на ведомые серверы. Любые индексы, создаваемые для нежурналируемой таблицы, автоматически становятся нежурналируемыми.

3. ***Определение поля***
   a. Имя
   b. Тип
   c. Ограничения

Ограничения столбца необязательны, описывается сразу после определения типа и применяются только к этому столбцу. После описания ограничений ставится запятая и определяется другой столбец (или переходим к ограничениям таблицы или завершаем описания таблицы и вместо запятой ставим скобку).



Ограничение PRIMARY KEY определяет, что столбец или столбцы таблицы могут содержать только уникальные (без повторений) значения, отличные от NULL. Для

таблицы может быть задан только один первичный ключ, будь то ограничение столбца или ограничение таблицы.

В определении первичного ключа должен задаваться набор столбцов, отличный от набора любого другого ограничения уникальности, установленного для данной таблицы. (В противном случае уникальное ограничение оказывается избыточным и будет отброшено.)

PRIMARY KEY устанавливает для данных те же ограничения, что и сочетание UNIQUE и NOT NULL, но образование первичного ключа из набора столбцов также добавляет метаданные о конструкции схемы, так как первичный ключ подразумевает, что другие таблицы могут ссылаться на этот набор столбцов, как на уникальный идентификатор строк.

Ограничение внешнего ключа, требующее, чтобы указанная группа из одного или нескольких столбцов новой таблицы содержала только такие значения, которым соответствуют значения в заданных внешних столбцах некоторой строки во внешней таблице. Если список внешних_столбцов опущен, в качестве него используется первичный ключ внешней_таблицы.

Внешний ключ можно создать только, если родительская таблица существует.

В ограничении CHECK задаётся выражение, возвращающее булевский результат, по которому определяется, будет ли успешна операция добавления или изменения для конкретных строк. Операция выполняется успешно, если результат выражения равен TRUE или UNKNOWN. Если же для какой-нибудь строки, задействованной в операции добавления или изменения, будет получен результат FALSE, возникает ошибка, и эта операция не меняет ничего в базе данных. Ограничение-проверка, заданное как ограничение столбца, должно ссылаться только на значение самого столбца, тогда как ограничение на уровне таблицы может ссылаться и на несколько столбцов.

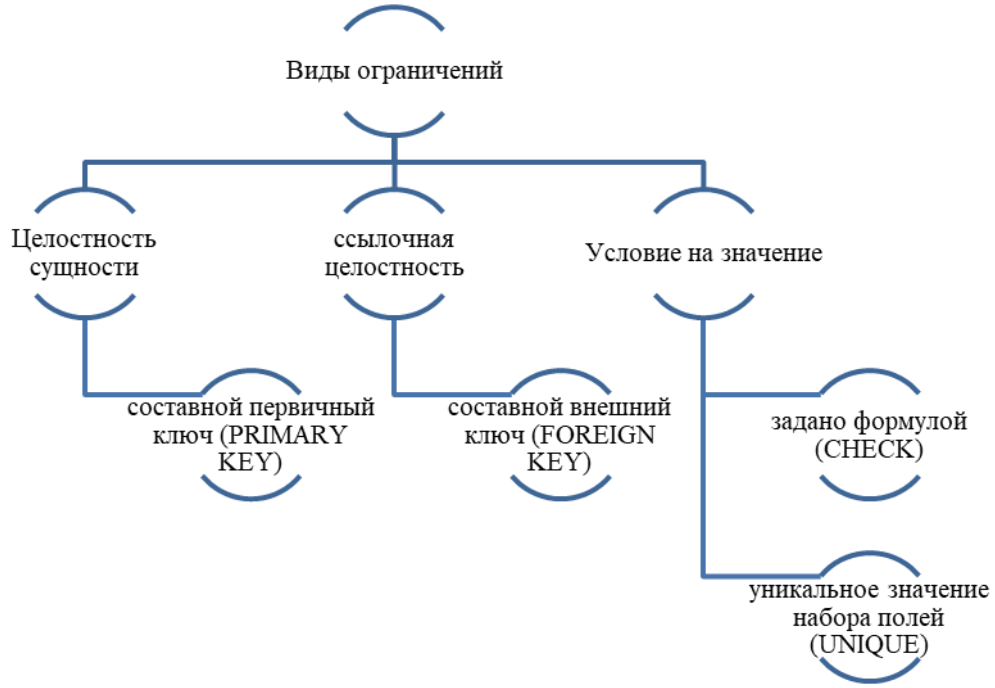
В настоящее время выражения CHECK не могут содержать подзапросы или ссылаться на переменные, кроме как на столбцы текущей строки.

Предложение DEFAULT задаёт значение по умолчанию для столбца, в определении которого оно присутствует. Значение задаётся выражением без переменных (подзапросы и перекрёстные ссылки на другие столбцы текущей таблицы в нём не допускаются). Тип данных выражения, задающего значение по умолчанию, должен соответствовать типу данных столбца.

Это выражение будет использоваться во всех операциях добавления данных, в которых не задаётся значение данного столбца. Если значение по умолчанию не определено, таким значением будет NULL.

4. *Ограничения таблицы*

Ограничения таблицы не обязательны, описываются после описания столбцов и могут распространяться на несколько столбцов таблицы. Например, если нужно, чтобы значения одного столбца всегда было больше другого, через ограничение столбца это сделать нельзя, так как там можно обращаться только к одному столбцу, а в ограничении таблицы это можно задать.



Синтаксис (упрощенный, полный см. в документации)

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ]
TABLE имя_таблицы ( [
  { имя_столбца тип_данных [ COLLATE правило_сортировки ] [
ограничение_столбца [ ... ] ]
    | ограничение_таблицы
}
[, ... ]
)
```

Здесь ограничение_столбца:

```
[ CONSTRAINT имя_ограничения ]
{ NOT NULL |
```

```
  NULL |
  CHECK ( выражение ) |
  DEFAULT выражение_по_умолчанию |
  UNIQUE параметры_индекса |
  PRIMARY KEY параметры_индекса |
  REFERENCES целевая_таблица [ ( целевой_столбец ) ] }
```

и ограничение_таблицы:

```
[ CONSTRAINT имя_ограничения ]
{ CHECK ( выражение ) |
  UNIQUE ( имя_столбца [, ... ] ) |
  PRIMARY KEY ( имя_столбца [, ... ] ) |
  FOREIGN KEY ( имя_столбца [, ... ] ) REFERENCES
целевая_таблица [ ( целевой_столбец [, ... ] ) ]
}
```

Примеры

Определение ограничения первичного ключа для таблицы distributors. Следующие два примера равнозначны, но в первом используется синтаксис ограничений для таблицы, а во втором — для столбца:

```
CREATE TABLE distributors (
    did     integer,
    name    varchar(40),
    PRIMARY KEY(did)
);

CREATE TABLE distributors (
    did     integer PRIMARY KEY,
    name    varchar(40)
);
```

Определение ограничения-проверки для столбца:

```
CREATE TABLE distributors (
    did     integer CHECK (did > 100),
    name    varchar(40)
```

```
);
```

Определение ограничения-проверки для таблицы:

```
CREATE TABLE distributors (
    did     integer,
    name    varchar(40),
    CONSTRAINT con1 CHECK (did > 100 AND name <> '')
);
```

Определение значений по умолчанию: для столбца name значением по умолчанию будет строка, для столбца did — следующее значение объекта последовательности, а для modtime — время, когда была вставлена запись:

```
CREATE TABLE distributors (
    name      varchar(40) DEFAULT 'Luso Films',
    did       integer DEFAULT nextval('distributors_serial'),
    modtime   timestamp DEFAULT current_timestamp
);
```

Удаление таблицы DROP TABLE

Drop table удаляет саму таблицу вместе с данными. Если нудно удалить только данные из таблицы, то воспользуйтесь операторами DELETE или TRUNCATE TABLE.

`DROP TABLE` всегда удаляет все индексы, правила, триггеры и ограничения, существующие в целевой таблице. Однако, чтобы удалить таблицу, на которую ссылается представление или ограничение внешнего ключа в другой таблице, необходимо дополнительно указать `CASCADE`. (С указанием `CASCADE` зависимое представление будет удалено полностью, но в случае с первичным ключом удалено будет только само ограничение, а не таблица, к которой оно относится.)

Синтаксис

DROP TABLE *имя* [, ...] [CASCADE | RESTRICT]

CASCADE

Автоматически удалять объекты, зависящие от данной таблицы (например, представления), и, в свою очередь, все зависящие от них объекты.

RESTRICT

Отказать в удалении таблицы, если от неё зависят какие-либо объекты. Это поведение по умолчанию.

Пример

DROP TABLE films, distributors;

Изменение таблицы оператор ALTER TABLE

Синтаксис

ALTER TABLE *имя действие*

ALTER TABLE *имя*

RENAME [COLUMN] *имя_столбца* TO *новое_имя_столбца*

ALTER TABLE *имя*

RENAME CONSTRAINT *имя_ограничения* TO *имя_нового_ограничения*

ALTER TABLE *имя*

RENAME TO *новое_имя*

ALTER TABLE *имя*

SET SCHEMA *новая_схема*

...

Действие может быть следующим (не полный список, схему см ниже):

ADD [COLUMN] *имя_столбца тип_данных* [*ограничение_столбца* [...]]

DROP [COLUMN] *имя_столбца*

ALTER [COLUMN] *имя_столбца* [SET DATA] TYPE *тип_данных*

ALTER [COLUMN] *имя_столбца* SET DEFAULT *выражение*

ALTER [COLUMN] *имя_столбца* DROP DEFAULT

ALTER [COLUMN] *имя_столбца* { SET | DROP } NOT NULL

DROP CONSTRAINT *имя_ограничения*

DISABLE TRIGGER [*имя_триггера* | ALL | USER]

ENABLE TRIGGER [*имя_триггера* | ALL | USER]

Добавление столбца с предложением DEFAULT или изменение типа существующего столбца влечёт за собой перезапись всей таблицы и её индексов.

Добавление ограничений CHECK или NOT NULL влечёт за собой необходимость просканировать таблицу, чтобы проверить, что все существующие строки удовлетворяют ограничению, но перезаписывать таблицу при этом не требуется.

Форма DROP COLUMN не удаляет столбец физически, а просто делает его невидимым для операций SQL. При последующих операциях добавления или изменения в этот столбец будет записываться значение NULL. Таким образом, удаление столбца выполняется быстро, но при этом размер таблицы на диске не уменьшается, так как пространство, занимаемое удалённым столбцом, не высвобождается. Это пространство будет освобождено со временем, по мере изменения существующих строк.

Какие-либо изменения таблиц системного каталога не допускаются.

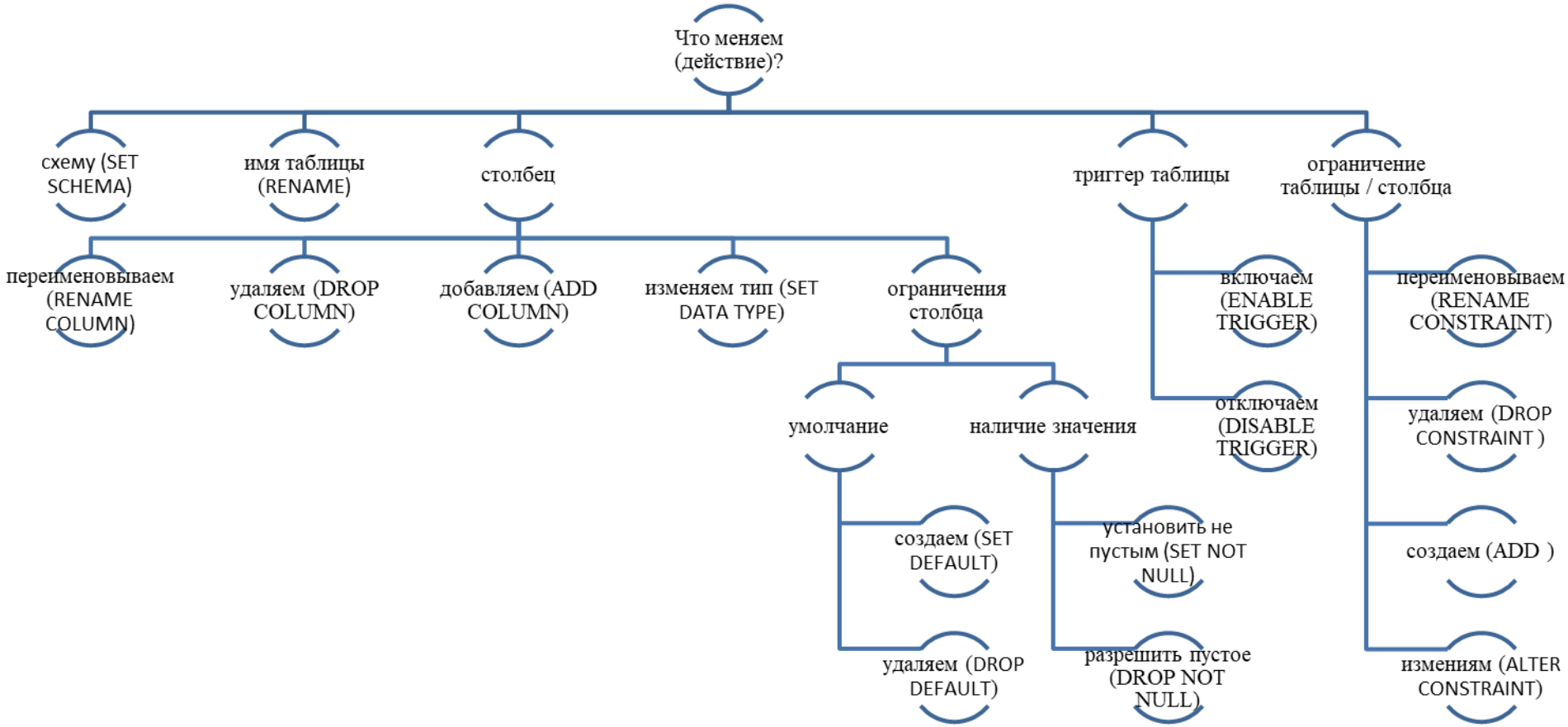


Рисунок 1 – Схема основных возможностей (схема синтаксиса оператора) оператора ALTER TABLE (не все, остальные см. в документации)

Примеры

Добавление в таблицу столбца типа `varchar`:

```
ALTER TABLE distributors ADD COLUMN address varchar(30);
```

Удаление столбца из таблицы:

```
ALTER TABLE distributors DROP COLUMN address RESTRICT;
```

Изменение типов двух существующих столбцов в одной операции:

```
ALTER TABLE distributors
    ALTER COLUMN address TYPE varchar(80),
    ALTER COLUMN name TYPE varchar(100);
```

Переименование существующего столбца:

```
ALTER TABLE distributors RENAME COLUMN address TO city;
```

Переименование существующей таблицы:

```
ALTER TABLE distributors RENAME TO suppliers;
```

Переименование существующего ограничения:

```
ALTER TABLE distributors RENAME CONSTRAINT zipchk TO zip_check;
```

Добавление в столбец ограничения NOT NULL:

```
ALTER TABLE distributors ALTER COLUMN street SET NOT NULL;
```

Удаление ограничения NOT NULL из столбца:

```
ALTER TABLE distributors ALTER COLUMN street DROP NOT NULL;
```

Добавление ограничения-проверки в таблицу и все её потомки:

```
ALTER TABLE distributors ADD CONSTRAINT zipchk CHECK
(char_length(zipcode) = 5);
```

Удаление ограничения-проверки из таблицы и из всех её потомков:

```
ALTER TABLE distributors DROP CONSTRAINT zipchk;
```

Добавление в таблицу ограничения внешнего ключа:

```
ALTER TABLE distributors ADD CONSTRAINT distfk FOREIGN KEY (address) REFERENCES addresses (address);
```

Добавление в таблицу ограничения внешнего ключа с наименьшим влиянием на работу других:

```
ALTER TABLE distributors ADD CONSTRAINT distfk FOREIGN KEY (address) REFERENCES addresses (address) NOT VALID;
ALTER TABLE distributors VALIDATE CONSTRAINT distfk;
```

Добавление в таблицу ограничения уникальности (по нескольким столбцам):

```
ALTER TABLE distributors ADD CONSTRAINT dist_id_zipcode_key UNIQUE (dist_id, zipcode);
```

Добавление в таблицу первичного ключа с автоматическим именем (учтите, что в таблице может быть только один первичный ключ):

```
ALTER TABLE distributors ADD PRIMARY KEY (dist_id);
```

Тема 7: SQL. Операторы модификации данных

Задание лабораторной работы

Цель работы: Получение практических навыков работы с СУБД и языком SQL (операторы insert, update, delete, truncate).

Задание:

1) внести данные с таблицы, созданные на предыдущих лабораторных работах, используя оператор INSERT (не менее 3 строк у каждую таблицу);
2) изменить данные в таблицах, используя оператор UPDATE (не менее 3 изменений);
3) внесите данные в одну из таблиц из другой таблицы (если нет подходящих данных создайте дополнительную таблицу и нанесите данные в нее);
4) удалить часть данных из заполненной таблицы, используя оператор DELETE;
5) удалить оставшуюся часть данных с просмотром удаленных полей, используя оператор DELETE и инструкцию RETURNING;
6) удалите данные из другой таблицы, использую TRUNCATE;

7) восстановите данные в таблицах, используя свои коды из пункта 1 (чтобы для следующей лабораторной работы таблицы были заполнены).

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание.
2. Коды операций.
3. Принтскриты всех выполненных операторов.

Методические указания по выполнению лабораторной работы

Вставка данных

– Insert (вставка конкретных значений данных, данных из других таблиц)

– Copy (вставка данным из файла/ в файл)

Оператор Insert

Синтаксис

INSERT INTO *<имя таблицы>*[(*<имя столбца>*,...)]
{VALUES (*< значение столбца>*,…)}
| *<выражение запроса>*
| {DEFAULT VALUES};

Примеры

Вставляемые данные	Пример
Целое число	5 Пример: INSERT INTO products VALUES (***1***, 'Cheese', 9.99);
Вещественное	9.99 Пример: INSERT INTO products VALUES (1, 'Cheese', ***9.99***);
Символьное	'Cheese' Пример: INSERT INTO products VALUES (1, ***'Cheese'***, 9.99);
Дата	'12.12.2012' Пример:
Деньги	2.99 Пример: INSERT INTO products (product_no, name, price) VALUES (3, 'Milk', 2.99);
Пустое	NULL / опускаем значение (если нет умолчания) Пример: INSERT INTO products (product_no, name, price) VALUES

	(3, 'Milk', NULL); INSERT INTO products (product_no, name) VALUES (3, 'Milk');
По умолчанию	DEFAULT / опускаем значение (умолчание должно быть определено) Пример: INSERT INTO products (product_no, name, price) VALUES (1, 'Cheese', DEFAULT); --значения по умолчанию для всей строки INSERT INTO products DEFAULT VALUES;
Одна строка	Пример: INSERT INTO products VALUES (1, 'Cheese', 9.99);
Несколько строк	Пример: INSERT INTO products (product_no, name, price) VALUES (1, 'Cheese', 9.99), (2, 'Bread', 1.99), (3, 'Milk', 2.99);
Строки из других таблиц	INSERT INTO products (product_no, name, price) SELECT product_no, name, price FROM new_products WHERE release_date = 'today';

Оператор COPY

COPY — копировать данные между файлом и таблицей

COPY перемещает данные между таблицами PostgreSQL и обычными файлами в файловой системе. COPY TO копирует содержимое таблицы *в* файл, а COPY FROM — *из* файла в таблицу (добавляет данные к тем, что уже содержались в таблице). COPY TO может также скопировать результаты запроса SELECT.

Если указывается список столбцов, COPY скопирует только данные указанных столбцов. Если в таблице есть столбцы, отсутствующие в этом списке, COPY FROM заполнит эти столбцы значениями по умолчанию.

Синтаксис

```
COPY имя_таблицы [ ( имя_столбца [, ...] ) ]
    FROM { 'имя_файла' | PROGRAM 'команда' | STDIN }
    [ [ WITH ] ( параметр [, ...] ) ]

COPY { имя_таблицы [ ( имя_столбца [, ...] ) ] | ( запрос ) }
    TO { 'имя_файла' | PROGRAM 'команда' | STDOUT }
    [ [ WITH ] ( параметр [, ...] ) ]
Здесь допускается параметр:
    FORMAT имя_формата
    OIDS [ boolean ]
    FREEZE [ boolean ]
    DELIMITER 'символ_разделитель'
    NULL 'маркер_NULL'
    HEADER [ boolean ]
    QUOTE 'символ_кавычек'
    ESCAPE 'символ_экранирования'
    FORCE_QUOTE { ( имя_столбца [, ...] ) | * }
    FORCE_NOT_NULL ( имя_столбца [, ...] )
    FORCE_NULL ( имя_столбца [, ...] )
    ENCODING 'имя_кодировки'
```

Файл может быть:

– Текстовый
– Двоичный
– Формат CSV

Команда `COPY FROM` распознаёт следующие спецпоследовательности:

Последовательность	Представляет
\b	Забой (ASCII 8)
\f	Подача формы (ASCII 12)
\n	Новая строка (ASCII 10)
\r	Возврат каретки (ASCII 13)
\t	Табуляция (ASCII 9)
\v	Вертикальная табуляция (ASCII 11)
\цифры	Обратная косая с последующими 1-3 восьмеричными цифрами представляет символ с заданным числовым кодом
\xцифры	Обратная косая с последующим x и 1-2 шестнадцатеричными цифрами представляет символ с заданным числовым кодом

Примеры

Копирование данных из файла в таблицу	COPY country FROM '/usr1/proj/bray/sql/country_data';
Копирование в файл результатов запроса	COPY (SELECT * FROM country WHERE country_name LIKE 'A%') TO '/usr1/proj/bray/sql/a_list_countries.copy';
Сжатие файла после копирования	COPY country TO PROGRAM 'gzip > /usr1/proj/bray/sql/country_data.gz';

Модификация данных

– Update (изменение значений уже существующих строк)

– Megre (слияние, в PostgreSQL не поддерживается)

Оператор Update

Оператор **UPDATE** изменяет имеющиеся данные в таблице.

Синтаксис

UPDATE <имя таблицы>
SET {имя столбца = {выражение для вычисления значения столбца
| NULL
| DEFAULT},...}
[{WHERE <предикат>}];

С помощью одного оператора могут быть заданы значения для любого количества столбцов. Однако в одном и том же операторе **UPDATE** можно вносить изменения в каждый столбец указанной таблицы только один раз. При отсутствии предложения **WHERE** будут обновлены все строки таблицы.

Если столбец допускает NULL-значение, то его можно указать в явном виде. Кроме того, можно заменить имеющееся значение на значение по умолчанию (**DEFAULT**) для данного столбца.

Примеры

Новые значения	
Пустое значение (во всех строках таблицы)	UPDATE Laptop SET hd =NULL
Простое значение	UPDATE Laptop SET price=1000.9
Значение по умолчанию	UPDATE Laptop SET price=DEFAULT
У одного столбца в строках, удовлетворяющих условию	UPDATE Laptop SET price=price*0.9 where PC LIKE '%Pentium%'
У нескольких столбцов с строках, удовлетворяющих условию	UPDATE Laptop SET price=price*0.9 , speed =NULL where PC LIKE '%Pentium%' UPDATE Laptop SET hd=ram/2 WHERE hd < 10
Значение по условию	Пример: UPDATE Laptop SET hd = CASE WHEN ram<128 THEN 20 ELSE 40 END
Значение подзапроса	Пример: UPDATE Laptop SET speed = (SELECT MAX(speed) FROM Laptop)

Удаление данных

– Delete (удаление данных их таблицы с возможность выбора и проверкой ссылочной целостности)

– Truncate (очистка таблицы от всех данных)

Оператор Delete

Команда DELETE удаляет из указанной таблицы строки, удовлетворяющие условию WHERE. Если предложение WHERE отсутствует, она удаляет из таблицы все строки, в результате будет получена рабочая, но пустая таблица.

Синтаксис

DELETE FROM <имя таблицы > [WHERE <предикат>];

Примеры

Удаление строк по условию	DELETE FROM Laptop WHERE screen < 12;
Удаление всех строк с учетом целостности	DELETE FROM Laptop
Удаление текущей строки через курсоры (см. тему курсоры)	DELETE FROM tasks WHERE CURRENT OF c_tasks;

Оператор TRUNCATE TABLE

Команда TRUNCATE быстро удаляет все строки из набора таблиц. Она действует так же, как безусловная команда DELETE для каждой таблицы, но гораздо быстрее, так как она фактически не сканирует таблицы. Более того, она немедленно высвобождает дисковое пространство,

Синтаксис

```
TRUNCATE [ TABLE ] [ ONLY ] имя [ * ] [, ... ]
    [ RESTART IDENTITY | CONTINUE IDENTITY ] [ CASCADE | RESTRICT ]
```

`RESTART IDENTITY`

Автоматически перезапускать последовательности, связанные со столбцами опустошаемой таблицы.

`CONTINUE IDENTITY`

Не изменять значения последовательностей. Это поведение по умолчанию.

`CASCADE`

Автоматически опустошать все таблицы, ссылающиеся по внешнему ключу на заданные таблицы, или на таблицы, затронутые в результате действия `CASCADE`.

`RESTRICT`

Отказать в опустошении любых таблиц, на которые по внешнему ключу ссылаются другие таблицы, не перечисленные в этой команде. Это поведение по умолчанию.

Отличий в реализации команды **TRUNCATE TABLE** от оператора **DELETE**:

1. Не журнализируется удаление отдельных строк таблицы. В журнал записывается только освобождение страниц, которые были заняты данными таблицы.
2. Не отрабатывают триггеры ON DELETE.
3. Нельзя использовать с таблицей, на которую по внешнему ключу ссылаются другие таблицы, если только и эти таблицы не опустошаются этой же командой.

Примеры

Удаление всех строк без учета целостности	TRUNCATE TABLE Laptop
Очистка нескольких таблиц	TRUNCATE bigtable, fattable;
Очистка связанных таблиц	TRUNCATE othertable CASCADE;

Тема 8: Основы SQL. Запросы

Задание лабораторной работы

Цель работы: Получение практических навыков работы с СУБД и языком SQL (оператор SELECT).

Задание:

1) разработать запросы к базе данных, созданной и заполненной на предыдущих лабораторных работах, следующих видов:

a. запрос с условием на числовые данные (>,<,=, between);
b. запрос с условием на текстовые данные (LIKE, IN);
c. запрос с вычисляемым полем;
d. запрос к нескольким таблицам (без явного указания JOIN);
e. запрос с агрегирующей функцией (AVG, SUM, COUNT, MIN, MAX);
f. запрос с группировкой (GROUP BY);
g. запрос с сортировкой (ORDER BY);
h. запрос с вложенным подзапросом (не менее 3 видов);
i. запрос с оператором UNION;
j. запрос с оператором INTERSECT;
k. запрос с оператором EXCEPT;
l. запрос с выражением CASE;
m. запрос с оператором JOIN (пять видов);
n. иерархический запрос.

2) Для каждого запроса подписать, что именно он возвращает с учетом предметной области (запросы со смыслом, а не только синтаксически правильные операторы).

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание.
2. Коды операций.
3. Принтскриты всех выполненных операторов.

Методические указания по выполнению лабораторной работы

Для формирования запросов на выборку данных в SQL используется оператор SELECT.

Синтаксис

```
SELECT [ ALL | DISTINCT]
```

```
[ * | выражение [ [ AS ] имя_результата ] [, ...] ]
[ FROM элемент_FROM [, ...] ]
[ WHERE условие ]
[ GROUP BY элемент_группирования [, ...] ]
[ HAVING условие [, ...] ]
[ ORDER BY выражение [ ASC | DESC ] ]
[ LIMIT { число | ALL } ]
```

Только два предложения SELECT и FROM являются обязательными.

- **FROM** – определяются имена используемых таблиц;
- **WHERE** – выполняется фильтрация строк объекта в соответствии с заданными условиями;
- **GROUP BY** – образуются группы строк, имеющих одно и то же значение в указанном столбце;
- **HAVING** – фильтруются группы строк объекта в соответствии с указанным условием;
- **SELECT** – устанавливается, какие столбцы должны присутствовать в выходных данных;
- **ORDER BY** – определяется упорядоченность результатов выполнения операторов.

Если используется имя поля, содержащее пробелы или разделители, его следует заключить в квадратные скобки.

`SELECT` получает строки из множества таблиц (возможно, пустого). Общая процедура выполнения `SELECT` следующая:

1. Вычисляются все элементы в списке `FROM`. (Каждый элемент в списке `FROM` представляет собой реальную или виртуальную таблицу.) Если список `FROM` содержит несколько элементов, они объединяются перекрёстным соединением. (INNER JOIN)
2. Если указано предложение `WHERE`, все строки, не удовлетворяющие условию, исключаются из результата.
3. Если присутствует указание `GROUP BY`, либо в запросе вызываются агрегатные функции, вывод разделяется по группам строк, соответствующим одному или нескольким значениям, а затем вычисляются результаты агрегатных функций. Если

добавлено предложение HAVING, оно исключает группы, не удовлетворяющие заданному условию.

4. Вычисляются фактические выходные строки по заданным в SELECT выражениям для каждой выбранной строки или группы строк.
5. SELECT DISTINCT исключает из результата повторяющиеся строки. SELECT DISTINCT ON исключает строки, совпадающие по всем указанным выражениям. SELECT ALL (по умолчанию) возвращает все строки результата, включая дубликаты.
6. Если присутствует предложение ORDER BY, возвращаемые строки сортируются в указанном порядке. В отсутствие ORDER BY строки возвращаются в том порядке, в каком системе будет проще их выдать.

Примеры:

Все столбцы из таблицы:

```
SELECT * FROM t1;
```

Конкретные столбцы из таблицы:

```
SELECT p1, p3 FROM t1;
```

Запрос с условием:

```
SELECT * FROM t1 WHERE p2=4;
```

Запрос с сортировкой:

```
SELECT p1,p3 FROM t1 order by p1;
```

Запрос с группировкой:

```
SELECT p1,count(*) FROM t1 group by p1;
```

Запрос с группировкой и условием на группу;

```
SELECT p1,count(*) FROM t1 group by p1 having max(p1)>3;
```

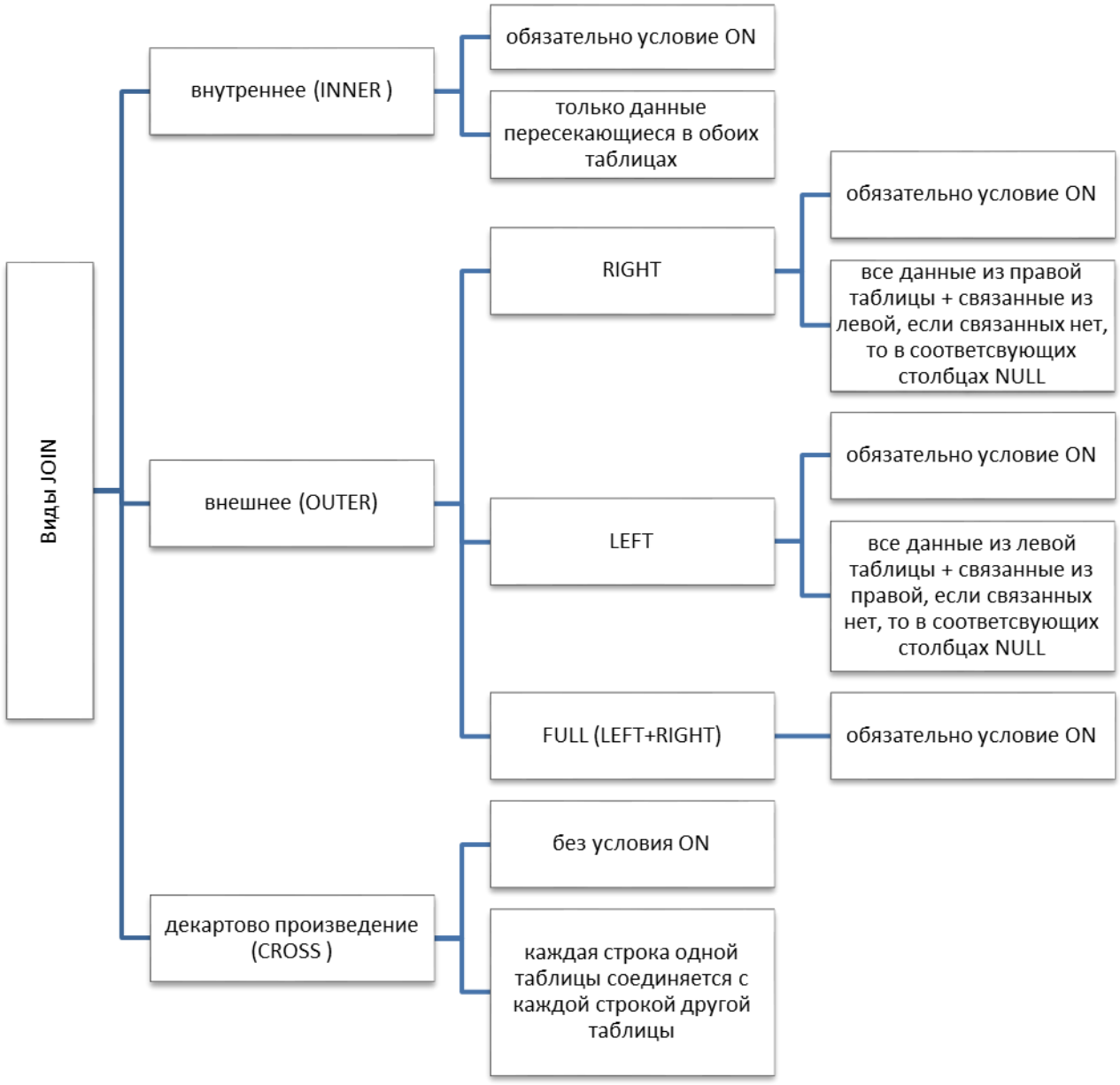
Язык SQL предоставляет два способа объединения таблиц:

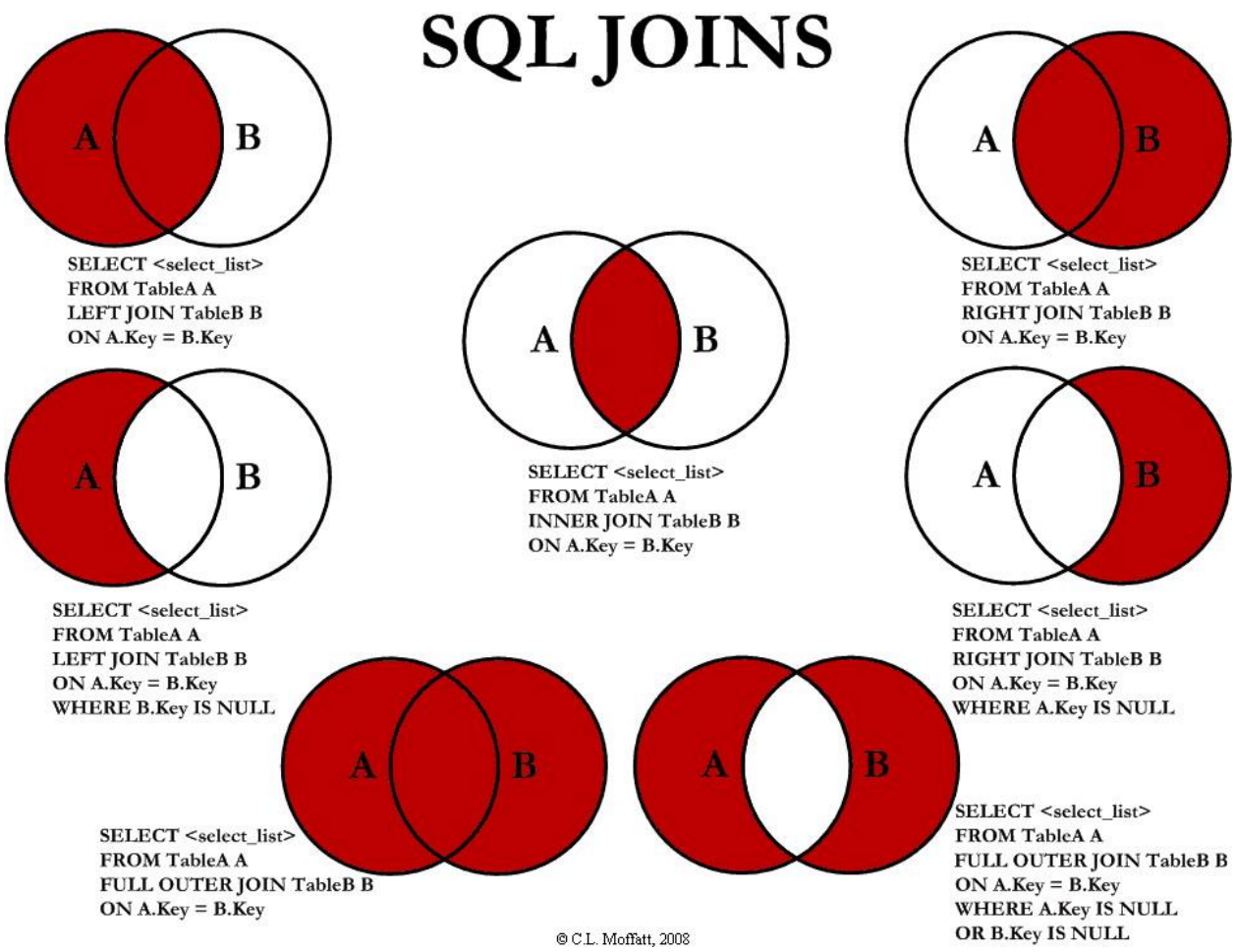
- указывая соединяемые таблицы (в том числе подзапросы) во фразе FROM оператора SELECT. Сначала выполняется соединение таблиц, а уже потом к полученному множеству применяются указанные фразой WHERE условия, определяемое фразой GROUP BY агрегирование, упорядочивание данных и т.п.;

- определяя объединение результирующих наборов, полученных при обработке оператора SELECT. В этом случае два оператора SELECT соединяются фразой UNION , INTERSECT или EXCEPT.

Соединение таблиц

Соединение таблиц (JOIN) позволяет получать в одном наборе данных данные из разных таблиц, соединённых по определенным условиям (данные соответствуют друг другу).





Объединение результатов запросов

Операторы UNION, INTERSECT и EXCEPT объединяют вывод нескольких команд SELECT в один результирующий набор. Оператор UNION возвращает все строки, представленные в одном, либо обоих наборах результатов. Оператор INTERSECT возвращает все строки, представленные строго в обоих наборах. Оператор EXCEPT возвращает все строки, представленные в первом наборе, но не во втором. Во всех трёх случаях повторяющиеся строки исключаются из результата, если явно не указано ALL. Чтобы явно обозначить, что выдаваться должны только неповторяющиеся строки, можно добавить избыточное слово DISTINCT. Заметьте, что в данном контексте по умолчанию подразумевается DISTINCT, хотя в самом SELECT по умолчанию подразумевается ALL.

Условия объединения операторов select:

- каждый из объединяемых запросов должен содержать одинаковое число столбцов;
- тип значений из попарно объединяемых столбцов должен быть одинаковым или приводимым. Так, нельзя объединять значения из столбца типа integer и столбца типа varchar;

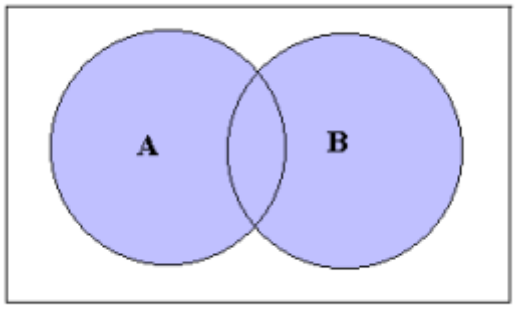
Синтаксис:

запрос1 UNION [ALL] *запрос2*

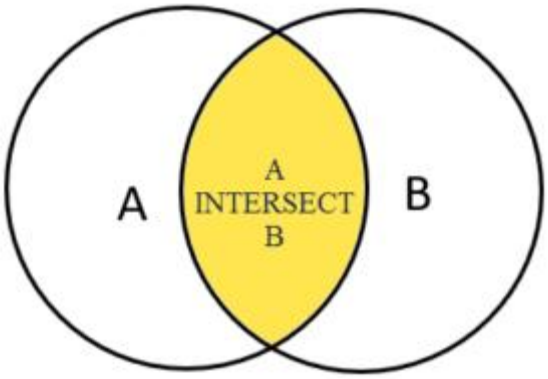
запрос1 INTERSECT [ALL] *запрос2*

запрос1 EXCEPT [ALL] *запрос2*

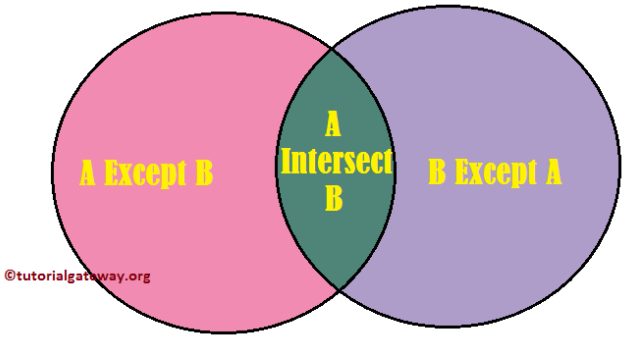
UNION добавляет результаты второго запроса к результатам первого (хотя никакой порядок возвращаемых строк при этом не гарантируется). Более того, эта операция убирает дублирующиеся строки из результата так же, как это делает DISTINCT, если только не указано UNION ALL.



INTERSECT возвращает все строки, содержащиеся в результате и первого, и второго запроса. Дублирующиеся строки отфильтровываются, если не указано ALL.



EXCEPT возвращает все строки, которые есть в результате первого запроса, но отсутствуют в результате второго. (Иногда это называют разницей двух запросов.) И здесь дублирующиеся строки отфильтровываются, если не указано ALL.



Тема 9: SQL. Индексы, просмотры

Задание лабораторной работы

Цель работы: Получение практических навыков работы с СУБД и языком SQL (операторы create index, create view, alter view, drop index, drop view).

Задание:

1) Разработать представления к базе данных, созданной и заполненной на предыдущих лабораторных работах, следующих видов:
    a. простое нематериализованное;
    b. материализованное неизменяемое;
    c. простое изменяемое (невозможно изменить неотображаемые в представлении строки);
    d. простое изменяемое (можно изменить неотображаемые в представлении строки).
2) Выполнить изменение данных в базовых таблицах через изменяемые преставления (три разных оператора модификации).
3) Обновить данные в материализованном представлении.
4) Разработать индексы к базе данных, созданной и заполненной на предыдущих лабораторных работах, следующих видов:
    a. простой в целой таблице;
    b. составной частичный к таблице;
    c. уникальный к материализованному представлению;
    d. с заданной сортировкой составной к таблице.
5) Переименовать одно из представлений.
6) Удалить один из индексов и одно представление.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание.
2. Коды операций.
3. Принтскриты всех выполненных операторов.

Методические указания по выполнению лабораторной работы

Представления – объекты базы данных, которые создаются на базе одной или нескольких базовых таблиц или других представлений, виртуальные таблицы.

Используются для:

– Управления доступом к частям таблиц
– Скрытия подробностей сложных запросов (упрощение понимания)
– Ограничения вставки или обновления значений некоторым диапазоном

CREATE VIEW — создать представление

Синтаксис

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] [ RECURSIVE ] VIEW имя [ ( имя_столбца [, ...] ) ]
    AS запрос
    [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
```

`TEMPORARY` или `TEMP`

С таким указанием представление создаётся как временное. Временные представления автоматически удаляются в конце сеанса. Существующее постоянное представление с тем же именем не будет видно в текущем сеансе, пока существует временное, однако к нему можно обратиться, дополнив имя указанием схемы.

Если в определении представления задействованы временные таблицы, представление так же создаётся как временное (вне зависимости от присутствия явного указания `TEMPORARY`).

RECURSIVE

Создаёт рекурсивное представление. Для рекурсивного представления обязательно должен задаваться список с именами столбцов.

WITH [CASCADED | LOCAL] CHECK OPTION

Это указание управляет поведением автоматически изменяемых представлений. Если оно присутствует, при выполнении операций INSERT и UPDATE с этим представлением будет проверяться, удовлетворяют ли новые строки условию, определяющему представление (то есть, проверяется, будут ли новые строки видны через это представление). Если они не удовлетворяют условию, операция не будет выполнена. Если указание CHECK OPTION отсутствует, команды INSERT и UPDATE смогут создавать в этом представлении строки, которые не будут видны в нём. Указание CHECK OPTION нельзя использовать с рекурсивными представлениями. Поддерживаются следующие варианты проверки:

LOCAL

Новые строки проверяются только по условиям, определённым непосредственно в самом представлении. Любые условия, определённые в нижележащих базовых представлениях, не проверяются (если только в них нет указания CHECK OPTION).

CASCADED

Новые строки проверяются по условиям данного представления и всех нижележащих базовых. Если указано CHECK OPTION, а LOCAL и CASCADED опущено, подразумевается указание CASCADED.

DROP VIEW — удалить представление

Синтаксис

DROP VIEW [IF EXISTS] *имя* [, ...] [CASCADE | RESTRICT]

CASCADE

Автоматически удалять объекты, зависящие от данного представления (например, другие представления), и, в свою очередь, все зависящие от них объекты.

RESTRICT

Отказать в удалении представления, если от него зависят какие-либо объекты. Это поведение по умолчанию.

Пример

Эта команда удаляет представление с именем kinds:

```
DROP VIEW kinds;
```

ALTER VIEW — изменить определение представления

Синтаксис

```
ALTER VIEW [ IF EXISTS ] имя ALTER [ COLUMN ] имя_столбца SET DEFAULT выражение
ALTER VIEW [ IF EXISTS ] имя ALTER [ COLUMN ] имя_столбца DROP DEFAULT
ALTER VIEW [ IF EXISTS ] имя OWNER TO { новый_владелец | CURRENT_USER | SESSION_USER }
ALTER VIEW [ IF EXISTS ] имя RENAME TO новое_имя
ALTER VIEW [ IF EXISTS ] имя SET SCHEMA новая_схема
```

SET/DROP DEFAULT

Эти формы устанавливают или удаляют значение по умолчанию в заданном столбце. Значение по умолчанию подставляется в команды INSERT и UPDATE, вносящие данные в представление, до применения каких-либо правил или триггеров в этом представлении. Таким образом, значения по умолчанию в представлении имеют приоритет перед значениями по умолчанию в нижележащих отношениях.

Примеры

Переименование представления foo в bar:

```
ALTER VIEW foo RENAME TO bar;
```

Добавление значения столбца по умолчанию в изменяемое представление:

```
CREATE TABLE base_table (id int, ts timestamptz);
CREATE VIEW a_view AS SELECT * FROM base_table;
ALTER VIEW a_view ALTER COLUMN ts SET DEFAULT now();
INSERT INTO base_table(id) VALUES(1);  -- в ts окажется значение NULL
INSERT INTO a_view(id) VALUES(2);  -- в ts окажется текущее время
```

Изменяемые представления

Простые представления становятся изменяемыми автоматически: система позволит выполнять команды INSERT, UPDATE и DELETE с таким представлением так же, как и с обычной таблицей. Представление будет автоматически изменяемым, если оно удовлетворяют одновременно всем следующим условиям:

- Список FROM в запросе, определяющем представлении, должен содержать ровно один элемент, и это должна быть таблица или другое изменяемое представление.
- Определение представления не должно содержать предложения WITH, DISTINCT, GROUP BY, HAVING, LIMIT и OFFSET на верхнем уровне запроса.
- Определение представления не должно содержать операции с множествами (UNION, INTERSECT и EXCEPT) на верхнем уровне запроса.
- Список выборки в запросе не должен содержать агрегатные и оконные функции, а также функции, возвращающие множества.

Автоматически обновляемое представление может содержать как изменяемые, так и не изменяемые столбцы. Столбец будет изменяемым, если это простая ссылка на изменяемый столбец нижележащего базового отношения; в противном случае, этот столбец будет доступен только для чтения, и если команда INSERT или UPDATE попытается записать значение в него, возникнет ошибка.

Примеры

Создание представления, содержащего все комедийные фильмы:

```
CREATE VIEW comedies AS
    SELECT *
    FROM films
    WHERE kind = 'Comedy';
```

Создание представления с указанием LOCAL CHECK OPTION:

```
CREATE VIEW universal_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'U'
    WITH LOCAL CHECK OPTION;
```

Создание представления с указанием CASCADED CHECK OPTION:

```
CREATE VIEW pg_comedies AS
    SELECT *
    FROM comedies
    WHERE classification = 'PG'
    WITH CASCADED CHECK OPTION;
```

Создание представления с изменяемыми и неизменяемыми столбцами:

```
CREATE VIEW comedies AS
    SELECT f.*,
```

```
            country_code_to_name(f.country_code) AS country,
            (SELECT avg(r.rating)
             FROM user_ratings r
             WHERE r.film_id = f.id) AS avg_rating
    FROM films f
    WHERE f.kind = 'Comedy';
```

Создание рекурсивного представления, содержащего числа от 1 до 100:

```
CREATE RECURSIVE VIEW public.nums_1_100 (n) AS
    VALUES (1)
UNION ALL
    SELECT n+1 FROM nums_1_100 WHERE n < 100;
```

Материализованные представления

Материализованное представление -- это промежуточное звено между таблицей (TABLE) и представлением (VIEW). Оно наполняется и обновляется с помощью SELECT-запроса, как обычное представление, но хранится на диске, как таблицы. Благодаря этому оно может иметь индексы и все остальные свойства таблиц. Изменение данных в материализованном представлении невозможно, поддерживается только чтение.

Синтаксис

```
CREATE MATERIALIZED VIEW [ IF NOT EXISTS ] имя_таблицы
[ (имя_столбца [, ...] ) ]
AS запрос
[ WITH [ NO ] DATA ]
```

Если указать `WITH NO DATA`, то получить данные из представления будет нельзя до тех пор, пока не будет произведено обновление с данными.

Пример:

```
CREATE MATERIALIZED VIEW full_features_history
AS
SELECT
*
FROM
(
SELECT
id, geog, layer, username, action_type, parent_id, sys_period
```

```
FROM
features
WHERE
layer != 'pano'

UNION ALL

SELECT
id, geog, layer, username, action_type, parent_id, sys_period
FROM
features_history
WHERE
layer != 'pano'
) AS combined
ORDER BY sys_period DESC;
```

Синтаксис

```
REFRESH MATERIALIZED VIEW [ CONCURRENTLY ] имя
[ WITH [ NO ] DATA ]
```

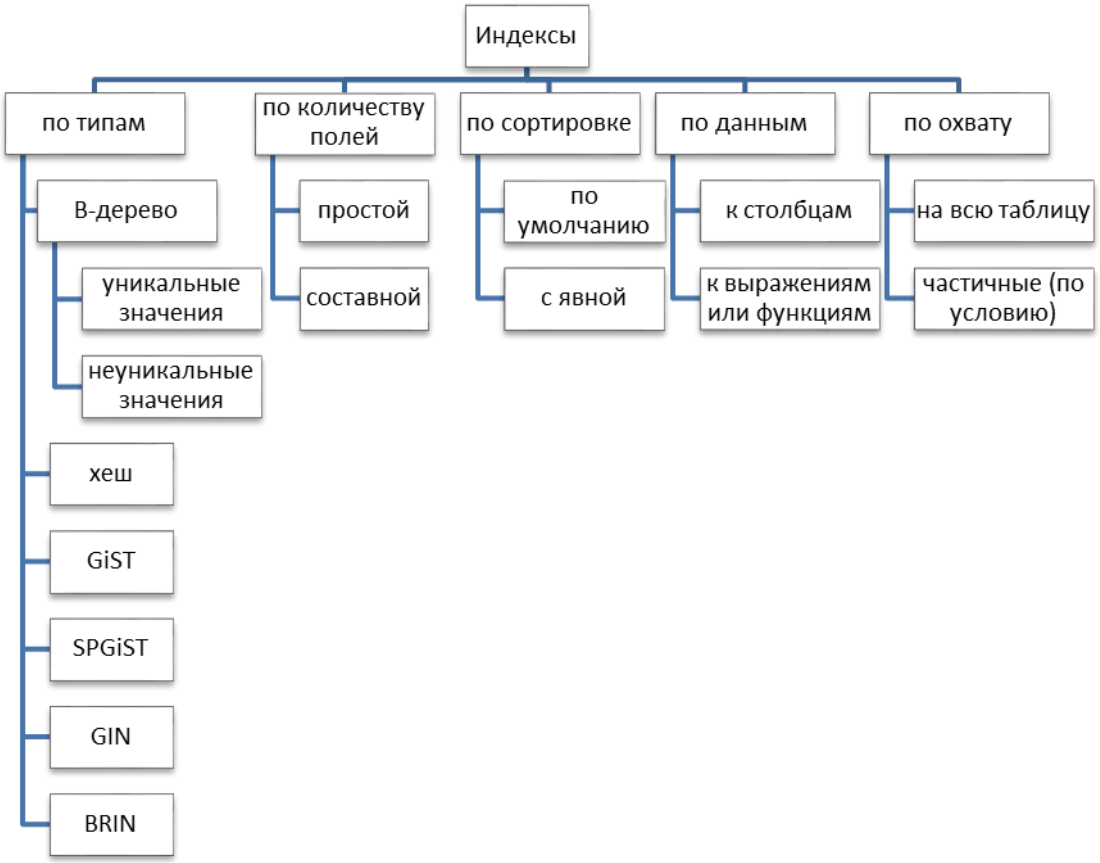
Пример:

```
REFRESH MATERIALIZED VIEW full_features_history;
```

Индексы

Индексы — отдельная физическая структура данных, которая позволяет получать доступ к данным, традиционное средство увеличения производительности БД.

Используя индекс, сервер баз данных может находить и извлекать нужные строки гораздо быстрее, чем без него. Однако с индексами связана дополнительная нагрузка на СУБД в целом, поэтому применять их следует обдуманно.



Тип	Расшифровка	Описание
B-дерево		PostgreSQL может задействовать индекс-B-дерево, когда индексируемый столбец участвует в сравнении с одним из следующих операторов: `<`, `<=`, `=>`,`=>`, `IS NULL`, `IS NOT NULL`, `BETWEEN` и `IN`, `LIKE` (*если* этот шаблон определяется константой и он привязан к началу строки).
хеш		Идея хеширования состоит в том, чтобы значению любого типа данных сопоставить некоторое небольшое число (от 0 до N−1, всего N значений). Такое сопоставление называют хеш-функцией. Полученное число можно использовать как индекс обычного массива, куда и складывать ссылки на строки таблицы (TID).
GiST	Generalized Search Tree (Обобщённое поисковое дерево)	Это сбалансированный иерархический метод доступа, который представляет собой базовый шаблон, на основе которого могут реализовываться произвольные схемы индексации. На базе GiST могут быть реализованы B-

		деревья, R-деревья и многие другие схемы индексации.
SP-GiST	Space-Partitioned GiST (GiST с разбиением пространства)	SP-GiST поддерживает деревья поиска с разбиением, что облегчает разработку широкого спектра различных несбалансированных структур данных, в том числе деревьев квадрантов, а также k-мерных и префиксных деревьев. Общей характеристикой этих структур является то, что они последовательно разбивают пространство поиска на сегменты, которые не обязательно должны быть равного размера. При этом поиск, хорошо соответствующий правилу разбиения, с таким индексом может быть очень быстрым.
GIN	Generalized Inverted Index(Обобщённый инвертированный индекс)	GIN предназначается для случаев, когда индексируемые значения являются составными, а запросы, на обработку которых рассчитан индекс, ищут значения элементов в этих составных объектах. Например, такими объектами могут быть документы, а запросы могут выполнять поиск документов, содержащих определённые слова.
BRIN	Block Range Index(Индекс зон блоков)	BRIN предназначается для обработки очень больших таблиц, в которых определённые столбцы некоторым естественным образом коррелируют с их физическим расположением в таблице. *Зоной блоков* называется группа страниц, физически расположенных в таблице рядом; для каждой зоны в индексе сохраняется некоторая сводная информация.

B-дерево – самый распространенный и используемый способ организации индексов.

B-дерево (по-русски произносится как Б-дерево) — структура данных, дерево поиска. С точки зрения внешнего логического представления, сбалансированное, сильно ветвистое дерево во внешней памяти.

Использование B-деревьев впервые было предложено Р. Бэйером (англ. R. Bayer) и Е. МакКрейтом (англ. E.McCreight) в 1970 году.

Сбалансированность означает, что длина любых двух путей от корня до листов различается не более, чем на единицу.

Ветвистость дерева — это свойство каждого узла дерева ссылаться на большое число узлов-потомков.

С точки зрения физической организации B-дерево представляется как мультисписочная структура страниц внешней памяти, то есть каждому узлу дерева соответствует блок внешней памяти (страница). Внутренние и листовые страницы обычно имеют разную структуру.

B-дерево порядка n представляет собой совокупность иерархически связанных страниц внешней памяти (каждая вершина дерева - страница), обладающая следующими свойствами:

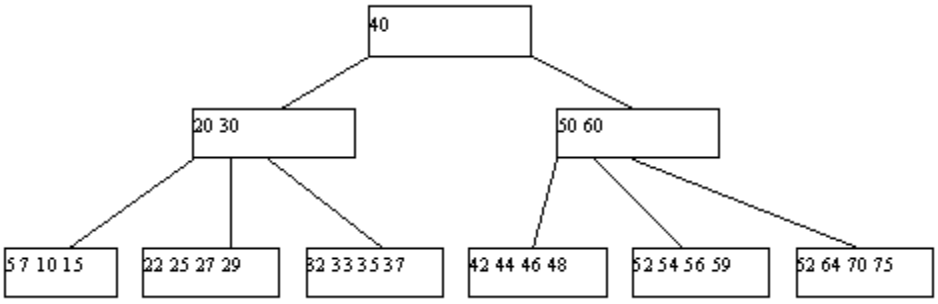
1.Каждая страница содержит не более 2*n элементов (записей с ключом).

2.Каждая страница, кроме корневой, содержит не менее n элементов.

3.Если внутренняя (не листовая) вершина B-дерева содержит m ключей, то у нее имеется m+1 страниц-потомков.

4.Все листовые страницы находятся на одном уровне.

Количество обращений к диску при этом для поиска любой записи одинаково и равно количеству уровней в построенном дереве. Такие деревья называются сбалансированными (balanced) именно потому, что путь от корня до любого листа в этом древе одинаков. Именно термин "сбалансированное" от английского "balanced" - "сбалансированный, взвешенный" и дал название данному методу организации данных.



Классическое B-дерево порядка 2

Поиск в B-дереве производится очевидным образом. Предположим, что происходит поиск ключа K. В основную память считывается корневая страница B-дерева. Предположим, что она содержит ключи k1, k2, ..., km и ссылки на страницы p0, p1, ..., pm. В ней последовательно (или с помощью какого-либо другого метода поиска в основной памяти) ищется ключ K. Если он обнаруживается, поиск завершен. Иначе возможны три ситуации:

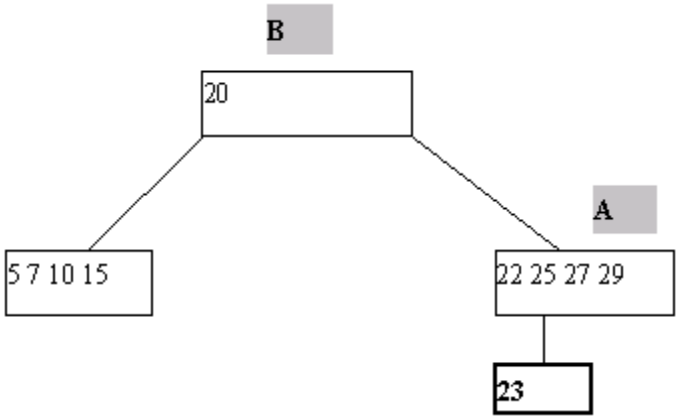
1.Если в считанной странице обнаруживается пара ключей ki и k(i+1) такая, что ki < K < k(i+1), то поиск продолжается на странице pi.

2.Если обнаруживается, что K > km, то поиск продолжается на странице pm.

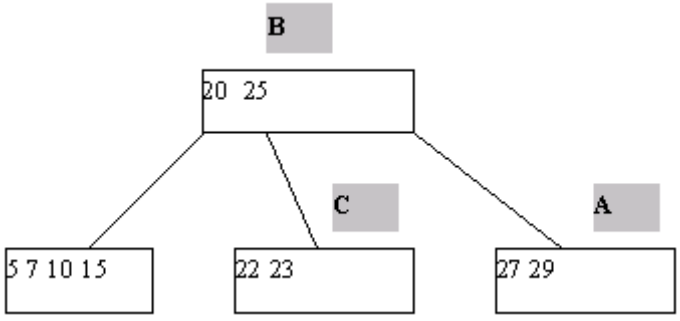
3.Если обнаруживается, что K < k1, то поиск продолжается на странице p0.

Для внутренних страниц поиск продолжается аналогичным образом, пока либо не будет найден ключ K, либо мы не дойдем до листовой страницы. Если ключ не находится и в листовой странице, значит ключ K в B-дереве отсутствует.

Включение нового ключа K в B-дерево выполняется следующим образом. По описанным раньше правилам производится поиск ключа K. Поскольку этот ключ в дереве отсутствует, найти его не удастся, и поиск закончится в некоторой листовой странице A. Далее возможны два случая. Если A содержит менее 2*n ключей, то ключ K просто помещается на свое место, определяемое порядком сортировки ключей в странице A. Если же страница A уже заполнена, то работает процедура расщепления. Заводится новая страница C. Ключи из страницы A (берутся 2*n-1 ключей) + ключ K поровну распределяются между A и C, а средний ключ вместе со ссылкой на страницу C переносится в непосредственно родительскую страницу B. Конечно, страница B может оказаться переполненной, рекурсивно сработает процедура расщепления и т.д., вообще говоря, до корня дерева. Если расщепляется корень, то образуется новая корневая вершина, и высота дерева увеличивается на единицу. Одношаговое включение ключа с расщеплением страницы показано на рисунке.



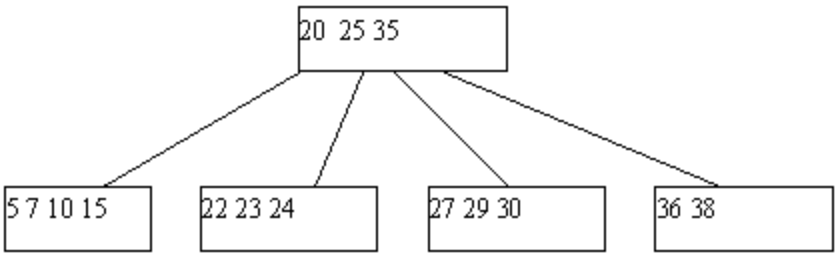
(a) Попытка вставить ключ 23 в уже заполненную страницу



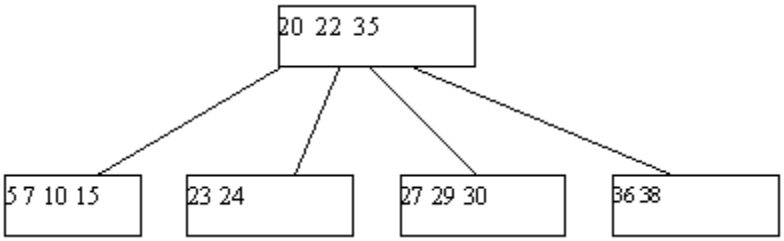
(b) Выполнение включения ключа 22 путем расщепления страницы A

Пример включения ключа в B-дерево

Процедура исключения ключа из классического B-дерева более сложна. Приходится различать два случая - удаление ключа из листовой страницы и удаления ключа из внутренней страницы B-дерева. В первом случае удаление производится просто: ключ просто исключается из списка ключей. При удалении ключа во втором случае для сохранения корректной структуры B-дерева его необходимо заменить на минимальный ключ листовой страницы, к которой ведет последовательность ссылок, начиная от правой ссылки от ключа K (это минимальный содержащийся в дереве ключ, значение которого больше значения K). Тем самым, этот ключ будет изъят из листовой страницы.



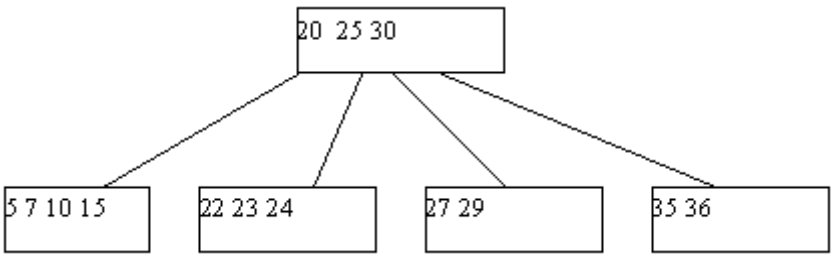
(a) Начальный вид B-дерева



(b) B-дерево после удаления ключа 25

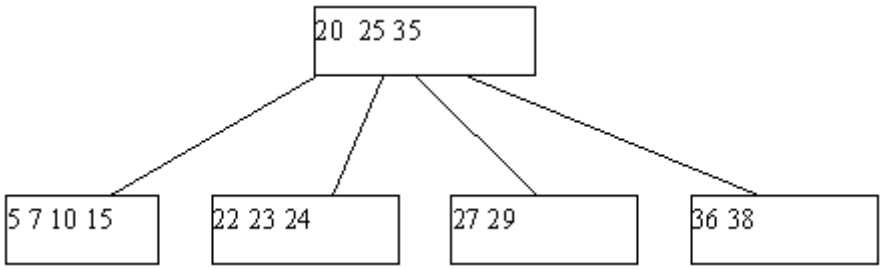
Пример исключения ключа из B-дерева

Поскольку в любом случае в одной из листовых страниц число ключей уменьшается на единицу, может нарушиться то требование, что любая, кроме корневой, страница B-дерева должна содержать не меньше n ключей. Если это действительно случается, начинает работать процедура переливания ключей. Берется одна из соседних листовых страниц (с общей страницей-предком); ключи, содержащиеся в этих страницах, а также средний ключ страницы-предка поровну распределяются между листовыми страницами, и новый средний ключ заменяет тот, который был заимствован у страницы-предка.

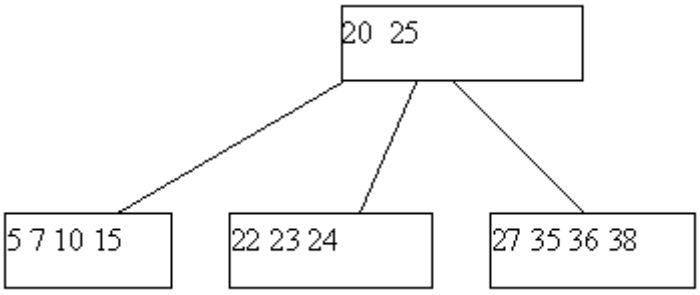


Результат удаления ключа 38 из B-дерева

Может оказаться, что ни одна из соседних страниц непригодна для переливания, поскольку содержат по n ключей. Тогда выполняется процедура слияния соседних листовых страниц. К 2*n-1 ключам соседних листовых страниц добавляется средний ключ из страницы-предка (из страницы-предка он изымается), и все эти ключи формируют новое содержимое исходной листовой страницы. Поскольку в странице-предке число ключей уменьшилось на единицу, может оказаться, что число элементов в ней стало меньше n, и тогда на этом уровне выполняется процедура переливания, а возможно, и слияния. Так может продолжаться до внутренних страниц, находящихся непосредственно под корнем B-дерева. Если таких страниц всего две, и они сливаются, то единственная общая страница образует новый корень. Высота дерева уменьшается на единицу, но по-прежнему длина пути до любого листа одна и та же. Пример удаления ключа со слиянием листовых страниц показан на рисунке.



(a) Начальный вид B-дерева



(b) B-дерево после удаления ключа 29

Пример удаления ключа из B-дерева со слиянием листовых страниц

Синтаксис (синтаксис упрошен, полный см. в документации)

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] имя  ON имя_таблицы
    ( { имя_столбца | ( выражение ) }  [ ASC | DESC ] [ NULLS { FIRST | LAST
} ] [, ...] )
    [ WHERE предикат ]
```

UNIQUE

Указывает, что система должна контролировать повторяющиеся значения в таблице при создании индекса (если в таблице уже есть данные) и при каждом добавлении данных. Попытки вставить или изменить данные, при которых будет нарушена уникальность индекса, будут завершаться ошибкой.

`CONCURRENTLY`

С этим указанием PostgreSQL построит индекс, не устанавливая никаких блокировок, которые бы предотвращали добавление, изменение или удаление записей в таблице; тогда как по умолчанию операция построения индекса блокирует запись (но не чтение) в таблице до своего завершения.

`WHERE`

Если в команде присутствует предложение WHERE, она создаёт *частичный индекс*. Такой индекс содержит записи только для части таблицы, обычно более полезной для индексации, чем остальная таблица.

`ASC`

Указывает порядок сортировки по возрастанию (подразумевается по умолчанию).

`DESC`

Указывает порядок сортировки по убыванию.

`NULLS FIRST`

Указывает, что значения NULL после сортировки оказываются перед остальными. Это поведение по умолчанию с порядком сортировки `DESC`.

`NULLS LAST`

Указывает, что значения NULL после сортировки оказываются после остальных. Это поведение по умолчанию с порядком сортировки `ASC`.

Примеры

Создание индекса-B-дерева по столбцу `title` в таблице `films`:

```
CREATE UNIQUE INDEX title_idx ON films (title);
```

Создание индекса по выражению `lower(title)`, позволяющего эффективно выполнять регистронезависимый поиск:

```
CREATE INDEX ON films ((lower(title)));
```

Создание индекса с нестандартным порядком значений NULL:

```
CREATE INDEX title_idx_nulls_low ON films (title NULLS FIRST);
```

Создание индекса GiST по координатам точек, позволяющего эффективно использовать операторы box с результатом функции преобразования:

```
CREATE INDEX pointloc
    ON points USING gist (box(location,location));
SELECT * FROM points
    WHERE box(location,location) && '(0,0),(1,1)'::box;
```

Создание индекса без блокировки записи в таблицу:

```
CREATE INDEX CONCURRENTLY sales_quantity_index ON sales_table (quantity);
```

ALTER INDEX — изменить определение индекса

Синтаксис

```
ALTER INDEX имя RENAME TO новое_имя
ALTER INDEX имя SET TABLESPACE табл_пространство
ALTER INDEX имя SET ( параметр_хранения = значение [, ... ] )
ALTER INDEX имя RESET ( параметр_хранения [, ... ] )
```

Примеры

Переименование существующего индекса:

```
ALTER INDEX distributors RENAME TO suppliers;
```

Перемещение индекса в другое табличное пространство:

```
ALTER INDEX distributors SET TABLESPACE fasttablespace;
```

Изменение фактора заполнения индекса (предполагается, что это поддерживает метод индекса):

```
ALTER INDEX distributors SET (fillfactor = 75);
REINDEX INDEX distributors;
```

DROP INDEX — удалить индекс

Синтаксис

```
DROP INDEX [ CONCURRENTLY ] имя [, ...] [ CASCADE | RESTRICT ]
```

`CONCURRENTLY`

С этим указанием индекс удаляется, не блокируя параллельные операции выборки, добавления, изменения и удаления данных в таблице индекса. Обычный оператор `DROP INDEX` запрашивает исключительную блокировку для таблицы, запрещая другим обращаться к ней до завершения удаления. Если же добавлено это указание, команда, напротив, будет ждать завершения конфликтующих транзакций.

`CASCADE`

Автоматически удалять объекты, зависящие от данного индекса, и, в свою очередь, все зависящие от них объекты.

`RESTRICT`

Отказать в удалении индекса, если от него зависят какие-либо объекты. Это поведение по умолчанию.

Пример

Эта команда удалит индекс `title_idx`:

```
DROP INDEX title_idx;
```

Тема 11: SQL и PL/pgSQL. Процедуры, функции

Задание лабораторной работы

Цель работы: Получение практических навыков работы с СУБД и языком SQL (операторы create function, create procedure, alter function, drop function).

Задание:

1) Разработать в базе данных, созданной и заполненной на предыдущих лабораторных работах, следующие виды функций:
    a. функция с пустыми входными параметрами, результат которой скалярное выражение;

b. функция со скалярным аргументом, результат которой соответствует типу существующей таблицы;
c. функция с выходными аргументами, определенными с помощью OUT;
d. функция, результат которой определен с помощью RETURNS TABLE.

2) В функциях использовать ветвление, циклы, обработку ошибок и т.д.

3) Объяснить логику работы каждой функции (что она делает).

4) Создать процедуру.

5) Переименовать одну из функций.

6) Удалить одну из функций.

Отчет по лабораторной работе должен содержать:

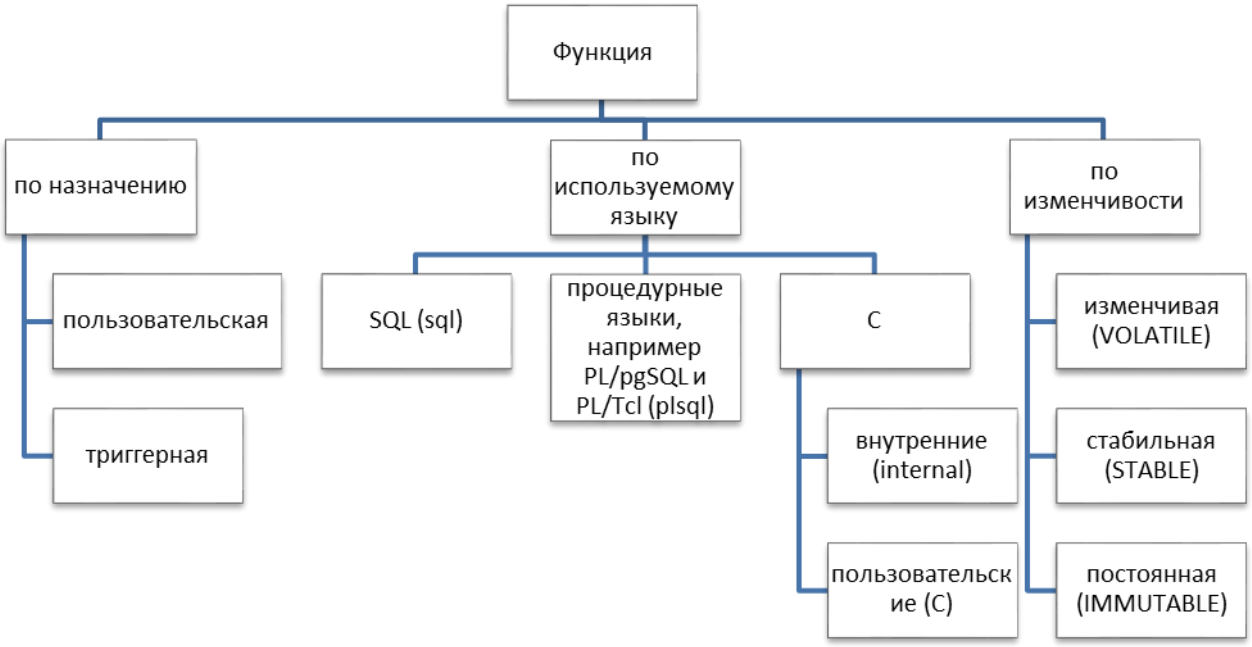
1. Фамилию и номер группы учащегося, задание.
2. Коды операций.
3. Принтскриты всех выполненных операторов.

Методические указания по выполнению лабораторной работы

SQL-функции выполняют произвольный список операторов SQL и возвращают результат последнего запроса в списке. Тело SQL-функции должно представлять собой список SQL-операторов, разделённых точкой с запятой. Точка с запятой после последнего оператора может отсутствовать. Если только функция не объявлена как возвращающая `void`, последним оператором должен быть `SELECT`, либо `INSERT`, `UPDATE` или `DELETE` с предложением `RETURNING`. Любой набор команд на языке SQL можно скомпоновать вместе и обозначить как функцию. Помимо запросов `SELECT`, эти команды могут включать запросы, изменяющие данные (`INSERT`, `UPDATE` и `DELETE`), а также другие SQL-команды. (В SQL-функциях нельзя использовать команды управления транзакциями, например `COMMIT`, `SAVEPOINT`).

Postgres Pro позволяет разрабатывать собственные функции и на языках, отличных от SQL и C. Эти другие языки в целом обычно называются *процедурными языками* (PL, Procedural Languages). Процедурные языки не встроены в сервер Postgres Pro; они предлагаются загружаемыми модулями.

Внутренние функции — это функции, написанные на языке C, и статически скомпонованные в исполняемый код сервера Postgres Pro. В «теле» определения функции задаётся имя функции на уровне C, которое не обязательно должно совпадать с именем, объявленным для использования в SQL.



Обычно все внутренние функции, представленные на сервере, объявляются в ходе инициализации кластера баз данных, но пользователь может воспользоваться командой `CREATE FUNCTION` и добавить дополнительные псевдонимы для внутренней функции.

Пользовательские функции могут быть написаны на C (или на языке, который может быть совместим с C, например C++). Такие функции компилируются в динамически загружаемые объекты (также называемые разделяемыми библиотеками) и загружаются сервером по требованию. Именно метод динамической загрузки отличает функции «на языке C» от «внутренних» функций — правила написания кода по сути одни и те же.

Для каждой функции определяется характеристика *изменчивости*, с возможными вариантами: `VOLATILE`, `STABLE` и `IMMUTABLE`. Если эта характеристика не задаётся явно в команде CREATE FUNCTION, по умолчанию подразумевается `VOLATILE`. Категория изменчивости представляет собой обещание некоторого поведения функции для оптимизатора:

- Изменчивая функция (`VOLATILE`) может делать всё, что угодно, в том числе, модифицировать базу данных. Она может возвращать различные результаты при нескольких вызовах с одинаковыми аргументами. Оптимизатор не делает никаких предположений о поведении таких функций. В запросе, использующем изменчивую функцию, она будет вычисляться заново для каждой строки, когда потребуется её результат.

- Стабильная функция (`STABLE`) не может модифицировать базу данных и гарантированно возвращает одинаковый результат, получая одинаковые аргументы, для всех строк в одном операторе. Эта характеристика позволяет оптимизатору заменить множество вызовов этой функции одним. В частности, выражение, содержащее такую функцию, можно безопасно использовать в условии поиска по индексу. (Так как при поиске по индексу целевое значение вычисляется только один раз, а не для каждой строки, использовать функцию с характеристикой `VOLATILE` в условии поиска по индексу нельзя.)
- Постоянная функция (`IMMUTABLE`) не может модифицировать базу данных и гарантированно всегда возвращает одинаковые результаты для одних и тех же аргументов. Эта характеристика позволяет оптимизатору предварительно вычислить функцию, когда она вызывается в запросе с постоянными аргументами. Например, запрос вида `SELECT ... WHERE x = 2 + 2` можно упростить до `SELECT ... WHERE x = 4`, так как нижележащая функция оператора сложения помечена как `IMMUTABLE`.

Можно определить несколько функций с одним именем SQL, если эти функции будут принимать разные аргументы. Другими словами, имена функций можно *перегружать*.

```
CREATE FUNCTION test(int, real) RETURNS ...
CREATE FUNCTION test(smallint, double precision) RETURNS ...
```

Синтаксис

CREATE FUNCTION — изменить определение функции

```
CREATE [ OR REPLACE ] FUNCTION
    имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента [ { DEFAULT |
= } выражение_по_умолчанию ] [, ...] ] )
    [ RETURNS тип_результата
      | RETURNS TABLE ( имя_столбца тип_столбца [, ...] ) ]
  { LANGUAGE имя_языка
    | IMMUTABLE | STABLE | VOLATILE | [ NOT ] LEAKPROOF
    | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT
    | COST стоимость_выполнения
  } ...
```

ALTER FUNCTION — изменить определение функции

```
ALTER FUNCTION имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    действие [ ... ] [ RESTRICT ]
ALTER FUNCTION имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    RENAME TO новое_имя
ALTER FUNCTION имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    OWNER TO { новый_владелец | CURRENT_USER | SESSION_USER }
ALTER FUNCTION имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    SET SCHEMA новая_схема
ALTER FUNCTION имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    DEPENDS ON EXTENSION имя_расширения
```

DROP FUNCTION — удалить функцию

```
DROP FUNCTION [ IF EXISTS ] имя [ ( [ [ режим_аргумента ] [ имя_аргумента ]
тип_аргумента [, ...] ] ) ] [, ...]
    [ CASCADE | RESTRICT ]
```

CALLED ON NULL INPUT
RETURNS NULL ON NULL INPUT
STRICT

CALLED ON NULL INPUT меняет функцию так, чтобы она вызывалась, когда некоторые или все её аргументы равны NULL. RETURNS NULL ON NULL INPUT или STRICT меняет функцию так, чтобы она не вызывалась, когда некоторые или

все её аргументы равны NULL, а вместо вызова автоматически выдавался результат NULL.

`IMMUTABLE`

`STABLE`

`VOLATILE`

Устанавливает заданный вариант изменчивости функции.

`COST` *`стоимость_выполнения`*

Изменяет ориентировочную стоимость выполнения функции.

`ROWS` *`строк_в_результате`*

Изменяет ориентировочное число строк в результате функции, возвращающей множество.

`CASCADE`

Автоматически удалять объекты, зависящие от данной функции (например, операторы или триггеры), и, в свою очередь, все зависящие от них объекты.

`RESTRICT`

Отказать в удалении функции, если от неё зависят какие-либо объекты. Это поведение по умолчанию.

Примеры:

Функции SQL

1) Простейшая возможная функция SQL не имеет аргументов и просто возвращает базовый тип, например `integer`:

```
CREATE FUNCTION one() RETURNS integer AS $$
    SELECT 1 AS result;
$$ LANGUAGE SQL;
-- Альтернативная запись строковой константы:
CREATE FUNCTION one() RETURNS integer AS '
    SELECT 1 AS result;
' LANGUAGE SQL;
SELECT one();
```

2) функция SQL принимает в аргументах базовые типы:

```
CREATE FUNCTION add_em(x integer, y integer) RETURNS integer AS
$$
    SELECT x + y;
$$ LANGUAGE SQL;
SELECT add_em(1, 2) AS answer;
```

3) Обращение к входным аргументам по номерам

```
CREATE FUNCTION add_em(integer, integer) RETURNS float8 AS $$
    SELECT $1 + $2;
$$ LANGUAGE SQL;
```

4) Функция со сложным типом входного аргумента

```
CREATE TABLE emp (
    name        text,
    salary      numeric,
    age         integer,
    cubicle     point
);
INSERT INTO emp VALUES ('Bill', 4200, 45, '(2,1)');
CREATE FUNCTION double_salary(emp) RETURNS numeric AS $$
    SELECT $1.salary * 2 AS salary;
$$ LANGUAGE SQL;
SELECT name, double_salary(emp.*) AS dream
    FROM emp
    WHERE emp.cubicle ~= point '(2,1)';
```

5) функция возвращает одну строку emp

```
CREATE FUNCTION new_emp() RETURNS emp AS $$
    SELECT text 'None' AS name,
        1000.0 AS salary,
        25 AS age,
        point '(2,2)' AS cubicle;
$$ LANGUAGE SQL;
```

6) Функции с выходными параметрами

```
CREATE FUNCTION add_em (IN x int, IN y int, OUT sum int)
AS 'SELECT x + y'
LANGUAGE SQL;
SELECT add_em(3,7);

CREATE TYPE sum_prod AS (sum int, product int);
```

```
CREATE FUNCTION sum_n_product (int, int) RETURNS sum_prod
AS 'SELECT $1 + $2, $1 * $2'
LANGUAGE SQL;
```

7) Функция с аргументами по умолчанию

```
CREATE FUNCTION foo(a int, b int DEFAULT 2, c int DEFAULT 3)
RETURNS int
LANGUAGE SQL
AS $$
    SELECT $1 + $2 + $3;
$$;
```

8) Удаление функции

```
DROP FUNCTION sqrt5(integer);
```

9) Переименование функции sqrt для типа integer в square_root:

```
ALTER FUNCTION sqrt(integer) RENAME TO square_root;
```

Функции PL/pgSQL

10) Функция увеличения целого числа на 1, использующая именованный аргумент, на языке PL/pgSQL:

```
CREATE OR REPLACE FUNCTION increment(i integer) RETURNS integer
AS $$
        BEGIN
                RETURN i + 1;
        END;
$$ LANGUAGE plpgsql;
```

11) Функция, возвращающая запись с несколькими выходными параметрами:

```
CREATE FUNCTION dup(in int, out f1 int, out f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;
SELECT * FROM dup(42);
```

12) Явное объявление составного типа:

```
CREATE TYPE dup_result AS (f1 int, f2 text);
```

```
CREATE FUNCTION dup(int) RETURNS dup_result
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;
SELECT * FROM dup(42);
```

13) Функция возвращает несколько столбцов (TABLE):

```
CREATE FUNCTION dup(int) RETURNS TABLE(f1 int, f2 text)
    AS $$ SELECT $1, CAST($1 AS text) || ' is text' $$
    LANGUAGE SQL;
SELECT * FROM dup(42);
```

Процедура

Процедура представляет собой объект базы данных, подобный функции. Отличие состоит в том, что процедура не возвращает значение, и поэтому для неё не определяется возвращаемый тип. Тогда как функция вызывается в составе запроса или команды DML, процедура вызывается явно, оператором CALL.

Синтаксис

CREATE PROCEDURE — создать процедуру

```
CREATE [ OR REPLACE ] PROCEDURE
    имя ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[ { DEFAULT | = } выражение_по_умолчанию ] [, ...] ] )
  { LANGUAGE имя_языка
```

ALTER PROCEDURE — изменить определение процедуры

```
ALTER PROCEDURE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    действие [ ... ] [ RESTRICT ]
ALTER PROCEDURE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    RENAME TO новое_имя
ALTER PROCEDURE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    OWNER TO { новый_владелец | CURRENT_USER | SESSION_USER }
```

```
ALTER PROCEDURE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    SET SCHEMA новая_схема
ALTER PROCEDURE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    DEPENDS ON EXTENSION имя_расширения
```

DROP PROCEDURE — удалить процедуру

```
DROP PROCEDURE [ IF EXISTS ] имя [ ( [ [ режим_аргумента ] [ имя_аргумента ]
тип_аргумента [, ...] ] ) ] [, ...]
    [ CASCADE | RESTRICT ]
```

Примеры

1) Пример простой функции

```
CREATE PROCEDURE insert_data(a integer, b integer)
LANGUAGE SQL
AS $$
INSERT INTO tbl VALUES (a);
INSERT INTO tbl VALUES (b);
$$;

CALL insert_data(1, 2);
```

2) Переименование процедуры `insert_data` с двумя аргументами типа `integer` в `insert_record`:

```
ALTER PROCEDURE insert_data(integer, integer) RENAME TO insert_record;
```

3) Смена владельца процедуры `insert_data` с двумя аргументами типа `integer` на `joe`:

```
ALTER PROCEDURE insert_data(integer, integer) OWNER TO joe;
```

4) Смена схемы процедуры `insert_data` с двумя аргументами типа `integer` на `accounting`:

```
ALTER PROCEDURE insert_data(integer, integer) SET SCHEMA accounting;
```

И процедура и функция являются подпрограммами и существуют инструкции, которые могут изменить и удалить объект без уточнения, чем он является.

Синтаксис

ALTER ROUTINE — изменить определение подпрограммы

```
ALTER ROUTINE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    действие [ ... ] [ RESTRICT ]
ALTER ROUTINE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    RENAME TO новое_имя
ALTER ROUTINE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    OWNER TO { новый_владелец | CURRENT_USER | SESSION_USER }
ALTER ROUTINE имя [ ( [ [ режим_аргумента ] [ имя_аргумента ] тип_аргумента
[, ...] ] ) ]
    SET SCHEMA новая_схема
```

DROP ROUTINE — удалить подпрограмму

```
DROP ROUTINE [ IF EXISTS ] имя [ ( [ [ режим_аргумента ] [ имя_аргумента ]
тип_аргумента [, ...] ] ) ] [, ...]
    [ CASCADE | RESTRICT ]
```

PL/pgSQL

PL/pgSQL это блочно-структурированный язык. Текст тела функции/процедуры должен быть *блоком*. Структура блока:

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

Каждое объявление и каждый оператор в блоке должны завершаться символом ";"(точка с запятой). Блок, вложенный в другой блок, должен иметь точку с запятой после `END`, как показано выше. Однако финальный `END`, завершающий тело функции, не требует точки с запятой.

Операторы PL/SQL

Оператор	Описание	Примеры
имя [CONSTANT] *тип* [NOT NULL] [{ DEFAULT | := | = } *выражение*];	Объявление переменной	quantity integer DEFAULT 32; url varchar := 'http://mysite.com'; user_id CONSTANT integer := 10;

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

переменная%TYPE	Наследование типов данных	user_id users.user_id%TYPE;

имя
имя_таблицы%ROWTYPE;
имя имя_составного_типа;
	Типы кортежей	CREATE FUNCTION merge_fields(t_row table1) RETURNS text AS $$
		DECLARE
		t2_row table2%ROWTYPE;
		BEGIN
		SELECT * INTO t2_row FROM table2 WHERE ... ;
		RETURN t_row.f1 \|\| t2_row.f3 \|\| t_row.f5 \|\| t2_row.f7;
		END;
		$$ LANGUAGE plpgsql;

имя RECORD;	Переменные типа record похожи на переменные строкового типа, но они не имеют предопределённой структуры. Они приобретают фактическую структуру от строки, которая им	

	присваивается командами SELECT или FOR.	
переменная { := | = } *выражение*;	Присвоение значения переменной	tax := subtotal * 0.06; my_record.user_id := 20;
PERFORM *запрос*;	Выполнение команды, не возвращающей результат	PERFORM create_mv('cs_session_page_requests_mv', my_query);

SELECT *выражения_select* INTO [`STRICT`] *цель* FROM ...;
INSERT ... RETURNING *выражения* INTO [`STRICT`] *цель*;
UPDATE ... RETURNING *выражения* INTO [`STRICT`] *цель*;
DELETE ... RETURNING *выражения* INTO [`STRICT`] *цель*;
	Результат SQL-команды, возвращающей одну строку (возможно из нескольких столбцов), может быть присвоен переменной типа `record`, переменной строкового типа или списку скалярных переменных. Для этого нужно к основной команде SQL добавить предложение `INTO`.	SELECT * INTO myrec FROM emp WHERE empname = myname;
		IF NOT FOUND THEN
		  RAISE EXCEPTION 'Сотрудник % не найден', myname;
		END IF;

EXECUTE *строка-команды* [`INTO [STRICT]` *цель*] [`USING` *выражение* [`, ...`]];	Выполнение динамически формируемых команд	EXECUTE 'SELECT count(*) FROM mytable WHERE inserted_by = $1
		AND inserted <= $2'
		  INTO c
		  USING checked_user, checked_date;
		
		EXECUTE 'UPDATE tbl SET '
		  || quote_ident(colname)
		  || ' = '
		  || quote_literal(newvalue)
		  || ' WHERE key = '

		|| quote_literal(keyvalue);
GET [CURRENT] DIAGNOSTICS *переменная* { = | := } *элемент* [, ...];	Определить результат команды	GET DIAGNOSTICS integer_var = ROW_COUNT;
Управляющие структуры		
RETURN *выражение*;	RETURN с последующим выражением прекращает выполнение функции и возвращает значение выражения в вызывающую программу.	

IF ... THEN ... END IF
 	
IF ... THEN ... ELSE ... END IF
 	
IF ... THEN ... ELSIF ... THEN ... ELSE ... END IF
	Условный оператор	IF v_user_id <> 0 THEN
		  UPDATE users SET email = v_email WHERE user_id = v_user_id;
		END IF;
		
		IF v_count > 0 THEN
		  INSERT INTO users_count (count) VALUES (v_count);
		  RETURN 't';
		ELSE
		  RETURN 'f';
		END IF;
		
		IF number = 0 THEN
		  result := 'zero';
		ELSIF number > 0 THEN
		  result := 'positive';
		ELSIF number < 0 THEN
		  result := 'negative';
		ELSE

		-- остаётся только один вариант: number имеет значение NULL result := 'NULL'; END IF;
CASE ... WHEN ... THEN ... ELSE ... END CASE CASE WHEN ... THEN ... ELSE ... END CASE	Условный оператор (может использоваться в SQL операторах)	CASE x WHEN 1, 2 THEN msg := 'один или два'; ELSE msg := 'значение, отличное от один или два'; END CASE; 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;

[<<*метка*>>]
LOOP
операторы
END LOOP [*метка*];
	безусловный цикл, который повторяется до бесконечности, пока не будет прекращён операторами `EXIT` или `RETURN`.	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 примеру
		END LOOP;

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`.	<<ablock>>
		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.	LOOP -- здесь производятся вычисления EXIT WHEN count > 100; CONTINUE WHEN count < 50; -- вычисления для count в диапазоне 50 .. 100 END LOOP;
[<<*метка*>>] 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.	WHILE amount_owed > 0 AND gift_certificate_balance > 0 LOOP -- здесь производятся вычисления END LOOP; WHILE NOT done LOOP -- здесь производятся вычисления END LOOP;
[<<*метка*>>]	В этой форме цикла `FOR` итерации выполняются по диапазону	FOR i IN 1..10 LOOP

FOR *имя* IN [REVERSE] *выражение* .. *выражение* [BY *выражение*] LOOP *операторы* END LOOP [*метка*];	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.	-- 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;
[<<*метка*>>] FOR *цель* IN *запрос* LOOP *операторы* END LOOP [*метка*];	цикл по результатам запроса 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.	CREATE FUNCTION cs_refresh_mviews() RETURNS integer AS $$ DECLARE mviews RECORD; BEGIN RAISE NOTICE 'Refreshing materialized views...'; FOR mviews IN SELECT * FROM cs_materialized_views ORDER BY sort_key LOOP -- Здесь "mviews" содержит одну запись из cs_materialized_views RAISE NOTICE 'Refreshing materialized view %s ...', quote_ident(mviews.mv_name); EXECUTE format('TRUNCATE TABLE %I', mviews.mv_name);

		EXECUTE format('INSERT INTO %I %s', mviews.mv_name, mviews.mv_query); END LOOP; RAISE NOTICE 'Done refreshing materialized views.'; RETURN 1; END; $$ LANGUAGE plpgsql;

[<<*метка*>>]
FOREACH *цель* [SLICE *число*] IN ARRAY *выражение*
LOOP
операторы
END LOOP [*метка*];
	Цикл по элементам массива
	
	Цикл FOREACH очень похож на FOR. Отличие в том, что вместо перебора строк SQL-запроса происходит перебор элементов массива. (В целом, FOREACH предназначен для перебора выражений составного типа. Варианты реализации цикла для работы с прочими составными выражениями помимо массивов могут быть добавлены в будущем.)
		CREATE FUNCTION sum(int[]) RETURNS int8 AS $$
		DECLARE
		s int8 := 0;
		x int;
		BEGIN
		FOREACH x IN ARRAY $1
		LOOP
		s := s + x;
		END LOOP;
		RETURN s;
		END;
		$$ LANGUAGE plpgsql;

[<<*метка*>>]
[DECLARE
объявления]
BEGIN
операторы
	Обработка ошибок
	
	По умолчанию любая возникающая ошибка прерывает выполнение функции на PL/pgSQL, а также транзакцию, относящуюся к этой функции. Использование в блоке секции EXCEPTION позволяет перехватывать и обрабатывать ошибки.
		INSERT INTO mytab(firstname, lastname) VALUES('Tom', 'Jones');
		BEGIN
		UPDATE mytab SET firstname = 'Joe' WHERE lastname = 'Jones';
		x := x + 1;
		y := x / 0;
		EXCEPTION

EXCEPTION WHEN *условие* [OR *условие* ...] THEN *операторы_обработчика* [WHEN *условие* [OR *условие* ...] THEN *операторы_обработчика* ...] END;		WHEN division_by_zero THEN RAISE NOTICE 'перехватили ошибку division_by_zero'; RETURN x; END;

GET [CURRENT] DIAGNOSTICS *переменная* { = | := } *элемент* [, ...];	Получение информации о месте выполнения	CREATE OR REPLACE FUNCTION outer_func() RETURNS integer AS $$
		BEGIN
		RETURN inner_func();
		END;
		$$ LANGUAGE plpgsql;
		
		CREATE OR REPLACE FUNCTION inner_func() RETURNS integer AS $$
		DECLARE
		stack text;
		BEGIN
		GET DIAGNOSTICS stack = PG_CONTEXT;
		RAISE NOTICE E'--- Стек вызова ---\n%', stack;
		RETURN 1;

		END; $$ LANGUAGE plpgsql;

Тема 12: SQL и PL/pgSQL. Курсоры

Задание лабораторной работы

Цель работы: Получение практических навыков работы с СУБД и языком SQL (операторы declare, open, fetch, move, update, delete, close).

Задание:

1) Разработать в базе данных, созданной и заполненной на предыдущих лабораторных работах:
    a. создать связанный с простым запросом курсор и использовать цикл для перемещения по нему MOVE и в теле цикла менять каждую четную строку и удалят каждую нечетную;
    b. создать связанный с параметрическим запросом курсор и вывести данные из пятой с конца строки на экран, для перемещения использовать FETCH;
    c. создать несвязанный курсор и открыть его для динамически создаваемого запроса.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание.
2. Коды операций.
3. Принтскриты всех выполненных операторов.

Методические указания по выполнению лабораторной работы

Курсор – поименованная область памяти базы данных для хранения результат выполнения оператора SQL.

Курсор инкапсулирует запрос и получает результаты порционно (по несколько строк за раз), что исключает возможность переполнения памяти, и позволяет перемещаться по строкам. Циклы FOR автоматически используют курсоры, чтобы избежать проблем с памятью.

Минусы использования курсоров:

- не позволяет проводить операции изменения над всем объемом данных
- ниже скорость выполнения.

Для создания курсора необходимо объявить курсорную переменную. Если курсор используются в рамках языка SQL, то его можно сразу использовать, он открывается не явно (если курсор связанный). В случае использования курсора в языке PL/pgSQL курсор требуется открыть явно.

После открытия курсор может использоваться для перемещения по записям (FETCH, MOVE), для изменения данных в текущей записи курсора (UPDATE, DELETE).

Когда курсор закрыт, никакие операции с ним невозможны. Закрывать курсор следует, когда он становится ненужным.

Все не удерживаемые открытые курсоры закрываются неявно при завершении транзакции командами COMMIT или ROLLBACK. Удерживаемый курсор закрывается неявно, если транзакция, его создавшая, прерывается командой ROLLBACK. Если создавшая его транзакция завершается успешной фиксацией, удерживаемый курсор остаётся открытым до явного вызова команды CLOSE или отключения клиента.

Объявление курсорных переменных

Синтаксис

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

SCROLL

Прокручиваемый (можно перемещаться и вперед и назад по записям)

NO SCROLL

Не прокручиваемый (можно перемещаться только вперед)

Примеры

```
DECLARE
--несвязанный курсор
    curs1 refcursor;
--связанный с простым запросом
    curs2 CURSOR FOR SELECT * FROM tenk1;
--связанный с  параметризованным запросом
    curs3 CURSOR (key integer) FOR SELECT * FROM tenk1 WHERE
unique1 = key;
```

Открытие курсора

Синтаксис

Открытие несвязанного курсора с простым запросом (OPEN FOR запрос)

```
OPEN несвязанная_переменная_курсора [[NO] SCROLL] FOR запрос;
```

Открытие несвязанного курсора с динамически формируемым запросом (OPEN FOR EXECUTE)

```
OPEN несвязанная_переменная_курсора [[NO] SCROLL] FOR EXECUTE строка_запроса
                                    [USING выражение [, ...]];
```

Открытие связанного курсора

```
OPEN связанная_переменная_курсора [( [имя_аргумента :=] значение_аргумента [, ...] )];
```

Примеры

```
OPEN curs2;
OPEN curs3(42);
OPEN curs3(key := 42);
```

Использование курсора

Синтаксис

Прокрутка с получением данных (FETCH)

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

Прокрутка без получения данных (MOVE)

```
MOVE [направление { FROM | IN }] курсор;
```

Здесь *направление* может быть пустым или принимать следующее значение:

NEXT

Выбрать следующую строку. Это действие подразумевается по умолчанию, если *направление* опущено.

PRIOR

Выбрать предыдущую строку.

FIRST

Выбрать первую строку запроса (аналогично указанию ABSOLUTE 1).

LAST

Выбрать последнюю строку запроса (аналогично ABSOLUTE -1).

ABSOLUTE *число*

Выбрать строку под номером *число* с начала, либо под номером abs(*число*) с конца, если *число* отрицательно. Если *число* выходит за границы набора строк, курсор размещается перед первой или после последней строки; в частности, с ABSOLUTE 0 курсор оказывается перед первой строкой.

RELATIVE *число*

Выбрать строку под номером *число*, считая со следующей вперёд, либо под номером abs(*число*), считая с предыдущей назад, если *число* отрицательно. RELATIVE 0 повторно считывает текущую строку, если таковая имеется.

число

Выбрать следующее *число* строк (аналогично FORWARD *число*).

ALL

Выбрать все оставшиеся строки (аналогично FORWARD ALL).

FORWARD

Выбрать следующую строку (аналогично NEXT).

FORWARD *число*

Выбрать следующее *число* строк. FORWARD 0 повторно выбирает текущую строку.

FORWARD ALL

Выбрать все оставшиеся строки.

BACKWARD

Выбрать предыдущую строку (аналогично PRIOR).

BACKWARD *число*

Выбрать предыдущее *число* строк (с перемещением назад). BACKWARD 0 повторно выбирает текущую строку.

BACKWARD ALL

Выбрать все предыдущие строки (с перемещением назад).

Когда курсор позиционирован на строку таблицы, эту строку можно изменить или удалить при помощи курсора. Есть ограничения на то, каким может быть запрос курсора (в частности, не должно быть группировок), и крайне желательно использовать указание FOR UPDATE.

Синтаксис

```
      Изменение данных (UPDATE/DELETE WHERE CURRENT OF)
UPDATE таблица SET ... WHERE CURRENT OF курсор;
DELETE FROM таблица WHERE CURRENT OF курсор;
```

Примеры

```
BEGIN WORK;
-- Создание курсора:
DECLARE liahona SCROLL CURSOR FOR SELECT * FROM films;
-- Получение первых 5 строк через курсор liahona:
FETCH FORWARD 5 FROM liahona;
-- Получение предыдущей строки:
FETCH PRIOR FROM liahona;
-- Закрытие курсора и завершение транзакции:
CLOSE liahona;
COMMIT WORK;

update tab11 set object_name = 'update1' , owner = 'OW'
   where current of CUR6;
```

Закрытие курсора и освобождение памяти

Используется для того, чтобы освободить ресурсы раньше, чем закончится транзакция, или чтобы освободить курсорную переменную для повторного открытия.

Синтаксис

```
CLOSE { имя | ALL }
```

Пример

```
CLOSE liahona;
```

Тема 13: SQL и PL/pgSQL. Триггеры

Задание лабораторной работы

Цель работы: Получение практических навыков работы с СУБД и языком SQL (операторы create trigger, alter trigger, drop trigger, create event trigger, alter event trigger, drop event trigger).

Задание:

1) Разработать в базе данных, созданной и заполненной на предыдущих лабораторных работах:
    a. триггеры на изменение данных (для таблиц) для разных событий модификации данных (вначале создаются триггерные функции):
        i. before на всю таблицу;
        ii. after на одну строку;
        iii. instead of на выбор.
    b. триггер на событие;
    c. используете в теле триггеров специальные переменные.
    d. Проверьте выполнение триггеров, генерирую нужные события (выполняя соответствующие операторы);
    e. измените один из триггеров (переименуйте и отключите);
    f. удалите один из триггеров (после удаления, восстановите, создав заново).

Отчет по лабораторной работе должен содержать:

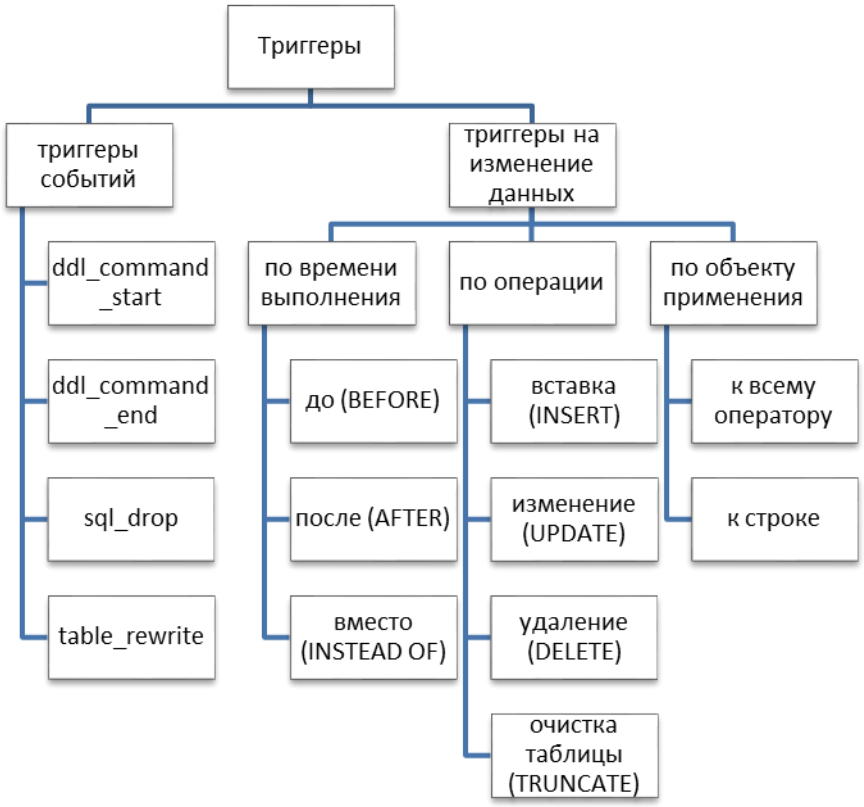
1. Фамилию и номер группы учащегося, задание.
2. Коды операций.
3. Принтскриты всех выполненных операторов.

Методические указания по выполнению лабораторной работы

Триггерный функции

В PL/pgSQL можно создавать триггерные функции, которые будут вызываться при изменениях данных или событиях в базе данных. Триггерная функция создаётся командой CREATE FUNCTION, при этом у функции не должно быть аргументов, а типом возвращаемого значения должен быть trigger (для триггеров, срабатывающих при изменениях данных) или event_trigger (для триггеров, срабатывающих при событиях в базе). Для триггеров автоматически определяются специальные локальные переменные с именами вида TG_имя, описывающие условие, повлёкшее вызов триггера.

Триггер является указанием, что база данных должна автоматически выполнить заданную функцию, всякий раз, когда выполнен определённый тип операции (происходит событие).



Триггеры на изменение данных (к операциям DML)

Триггеры можно использовать с таблицами, с представлениями и с внешними таблицами.

В следующей таблице перечисляются типы триггеров, которые могут использоваться для таблиц, представлений и сторонних таблиц:

Когда	Событие	На уровне строк	На уровне оператора
BEFORE	INSERT/UPDATE/DELETE	Таблицы и сторонние таблицы	Таблицы, представления и сторонние таблицы
	TRUNCATE	—	Таблицы
AFTER	INSERT/UPDATE/DELETE	Таблицы и сторонние таблицы	Таблицы, представления и сторонние таблицы
	TRUNCATE	—	Таблицы
INSTEAD OF	INSERT/UPDATE/DELETE	Представления	—
	TRUNCATE	—	—

Специальные переменные, которые могут использоваться в триггерах

Когда функция на PL/pgSQL срабатывает как триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных:

NEW

Тип данных RECORD. Переменная содержит новую строку базы данных для команд INSERT/UPDATE в триггерах уровня строки. В триггерах уровня оператора и для команды DELETE этой переменной значение не присваивается.

OLD

Тип данных RECORD. Переменная содержит старую строку базы данных для команд UPDATE/DELETE в триггерах уровня строки. В триггерах уровня оператора и для команды INSERT этой переменной значение не присваивается.

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 процедуру.

TG_ARGV[]

Тип данных массив text. Аргументы от оператора CREATE TRIGGER. Индекс массива начинается с 0. Для недопустимых значений индекса (< 0 или >= tg_nargs) возвращается NULL.

Синтаксис

CREATE TRIGGER -- создать триггер

```
CREATE TRIGGER имя { BEFORE | AFTER | INSTEAD OF } { событие [
OR ... ] }
    ON table_name
    [ FROM ссылающаяся_таблица ]
    [ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE |
INITIALLY DEFERRED } ]
    [ FOR [ EACH ] { ROW | STATEMENT } ]
    [ WHEN ( условие ) ]
    EXECUTE PROCEDURE имя_функции ( аргументы )
```

```
Здесь допускается событие:    INSERT, UPDATE [ OF имя_колонки [,
... ] ],     DELETE, TRUNCATE
```

ALTER TRIGGER — изменить определение триггера

```
ALTER TRIGGER имя ON имя_таблицы RENAME TO новое_имя
ALTER TRIGGER имя ON имя_таблицы DEPENDS ON EXTENSION имя_расширения
```

DROP TRIGGER — удалить триггер

```
DROP TRIGGER [ IF EXISTS ] имя ON имя_таблицы [ CASCADE | RESTRICT ]
```

IF EXISTS

Не считать ошибкой, если триггер не существует. В этом случае будет выдано замечание.

CASCADE

Автоматически удалять объекты, зависящие от данного триггера, и, в свою очередь, все зависящие от них объекты.

RESTRICT

Отказать в удалении триггера, если от него зависят какие-либо объекты. Это поведение по умолчанию.

Примеры:

1) Триггер 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
            RAISE 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();
```

2) Переименование триггера

```
ALTER TRIGGER emp_stamp ON emp RENAME TO emp_track_chgs;
```

3) Удаление триггера

```
DROP TRIGGER if_dist_exists ON films;
```

Триггеры событий

PostgreSQL требует, чтобы процедура, которая вызывается как триггер события, была объявлена без аргументов и имела тип возвращаемого значения event_trigger.

Триггер события срабатывает всякий раз, когда в базе данных, в которой он определён, происходит связанное с ним событие.

Функция триггера выполняется, когда происходит указанное событие и удовлетворяется связанное с триггером условие WHEN (если такое имеется). Владельцем триггера становится пользователь его создавший.

Для создания триггера события предварительно нужно создать функцию, со специальным возвращаемым типом event_trigger. Данная функция не обязана возвращать значение (и может не возвращать). Возвращаемый тип служит лишь указанием на то, что функция будет вызываться из триггера события.

Если есть несколько триггеров на одно и то же событие, то они будут вызываться в алфавитном порядке по имени триггера.

В определении триггера можно использовать условие WHEN, чтобы, например, триггер ddl_command_start срабатывал только для отдельных команд, которые нужно перехватить. Триггеры событий часто используются для ограничения диапазона DDL-команд, доступных пользователям.

Виды событий

Событие	Команды	Описание

ddl_command_start	**Перед** CREATE, ALTER, DROP, SECURITY LABEL, COMMENT, GRANT и REVOKE	Проверка на существование объекта перед срабатыванием триггера не производится. В качестве исключения, однако, это событие не происходит для команд DDL, обращающихся к общим объектам кластера базы данных — базам данных, табличным пространствам, ролям, а также к самим триггерам событий. Событие ddl_command_start также происходит непосредственно перед выполнением команды SELECT INTO, так как она равнозначна команде CREATE TABLE AS.
ddl_command_end	**После** CREATE, ALTER, DROP, SECURITY LABEL, COMMENT, GRANT и REVOKE	Чтобы получить дополнительную информацию об операциях DDL, повлёкших произошедшее событие, вызовите функцию pg_event_trigger_ddl_commands(), возвращающую множество, из кода обработчика события ddl_command_end, триггер срабатывает после того, как эти действия имели место (но до фиксации транзакции), так что в системных каталогах можно увидеть уже изменённое состояние.
sql_drop	перед ddl_command_end для команд, которые удаляют объекты базы данных	Для получения списка удалённых объектов используйте возвращающую набор строк функцию pg_event_trigger_dropped_objects() в триггере события sql_drop, триггер выполняется после удаления объектов из таблиц системного каталога, поэтому их невозможно больше увидеть.
table_rewrite	после ALTER TABLE и ALTER TYPE	Триггер сработает после того, как таблица будет перезаписана

Специальные переменные, которые могут использоваться в триггерах

Когда функция на PL/pgSQL срабатывает как триггер события, в блоке верхнего уровня автоматически создаются несколько специальных переменных:

TG_EVENT

Тип данных `text`. Строка, содержащая событие, по которому сработал триггер.

`TG_TAG`

Тип данных `text`. Переменная, содержащая тег команды, для которой сработал триггер.

Синтаксис

CREATE EVENT TRIGGER -- создать событийный триггер

```
CREATE EVENT TRIGGER имя
  ON событие
  [ WHEN переменная_фильтра IN (filter_value [, ... ]) [ AND ...
] ]
  EXECUTE PROCEDURE имя_функции()
```

ALTER EVENT TRIGGER — изменить определение событийного триггера

```
ALTER EVENT TRIGGER имя DISABLE
ALTER EVENT TRIGGER имя ENABLE [ REPLICA | ALWAYS ]
ALTER EVENT TRIGGER имя OWNER TO { новый_владелец | CURRENT_USER |
SESSION_USER }
ALTER EVENT TRIGGER имя RENAME TO новое_имя
```

`DISABLE/ENABLE [ REPLICA | ALWAYS ] TRIGGER`

Эти формы настраивают срабатывание событийных триггеров. Отключённый триггер сохраняется в системе, но не выполняется, когда происходит его событие срабатывания.

DROP EVENT TRIGGER — удалить событийный триггер

```
DROP EVENT TRIGGER [ IF EXISTS ] имя [ CASCADE | RESTRICT ]
```

Примеры:

1) Триггер, запрещающий выполнение любой команды DDL:

```
CREATE OR REPLACE FUNCTION abort_any_command()
  RETURNS event_trigger
```

```
 LANGUAGE plpgsql
  AS $$
BEGIN
  RAISE EXCEPTION 'команда % отключена', tg_tag;
END;
$$;

CREATE EVENT TRIGGER abort_ddl ON ddl_command_start
   EXECUTE PROCEDURE abort_any_command();
```

2) Триггер просто выдаёт сообщение всякий раз, когда выполняется поддерживаемая команда.

```
CREATE OR REPLACE FUNCTION snitch() RETURNS event_trigger AS $$
BEGIN
    RAISE NOTICE 'Произошло событие: % %', tg_event, tg_tag;
END;
$$ LANGUAGE plpgsql;

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

Поддержка триггеров событий командами DDL

Тег команды	ddl_command_start	ddl_command_end	sql_drop	table_rewrite	Замечания
ALTER AGGREGATE	X	X	-	-	
ALTER COLLATION	X	X	-	-	
ALTER CONVERSION	X	X	-	-	
ALTER DOMAIN	X	X	-	-	
ALTER EXTENSION	X	X	-	-	
ALTER FOREIGN DATA WRAPPER	X	X	-	-	
ALTER FOREIGN TABLE	X	X	X	-	
ALTER FUNCTION	X	X	-	-	
ALTER LANGUAGE	X	X	-	-	
ALTER OPERATOR	X	X	-	-	
ALTER OPERATOR CLASS	X	X	-	-	
ALTER OPERATOR FAMILY	X	X	-	-	
ALTER POLICY	X	X	-	-	
ALTER SCHEMA	X	X	-	-	
ALTER SEQUENCE	X	X	-	-	
ALTER SERVER	X	X	-	-	
ALTER TABLE	X	X	X	X	
ALTER TEXT SEARCH CONFIGURATION	X	X	-	-	
ALTER TEXT SEARCH DICTIONARY	X	X	-	-	
ALTER TEXT SEARCH PARSER	X	X	-	-	
ALTER TEXT SEARCH TEMPLATE	X	X	-	-	
ALTER TRIGGER	X	X	-	-	
ALTER TYPE	X	X	-	X	
ALTER USER MAPPING	X	X	-	-	
ALTER VIEW	X	X	-	-	
CREATE AGGREGATE	X	X	-	-	

COMMENT	X	X	-	-	Только для локальных объектов
CREATE CAST	X	X	-	-	
CREATE COLLATION	X	X	-	-	
CREATE CONVERSION	X	X	-	-	
CREATE DOMAIN	X	X	-	-	
CREATE EXTENSION	X	X	-	-	
CREATE FOREIGN DATA WRAPPER	X	X	-	-	
CREATE FOREIGN TABLE	X	X	-	-	
CREATE FUNCTION	X	X	-	-	
CREATE INDEX	X	X	-	-	
CREATE LANGUAGE	X	X	-	-	
CREATE OPERATOR	X	X	-	-	
CREATE OPERATOR CLASS	X	X	-	-	
CREATE OPERATOR FAMILY	X	X	-	-	
CREATE POLICY	X	X	-	-	
CREATE RULE	X	X	-	-	
CREATE SCHEMA	X	X	-	-	
CREATE SEQUENCE	X	X	-	-	
CREATE SERVER	X	X	-	-	
CREATE STATISTICS	X	X	-	-	
CREATE TABLE	X	X	-	-	
CREATE TABLE AS	X	X	-	-	
CREATE TEXT SEARCH CONFIGURATION	X	X	-	-	
CREATE TEXT SEARCH DICTIONARY	X	X	-	-	
CREATE TEXT SEARCH PARSER	X	X	-	-	

CREATE TEXT SEARCH TEMPLATE	X	X	-	-	
CREATE TRIGGER	X	X	-	-	
CREATE TYPE	X	X	-	-	
CREATE USER MAPPING	X	X	-	-	
CREATE VIEW	X	X	-	-	
DROP AGGREGATE	X	X	X	-	
DROP CAST	X	X	X	-	
DROP COLLATION	X	X	X	-	
DROP CONVERSION	X	X	X	-	
DROP DOMAIN	X	X	X	-	
DROP EXTENSION	X	X	X	-	
DROP FOREIGN DATA WRAPPER	X	X	X	-	
DROP FOREIGN TABLE	X	X	X	-	
DROP FUNCTION	X	X	X	-	
DROP INDEX	X	X	X	-	
DROP LANGUAGE	X	X	X	-	
DROP OPERATOR	X	X	X	-	
DROP OPERATOR CLASS	X	X	X	-	
DROP OPERATOR FAMILY	X	X	X	-	
DROP OWNED	X	X	X	-	
DROP POLICY	X	X	X	-	
DROP RULE	X	X	X	-	
DROP SCHEMA	X	X	X	-	
DROP SEQUENCE	X	X	X	-	
DROP SERVER	X	X	X	-	
DROP STATISTICS	X	X	X	-	
DROP TABLE	X	X	X	-	
DROP TEXT SEARCH CONFIGURATION	X	X	X	-	
DROP TEXT SEARCH DICTIONARY	X	X	X	-	
DROP TEXT SEARCH PARSER	X	X	X	-	
DROP TEXT SEARCH	X	X	X	-	

TEMPLATE					
DROP TRIGGER	X	X	X	-	
DROP TYPE	X	X	X	-	
DROP USER MAPPING	X	X	X	-	
DROP VIEW	X	X	X	-	
GRANT	X	X	-	-	Только для локальных объектов
IMPORT FOREIGN SCHEMA	X	X	-	-	
REVOKE	X	X	-	-	Только для локальных объектов
SECURITY LABEL	X	X	-	-	Только для локальных объектов
SELECT INTO	X	X	-	-	

Тема 14: SQL. Роли, привилегии и операторы для работы с ними

Цель работы: Получение практических навыков работы с СУБД и языком SQL (операторы create role, alter role, drop role, grant, revoke).

Задание:

1) Разработать в базе данных, созданной и заполненной на предыдущих лабораторных работах:

a. создайте две новых роли;
b. наделите первую роль привилегиями на часть таблиц;
c. назначьте второй роли первую в качестве роли;
d. отмените одну из привилегий;
e. изменить первую роль;
f. удалите вторую роль;
g. войдите под первой ролью и проверьте доступность привилегий.

Отчет по лабораторной работе должен содержать:

1. Фамилию и номер группы учащегося, задание.
2. Коды операций.
3. Принтскриты всех выполненных операторов.

Материал для выполнения

База данных содержит одну или несколько именованных *схем*, которые в свою очередь содержат таблицы. Схемы также содержат именованные объекты других видов, включая типы данных, функции и операторы. Одно и то же имя объекта можно свободно использовать в разных схемах, например и schema1, и myschema могут содержать таблицы с именем mytable. В отличие от баз данных, схемы не ограничивают доступ к данным: пользователь может обращаться к объектам в любой схеме текущей базы данных, если ему назначены соответствующие права.

Есть несколько возможных объяснений, для чего стоит применять схемы:

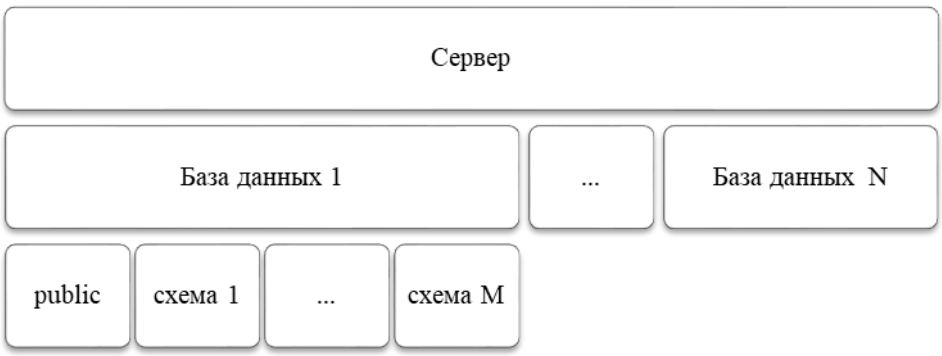
- Чтобы одну базу данных могли использовать несколько пользователей, независимо друг от друга.
- Чтобы объединить объекты базы данных в логические группы для облегчения управления ими.
- Чтобы в одной базе сосуществовали разные приложения, и при этом не возникало конфликтов имён.

Схемы в некоторым смысле подобны каталогам в операционной системе, но они не могут быть вложенными.

В PostgreSQL схема не связана с пользователем. Роли определяется не на уровне базы данных, а на уровне сервера (кластера).

По умолчанию пользователь не может обращаться к объектам в чужих схемах. Чтобы изменить это, владелец схемы должен дать пользователю право USAGE для данной схемы. Чтобы пользователи могли использовать объекты схемы, может понадобиться назначить дополнительные права на уровне объектов.

Пользователю также можно разрешить создавать объекты в схеме, не принадлежащей ему. Для этого ему нужно дать право CREATE в требуемой схеме. Заметьте, что по умолчанию все имеют права CREATE и USAGE в схеме public. Благодаря этому все пользователи могут подключаться к заданной базе данных и создавать объекты в её схеме public.



Роль — это сущность, которая может владеть объектами и иметь определённые права в базе; роль может представлять «пользователя», «группу» или и то, и другое, в зависимости от варианта использования.

PostgreSQL использует концепцию ролей (*roles*) для управления разрешениями на доступ к базе данных. Роль можно рассматривать как пользователя базы данных или как группу пользователей, в зависимости от того, как роль настроена. Роли могут владеть объектами базы данных (например, таблицами и функциями) и выдавать другим ролям разрешения на доступ к этим объектам, управляя тем, кто имеет доступ и к каким объектам. Кроме того, можно предоставить одной роли *членство* в другой роли, таким образом одна роль может использовать привилегии других ролей.

Концепция ролей включает в себя концепцию пользователей («users») и групп («groups»). До версии 8.1 в PostgreSQL пользователи и группы были отдельными сущностями, но теперь есть только роли. Любая роль может использоваться в качестве пользователя, группы, и того и другого.

Для добавления и удаления членов ролей, используемых в качестве групп, рекомендуется использовать GRANT и REVOKE.

Роль базы данных может иметь атрибуты, определяющие её полномочия и взаимодействие с системой аутентификации клиентов.

Для PostgreSQL: *CREATE ROLE = CREATE GROUP = CREATE USER*

Команда CREATE USER теперь является просто синонимом CREATE ROLE. Единственное отличие в том, что для команды, записанной в виде CREATE USER, по умолчанию подразумевается LOGIN, а в виде CREATE ROLE подразумевается NOLOGIN.

Оператор CREATE GROUP теперь является синонимом оператора CREATE ROLE.

Атрибуты роли

Право подключения

Только роли с атрибутом `LOGIN` могут использоваться для начального подключения к базе данных. Роль с атрибутом `LOGIN` можно рассматривать как пользователя базы данных. Для создания такой роли можно использовать любой из вариантов:

```
CREATE ROLE имя LOGIN;
CREATE USER имя;
```

Статус суперпользователя

Суперпользователь базы данных обходит все проверки прав доступа, за исключением права на вход в систему. Это опасная привилегия и она не должна использоваться небрежно. Лучше всего выполнять большую часть работы не как суперпользователь. Для создания нового суперпользователя используется `CREATE ROLE имя SUPERUSER`. Это нужно выполнить из под роли, которая также является суперпользователем.

Создание базы данных

Роль должна явно иметь разрешение на создание базы данных (за исключением суперпользователей, которые пропускают все проверки). Для создания такой роли используется `CREATE ROLE имя CREATEDB`.

Создание роли

Роль должна явно иметь разрешение на создание других ролей (за исключением суперпользователей, которые пропускают все проверки). Для создания такой роли используется `CREATE ROLE имя CREATEROLE`. Роль с привилегией `CREATEROLE` может также изменять и удалять другие роли, а также выдавать и отзывать членство в ролях. Однако, для создания, изменения, удаления суперпользовательских ролей, а также изменения в них членства, требуется иметь статус суперпользователя; привилегии `CREATEROLE` в таких случаях недостаточно.

Запуск репликации

Роль должна иметь явное разрешение на запуск потоковой репликации (за исключением суперпользователей, которые пропускают все проверки). Роль, используемая для потоковой репликации, также должна иметь атрибут `LOGIN`. Для создания такой роли используется `CREATE ROLE имя REPLICATION LOGIN`.

Пароль

Пароль имеет значение, если метод аутентификации клиентов требует, чтобы пользователи предоставляли пароль при подключении к базе данных. Методы аутентификации `password` и `md5` используют пароли. База данных и операционная система используют раздельные пароли. Пароль указывается при создании роли: `CREATE ROLE имя PASSWORD 'строка'`.

Предопределённые роли

Роль	Разрешаемый доступ

pg_read_all_settings	Читать все конфигурационные переменные, даже те, что обычно видны только суперпользователям.
pg_read_all_stats	Читать все представления pg_stat_* и использовать различные расширения, связанные со статистикой, даже те, что обычно видны только суперпользователям.
pg_stat_scan_tables	Выполнять функции мониторинга, которые могут устанавливать блокировки `ACCESS SHARE` в таблицах, возможно, на длительное время.
pg_signal_backend	Передавать сигналы другим обслуживающим процессам (например, отменять запрос, завершать процесс).
pg_read_server_files	Читать файлы в любом месте файловой системы, куда имеет доступ СУБД на сервере, выполняя COPY и другие функции работы с файлами.
pg_write_server_files	Записывать файлы в любом месте файловой системы, куда имеет доступ СУБД на сервере, выполняя COPY и другие функции работы с файлами.
pg_execute_server_program	Выполнять программы на сервере (от имени пользователя, запускающего СУБД), так же, как это делает команда COPY и другие функции, выполняющие программы на стороне сервера.
pg_monitor	Читать/выполнять различные представления и функции для мониторинга. Эта роль включена в роли `pg_read_all_settings`, `pg_read_all_stats` и `pg_stat_scan_tables`.

Синтаксис

CREATE ROLE — создать роль в базе данных

```
CREATE ROLE имя [ [ WITH ] параметр [ ... ] ]
Здесь параметр:
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT предел_подключений
```

```
    | [ ENCRYPTED ] PASSWORD 'пароль' | PASSWORD NULL
    | VALID UNTIL 'дата_время'
    | IN ROLE имя_роли [, ...]
    | IN GROUP имя_роли [, ...]
    | ROLE имя_роли [, ...]
    | ADMIN имя_роли [, ...]
    | USER имя_роли [, ...]
    | SYSID uid
```

ALTER ROLE — изменить роль в базе данных

```
ALTER ROLE указание_роли [ WITH ] параметр [ ... ]
Здесь параметр:
      SUPERUSER | NOSUPERUSER
    | CREATEDB | NOCREATEDB
    | CREATEROLE | NOCREATEROLE
    | INHERIT | NOINHERIT
    | LOGIN | NOLOGIN
    | REPLICATION | NOREPLICATION
    | BYPASSRLS | NOBYPASSRLS
    | CONNECTION LIMIT предел_подключений
    | [ ENCRYPTED ] PASSWORD 'пароль' | PASSWORD NULL
    | VALID UNTIL 'дата_время'
ALTER ROLE имя RENAME TO новое_имя
ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] SET
параметр_конфигурации { TO | = } { значение | DEFAULT }
ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] SET
параметр_конфигурации FROM CURRENT
ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] RESET
параметр_конфигурации
ALTER ROLE { указание_роли | ALL } [ IN DATABASE имя_бд ] RESET ALL
Здесь указание_роли:
    имя_роли
  | CURRENT_USER
  | SESSION_USER
```

SUPERUSER

NOSUPERUSER

Эти предложения определяют, будет ли эта роль «суперпользователем», который может переопределить все ограничения доступа в базе данных. Статус

суперпользователя несёт опасность и назначать его следует только в случае необходимости. Создать нового суперпользователя может только суперпользователь. В отсутствие этих предложений по умолчанию подразумевается NOSUPERUSER.

CREATEDB

NOCREATEDB

Эти предложения определяют, сможет ли роль создавать базы данных. Указание CREATEDB даёт новой роли это право, а NOCREATEDB запрещает роли создавать базы данных. По умолчанию подразумевается NOCREATEDB.

CREATEROLE

NOCREATEROLE

Эти предложения определяют, сможет ли роль создавать новые роли (т. е. выполнять CREATE ROLE). Роль с правом CREATEROLE может также изменять и удалять другие роли. По умолчанию подразумевается NOCREATEROLE.

INHERIT

NOINHERIT

Эти предложения определяют, будет ли роль «наследовать» права ролей, членом которых она является. Роль с атрибутом INHERIT может автоматически использовать в базе данных любые права, назначенные всем ролям, в которые она включена, непосредственно или опосредованно. Без INHERIT членство в другой роли позволяет только выполнить SET ROLE и переключиться на эту роль; правами, назначенными другой роли, можно будет пользоваться только после этого. По умолчанию подразумевается INHERIT.

LOGIN

NOLOGIN

Эти предложения определяют, разрешается ли новой роли вход на сервер; то есть, может ли эта роль стать начальным авторизованным именем при подключении клиента. Можно считать, что роль с атрибутом LOGIN соответствует пользователю. Роли без этого атрибута бывают полезны для управления доступом в базе данных, но это не пользователи в обычном понимании.

REPLICATION

NOREPLICATION

Эти предложения определяют, будет ли роль ролью репликации. Чтобы роль могла подключаться к серверу в режиме репликации (в режиме физической или логической репликации) и создавать/удалять слоты репликации, у неё должен быть

этот атрибут (либо это должна быть роль суперпользователя). Роль, имеющая атрибут REPLICATION, обладает очень большими привилегиями и поэтому этот атрибут должны иметь только роли, фактически используемые для репликации. По умолчанию подразумевается вариант NOREPLICATION.

BYPASSRLS

NOBYPASSRLS

Эти предложения определяют, будут ли для роли игнорироваться все политики защиты на уровне строк (RLS). Значение по умолчанию — NOBYPASSRLS. Заметьте, что pg_dump по умолчанию отключает row_security (устанавливает значение OFF) для уверенности, что выгружено всё содержимое таблицы. Если пользователь, запускающий pg_dump, не будет иметь необходимых прав, он получит ошибку. Суперпользователь и владелец выгружаемой таблицы всегда обходят защиту RLS.

CONNECTION LIMIT *предел_подключений*

Если роли разрешён вход, этот параметр определяет, сколько параллельных подключений может установить роль. Значение -1 (по умолчанию) снимает ограничение. Заметьте, что под это ограничение подпадают только обычные подключения. Ни подготовленные транзакции, ни соединения фоновых рабочих процессов в расчёт не берутся.

[ENCRYPTED] PASSWORD '*пароль*'

PASSWORD NULL

Задаёт пароль роли. (Пароль полезен только для ролей с атрибутом LOGIN, но задать его можно и для ролей без такого атрибута.) Если проверка подлинности по паролю не будет использоваться, этот параметр можно опустить. При указании пустого значения будет задан пароль NULL, что не позволит данному пользователю пройти проверку подлинности по паролю. При желании пароль NULL можно установить явно, указав PASSWORD NULL.

VALID UNTIL '*дата_время*'

Предложение VALID UNTIL устанавливает дату и время, после которого пароль роли перестаёт действовать. Если это предложение отсутствует, срок действия пароля будет неограниченным.

IN ROLE *имя_роли*

В предложении IN ROLE перечисляются одна или несколько существующих ролей, в которые будет немедленно включена новая роль. (Заметьте, что добавить

новую роль с правами администратора таким образом нельзя; для этого надо отдельно выполнить команду GRANT.)

IN GROUP *имя_роли*

IN GROUP — устаревшее написание предложения IN ROLE.

ROLE *имя_роли*

В предложении ROLE перечисляются одна или несколько существующих ролей, которые автоматически становятся членами создаваемой роли. (По сути таким образом новая роль становится «группой».)

ADMIN *имя_роли*

Предложение ADMIN подобно ROLE, но перечисленные в нём роли включаются в новую роль с атрибутом WITH ADMIN OPTION, что даёт им право включать в эту роль другие роли.

USER *имя_роли*

Предложение USER является устаревшим написанием предложения ROLE.

SYSID *uid*

Предложение SYSID игнорируется, но принимается для обратной совместимости.

DROP ROLE — удалить роль в базе данных

```
DROP ROLE [ IF EXISTS ] имя [, ...]
```

SET ROLE — установить идентификатор текущего пользователя в рамках сеанса

```
SET [ SESSION | LOCAL ] ROLE имя_роли
SET [ SESSION | LOCAL ] ROLE NONE
RESET ROLE
```

SET SESSION AUTHORIZATION — установить идентификатор пользователя сеанса и идентификатор текущего пользователя в рамках сеанса

```
SET [ SESSION | LOCAL ] SESSION AUTHORIZATION имя_пользователя
SET [ SESSION | LOCAL ] SESSION AUTHORIZATION DEFAULT
RESET SESSION AUTHORIZATION
```

Примеры

1) Создание роли, для которой разрешён вход, но не задан пароль:

```
CREATE ROLE jonathan LOGIN;
```

2) Создание роли с паролем:

```
CREATE USER davide WITH PASSWORD 'jw8s0F4';
```

3) Создание роли с паролем, действующим до конца 2004 г., то есть пароль перестаёт действовать в первую же секунду 2005 г.

```
CREATE ROLE miriam WITH LOGIN PASSWORD 'jw8s0F4' VALID UNTIL '2005-01-01';
```

4) Создание роли, которая может создавать базы данных и управлять ролями:

```
CREATE ROLE admin WITH CREATEDB CREATEROLE;
```

5) Изменение пароля роли:

```
ALTER ROLE davide WITH PASSWORD 'hu8jmn3';
```

6) Удаление пароля роли:

```
ALTER ROLE davide WITH PASSWORD NULL;
```

7) Изменение срока действия пароля (в частности, определяется, что пароль должен перестать действовать в полдень 4 мая 2015 г. в часовом поясе UTC+1):

```
ALTER ROLE chris VALID UNTIL 'May 4 12:00:00 2015 +1';
```

8) Установка бесконечного срока действия пароля:

```
ALTER ROLE fred VALID UNTIL 'infinity';
```

9) Наделение роли правами на создание других ролей и новых баз данных:

```
ALTER ROLE miriam CREATEROLE CREATEDB;
```

10) Удаление роли:

```
DROP ROLE jonathan;
```

Привилегии (права)

Когда создаётся любой объект, ему назначается владелец. Обычно владелец — это та роль, которая запустила оператор создания данного объекта. Для большинства видов объектов, начальное состояние таково, что делать что-либо с объектом может только владелец (или суперпользователь). Чтобы разрешить другим ролям использовать его, им должны быть предоставлены привилегии.

Эти привилегии применяются к отдельному объекту, в зависимости от типа объёкта (таблица, функция и т.д.)

Объекту может быть назначен новый владелец с помощью команды ALTER для соответствующего вида объекта, например, ALTER TABLE. Суперпользователи могут делать это всегда; обычные роли могут делать это только если они одновременно и являются текущим владельцем данного объекта (или членом роли текущего владельца) и членом роли нового владельца.

Список привилегий

`SELECT`

Позволяет выполнять SELECT для любого столбца или перечисленных столбцов в заданной таблице, представлении или последовательности. Также позволяет выполнять COPY TO. Помимо того, это право требуется для обращения к существующим значениям столбцов в UPDATE или DELETE.

`INSERT`

Позволяет вставлять строки в заданную таблицу с помощью INSERT. Если право ограничивается несколькими столбцами, только их значение можно будет задать в команде `INSERT` (другие столбцы получат значения по умолчанию). Также позволяет выполнять COPY FROM.

`UPDATE`

Позволяет изменять (с помощью UPDATE) данные во всех, либо только перечисленных, столбцах в заданной таблице.

`DELETE`

Позволяет удалять строки из заданной таблицы с помощью DELETE.

`TRUNCATE`

Позволяет опустошить заданную таблицу с помощью TRUNCATE.

`REFERENCES`

Позволяет создавать ограничение внешнего ключа, ссылающееся на определённую таблицу либо на определённые столбцы таблицы.

`TRIGGER`

Позволяет создавать триггеры в заданной таблице.

`CREATE`

Для баз данных это право позволяет создавать схемы и публикации в заданной базе.

`CONNECT`

Позволяет пользователю подключаться к указанной базе данных.

`TEMPORARY`

`TEMP`

Позволяет создавать временные таблицы в заданной базе данных.

`EXECUTE`

Позволяет выполнять заданную функцию или процедуру и применять любые определённые поверх неё операторы. Это единственный тип прав, применимый к функциям и процедурам.

`USAGE`

Для процедурных языков это право позволяет создавать функции на заданном языке. Это единственный тип прав, применимый к процедурным языкам.

`ALL PRIVILEGES`

Даёт целевой роли все права сразу.

Синтаксис

GRANT — определить права доступа

Для таблицы

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER
}
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] имя_таблицы [, ...]
         | ALL TABLES IN SCHEMA имя_схемы [, ...] }
    TO указание_роли [, ...] [ WITH GRANT OPTION ]
```

Для столбца

```
GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( имя_столбца [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( имя_столбца [, ...] ) }
    ON [ TABLE ] имя_таблицы [, ...]
    TO указание_роли [, ...] [ WITH GRANT OPTION ]
```

Для функции или процедуры

```
GRANT { EXECUTE | ALL [ PRIVILEGES ] }
    ON { { FUNCTION | PROCEDURE | ROUTINE } имя_подпрограммы [ ( [ [
режим_аргумента ] [ имя_аргумента ] тип_аргумента [, ...] ] ) ] [, ...]
         | ALL { FUNCTIONS | PROCEDURES | ROUTINES } IN SCHEMA имя_схемы [,
...] }
    TO указание_роли [, ...] [ WITH GRANT OPTION ]
```

Здесь *указание_роли*:

```
  [ GROUP ] имя_роли
  | PUBLIC
  | CURRENT_USER
  | SESSION_USER
```

Для назначения роли другой роли (пользователю)

```
GRANT имя_роли [, ...] TO имя_роли [, ...] [ WITH ADMIN OPTION ]
```

REVOKE — отозвать права доступа

Для таблицы

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE | REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] имя_таблицы [, ...]
         | ALL TABLES IN SCHEMA имя_схемы [, ...] }
    FROM { [ GROUP ] имя_роли | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Для столбца

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | REFERENCES } ( имя_столбца [, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( имя_столбца [, ...] ) }
    ON [ TABLE ] имя_таблицы [, ...]
    FROM { [ GROUP ] имя_роли | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Для функции или процедуры

```
REVOKE [ GRANT OPTION FOR ]
    { EXECUTE | ALL [ PRIVILEGES ] }
    ON { { FUNCTION | PROCEDURE | ROUTINE } имя_функции [ ( [ [
режим_аргумента ] [ имя_аргумента ] тип_аргумента [, ...] ] ) ] [, ...]
         | ALL { FUNCTIONS | PROCEDURES | ROUTINES } IN SCHEMA имя_схемы [,
...] }
    FROM { [ GROUP ] имя_роли | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Для роли

```
REVOKE [ ADMIN OPTION FOR ]
    имя_роли [, ...] FROM имя_роли [, ...]
    [ CASCADE | RESTRICT ]
```

ALTER DEFAULT PRIVILEGES — определить права доступа по умолчанию

```
ALTER DEFAULT PRIVILEGES
    [ FOR { ROLE | USER } целевая_роль [, ...] ]
    [ IN SCHEMA имя_схемы [, ...] ]
    предложение_GRANT_или_REVOKE
```

Примеры

1) Следующая команда разрешает всем добавлять записи в таблицу `films`:

```
GRANT INSERT ON films TO PUBLIC;
```

2) Эта команда даёт пользователю `manuel` все права для представления `kinds`:

```
GRANT ALL PRIVILEGES ON kinds TO manuel;
```

3) Включение в роль `admins` пользователя `joe`:

```
GRANT admins TO joe;
```

4) Лишение группы public права добавлять данные в таблицу `films`:

```
REVOKE INSERT ON films FROM PUBLIC;
```

5) Лишение пользователя `manuel` всех прав для представления `kinds`:

```
REVOKE ALL PRIVILEGES ON kinds FROM manuel;
```

6) Исключение из членов роли `admins` пользователя `joe`:

```
REVOKE admins FROM joe;
```

7) Наделение всех правом SELECT для всех таблиц (и представлений), которые будут созданы в дальнейшем в схеме `myschema`, и наделение роли `webuser` правом INSERT для этих же таблиц:

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT SELECT ON TABLES TO PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema GRANT INSERT ON TABLES TO
webuser;
```

8) Отмена предыдущих изменений с тем, чтобы для таблиц, создаваемых в будущем, были определены только обычные права, без дополнительных:

```
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE SELECT ON TABLES FROM
PUBLIC;
ALTER DEFAULT PRIVILEGES IN SCHEMA myschema REVOKE INSERT ON TABLES FROM
webuser;
```

9) Лишение роли public права на выполнение (EXECUTE), которое обычно даётся для функций (для всех функций, которые будут созданы ролью `admin`):

```
ALTER DEFAULT PRIVILEGES FOR ROLE admin REVOKE EXECUTE ON FUNCTIONS FROM
PUBLIC;
```

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ

Список рекомендуемой литературы

основная

1) Советов, Б. Я. Базы данных : учебник для прикладного бакалавриата / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2019. — 420 с. — (Бакалавр. Прикладной курс). — ISBN 978-5-534-07217-4. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://www.biblio-online.ru/bcode/431947
2) Латыпова Р.Р., Базы данных. Курс лекций: учебное пособие [Электронный ресурс] / Латыпова Р.Р. - М. : Проспект, 2016. - 96 с. - ISBN 978-5-392-19240-3 - Режим доступа: http://www.studentlibrary.ru/book/ISBN9785392192403.html
3) SQL / Фиайли К. - М. : ДМК Пресс, 2008. - ISBN 5-94074-233-5 - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : https://www.studentlibrary.ru/book/ISBN5940742335.html (дата обращения: 11.11.2020). - Режим доступа : по подписке.

дополнительная

4) Стружкин, Н. П. Базы данных: проектирование. Практикум : учебное пособие для академического бакалавриата / Н. П. Стружкин, В. В. Годин. — Москва : Издательство Юрайт, 2019. — 291 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-00739-8. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://www.biblio-online.ru/bcode/433865
5) Редмонд Э., Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL / Эрик Редмонд, Джим. Р. Уилсон ; Пер. с англ. Слинкин А.А. - М. : ДМК Пресс, 2013. - 384 с. - ISBN 978-5-94074-866-3 - Текст : электронный // ЭБС "Консультант студента" : [сайт]. - URL : https://www.studentlibrary.ru/book/ISBN9785940748663.html (дата обращения: 11.11.2020). - Режим доступа : по подписке.

учебно-методическая

6) SQL-запросы: учеб.-метод. пособие / Кондратьев Алексей Евгеньевич, О. А. Фатьянова; Ульяновск. гос. ун-т, ФМиИТ. - Ульяновск : УлГУ, 2009. URL: ftp://10.2.96.134/Text/Kondratiev_SQL.pdf

Программное обеспечение

1. Open System Architect (open source),
2. СУБД PostgreSQL (open source),
3. pgAdmin4 (open source).