МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

Современные реляционные базы данных.

Методические указания к выполнению лабораторных работ

Санкт-Петербург

2024

Составитель: Н.В. Путилова, Величко М.В.

Рассмотрены вопросы проектирования, разработки и серверного программирования реляционных и объектно-реляционных баз данных на примере MySQL и PostgreSQL. Приведено много примеров написания различных запросов и процедур для соответствующих СУБД. Также рассмотрено назначение СУБД других моделей данных. Учебно-методическое пособие предназначено для студентов всех форма обучения направлений 02.03.03 и 09.03.04, изучающих дисциплину «Проектирование баз данных». В пособие включены краткие теоретические сведения, необходимые для выполнения лабораторных работ.

Содержание

Введение

Большая часть современных приложений и программных комплексов имеет базы данных в своем составе, зачастую различных моделей и назначения. Часто можно встретить реляционную базу данных в качестве основного хранилища, ключ-значение для кэширования запросов сервера и столбцовую базу для сложных аналитических работ в одном приложении. Современному специалисту в области программной инженерии необходимо понимать особенности проектирования и работы с современными базами данных. Поэтому данное пособие актуально для изучения обучающимися по направлениям, связанным с программной инженерией.

В учебно-методическом пособии основное внимание уделено вопросам проектирования и работы с реляционными базами данных. Приведена методика анализа пользовательских историй предметной области для построения концептуальной модели. Приведены примеры написания запросов с использованием различных техник. Разобрано серверное программирование в, представленное хранимыми процедурами, функциями и триггерами для СУБД MySQL и PostgreSQL. Приведены примеры управления доступа к данным на основе ролей и пользовательских привилегий. Учебно-методическое пособие состоит из 10 разделов, в каждом из которых теоретическая часть дополнена большим количеством практических примеров. Для получения умений и навыков обучающимся предложено 8 лабораторных работ. В первых 4 разделах рассматривается проектирование баз данных соответственно его этапы в общем, в первом разделе и концептуальное, логическое и физическое проектирование в последующих. Пятый раздел посвящен использованию языка определения данных в SQL. В разделах с шестого по восьмой показаны разные аспекты манипулирования данными в реляционных базах данных. Девятый и десятый разделы раскрывают особенности серверного программирования и управления доступом MySQL и PostgreSQL. Представленные в библиографическом списке материалы позволят студентам получить более глубокие знания и навыки в области современных баз данных.

Этапы проектирования баз данных

Выделяют следующие этапы проектирования баз данных[1,2]:

1. **Концептуальное проектирование.** Концептуальное проектирование — это процесс создания модели используемой информации, независящей от любых физических аспектов ее представления.
2. **Логическое проектирование.** Логическое проектирование— это процесс создания модели используемой информации на основе выбранной модели организации данных, но без учета конкретной целевой СУБД и других физических аспектов реализации.
3. **Физическое проектирование.** Описание конкретной реализации базы данных, размещаемой во внешней памяти, определяет организацию файлов и состав индексов.

Под понятие модели данных в данном случае рассматривается абстрактное, независимое, логическое определение структур данных. Таким образом, существует одна концептуальная модель предметной области и много логических моделей данных для этой концептуальной, по количеству существующих моделей данных для СУБД, а физических для данной логической может быть минимум столько, сколько конкретных СУБД для данной модели данных. При этом для каждой СУБД по одной логической модели можно задать много различных параметров хранения. Общие соотношения моделей на разных этапах проектирования можно рассмотреть на рис. 1.

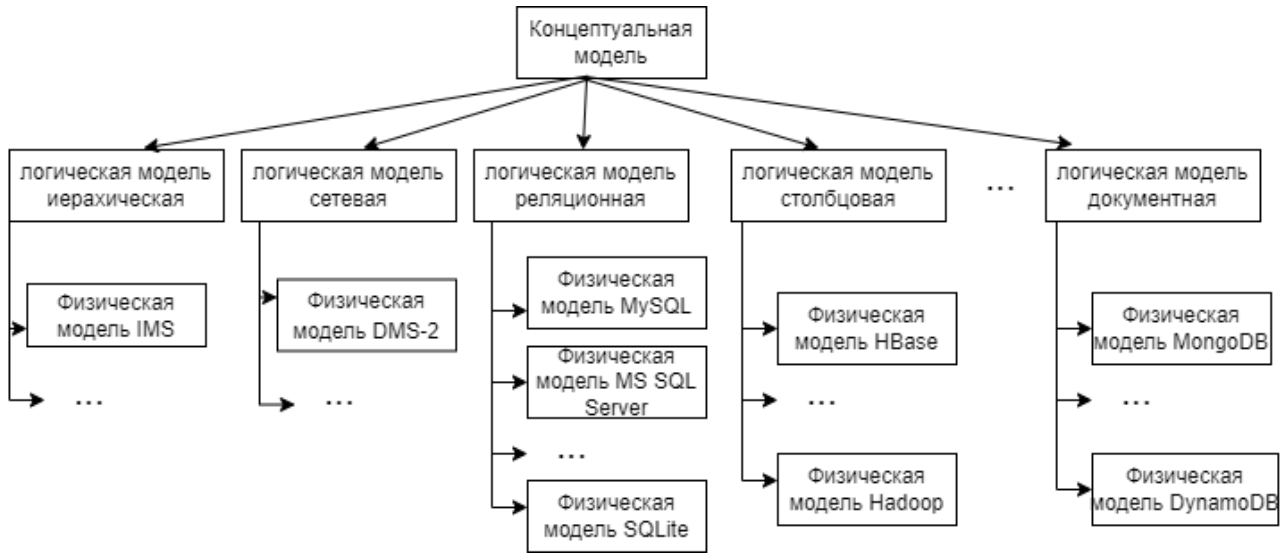


Рис.1. Соотношение моделей на различных этапах проектирования

Концептуальное проектирование баз данных

Модель сущность-связь

Для моделирования данных в парадигме концептуальной методологии используется модель «сущность–связь» (или ER-модель, от английского «Entity-Relationship»), разработанная Питером Ченом в 1976г[3]. ER-модель позволяет формализовать структуру и отношения между информационными объектами рассматриваемой предметной области. Все предметы реального мира обладают различными свойствами: машина характеризуется размером, весом, номерным знаком; у человека есть имя, фамилия, адрес и т. д. При разработке модели предмет или класс предметов рассматривается как некоторая **сущность**, а элементы данных, которые описывают свойства предметов, – как **атрибуты** сущностей. Между сущностями устанавливаются **связи**, представляющие в модели отношения между объектами реального мира. Базовыми элементами модели являются понятия атрибута, сущности (еще называемым **типами сущности**) и связи.

Атрибут –элемент информации, некоторую характеристику объекта.

Сущность описывается набором атрибутов. Каждый атрибут описывает отдельное свойство сущности. Например, для объекта «КНИГА» атрибутами будут фамилия автора, год издания, место издания и т. д.

Тип сущности. (часто просто сущность**)**Группа объектов с одинаковыми свойствами, которая рассматривается в конкретной предметной области как имеющая независимое существование.

Экземпляр сущности. Однозначно идентифицируемый объект, который относится к сущности определенного типа.

Каждому типу сущности в ER-модели присваивается имя. Поскольку имя представляет класс или множество объектов, то для обозначения типа сущности используется простое существительное в единственном числе[4].

Связь описывает соединение между данными (типами сущностей)

Все сущности или связи одного и того же типа обладают определенными общими свойствами. Сущности подразделяются на обычные (сильные) и слабые сущности. **Слабой** называется такая сущность, существование которой зависит от другой сущности, т.е. она не может существовать, если этой другой сущности не существует. **Обычной(сильной)**называется сущность, которая не является слабой. **Пример: Студент** (№ зачетки, ФИО, дата рождения) –сильная сущность, личный кабинет студента (ИД, логин, пароль, дата создания) –слабая сущность, не существует без студента.

Характеристики свойств:

Простое или **составное** свойство. Например, свойство "имя работника" может быть составным, если его значение составляется из значений простых свойств "имя", "отчество" и "фамилия".

Ключевое свойство или **неключевое** свойство (Ключевое свойство—свойство, которое может оказаться уникальным только в определенном контексте). Например, Фамилия и имя студента обычно бывает уникальным только в контексте данных о группе, в которой он учится.

В качестве ключа может использоваться:

- **Естественный Ключ (ЕК)** – набор атрибутов описываемой записью сущности, уникально её идентифицирующий (например, номер паспорта для человека).
- **Суррогатный Ключ (СК)** – автоматически сгенерированное поле, никак не связанное с информационным содержанием записи.

Однозначное или **многозначное** свойство (т.е. в этой модели допускаются повторяющиеся группы). Но если некоторый человек будет иметь несколько разных электронных почт, то свойство логин на сайте для него может быть многозначным. [5]

Отсутствующее (необязательное) свойство (т.е. свойство *неизвестное* или *неприменимое* для некоторых экземпляров сущности)и **обязательное**.

Базовое или **производное** свойство. Например, общее количество студентов в определенной группе может быть вычислено с помощью суммирования подсчета отдельных студентов данной группы (дата рождения –базовое, возраст производное).

Связи. Название связи представляется в глагольной форме и описывает отношение между 2 или более сущностями. Сущности, включенные в связь, называются ее участниками, а количество участников связи называется ее **степенью** [5].Выделяют следующие типы связей по **Мощности** (кардинальности):"один-ко-многим" (1:M) (1:*),"многие-ко-многим" (M:N) (*:*), "один-к-одному" (1:1).

Описание типов мощностей представлено в таблице 1.

Таблица 1. Описание типов мощности связи

Мощность	Объяснение	Пример
"один-ко-многим" (1:M) (1:*)	Один экземпляр сущности 1 связан с множеством экземпляров сущности 2, но экземпляр сущности 2, связан только с одним экземпляром сущности 1	Человек - мобильный телефон. У человека может быть несколько номеров, но номер (сим-карта) зарегистрирован только на одного
"многие-ко-многим" (M:N) (*:*)	Один экземпляр сущности 1 связан с множеством экземпляров сущности 2, и экземпляр сущности 2 связан с многими экземплярами сущности 1	Студент- кружки по интересам. Студент может ходить в несколько кружков, и в один кружок ходит несколько студентов

"один-к-одному" (1:1)	Один экземпляр сущности 1 связан только с одним экземпляром сущности 2 и экземпляр сущности 2, связан только с одним экземпляром сущности 1	Студент- личный кабинет. У студента только один личный кабинет студента и личный кабинет строго привязан только к одному студенту

В расширенной ER-модели также выделяют подтипы и супертипы сущностей. Где супертип — это сущность, объединяющая разные подтипы, которые необходимо представить в модели данных. Подтип — это сущность, являющаяся членом супертипа, но выполняющая отдельную роль в нем.

Диаграмма сущность-связь (ER-диаграмма)

Для документального оформления концептуальных моделей используется ER-диаграмма — граф-схема специального вида, представляющая сущности (именованные узлы графа) и связи между ними (именованные дуги графа, помеченные специальными символами).[6]

Несмотря на единую модель, существует несколько различных нотаций для отображения диаграмм. Среди основных нотаций можно выделить следующие:

- Нотация П. Чена (Peter Chen Notation)
- Нотация Баркера (Barker Notation)
- Нотация Птичья лапка (Crow’sFeet), представляющая собойсоединение нотаций IE (Information Engineering) (Дж. Мартина (James Martin) иК. Финкельштейна (Clive Finkelstein))-
- Нотация Ч. Бахмана
- НотацияIDEF1X (Integration Definition for Information Modeling)

Наиболее часто используются2нотации: модель Чена и модель «птичья лапка». В модели птичья лапка сущность представлена прямоугольника со вложенными описаниями атрибутов, а связи линиями, окончания которых визуально показывают мощность.

Основные элементы описания сущностей в нотации «птичья лапка» представлены на рис.2.

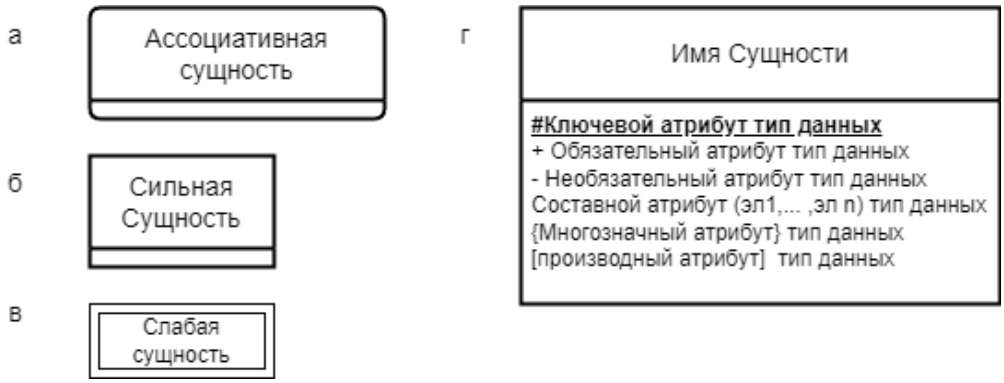


Рис.2.Отображение сущностей и атрибутов в нотации «птичья лапка»: а-ассоциативная сущность, б - сильная сущность, в - слабая сущность, г-описание различных атрибутов сущности

Ассоциативная сущность используется для раскрытия связи M:M или связей высоких степеней

Для отображения связей могут использоваться различные окончания линии связи, обозначающие множественность и обязательность связи. Всего 4 комбинации: (1 и только 1), необязательная один (0 или 1), обязательная много (1 или много), необязательная много (0 или много). Они представлены на рис. 3.

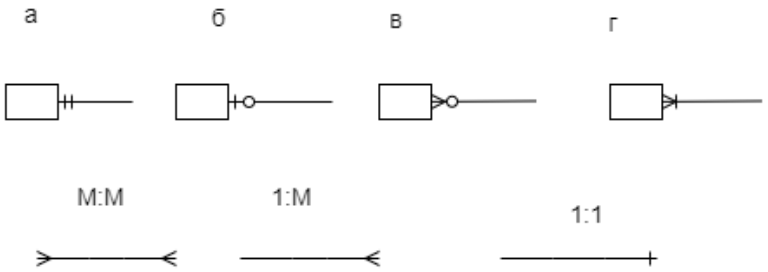


Рис. 3. Отображение типов связей в нотации «птичья лапка»: а – обязательная один б –1и только 1, обязательная связь, в–необязательная связь много, г –обязательная связь много

В нотации Чена сущность также изображена прямоугольником, но атрибуты расположена в овалах вокруг неё. Основные элементы описания сущностей в нотации Чена представлены на рис.4

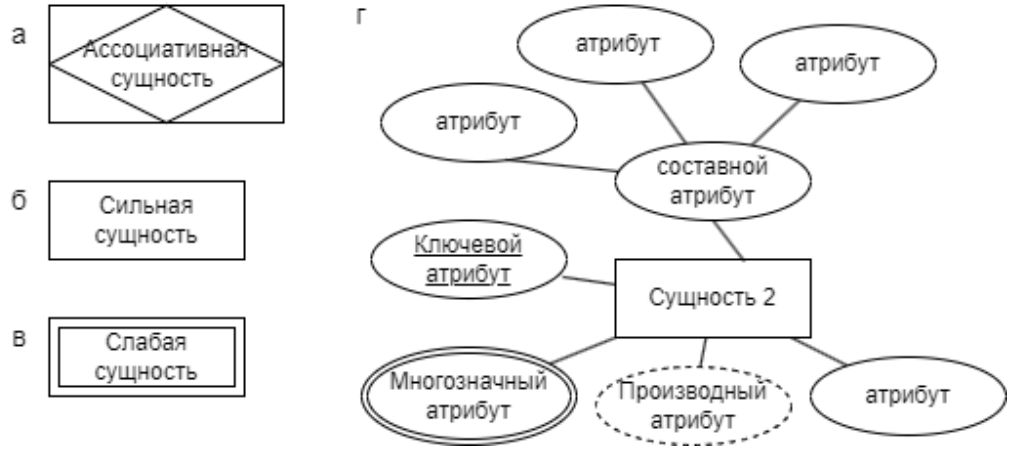


Рис. 4.Отображение сущностей и атрибутов в нотации Чена а- ассоциативная сущность, б - сильная сущность, в - слабая сущность, г-описание различных атрибутов сущности

Связи отображаются ромбом, где мощность прописана числом, а обязательность обозначена штрихом, как на рис. 5.

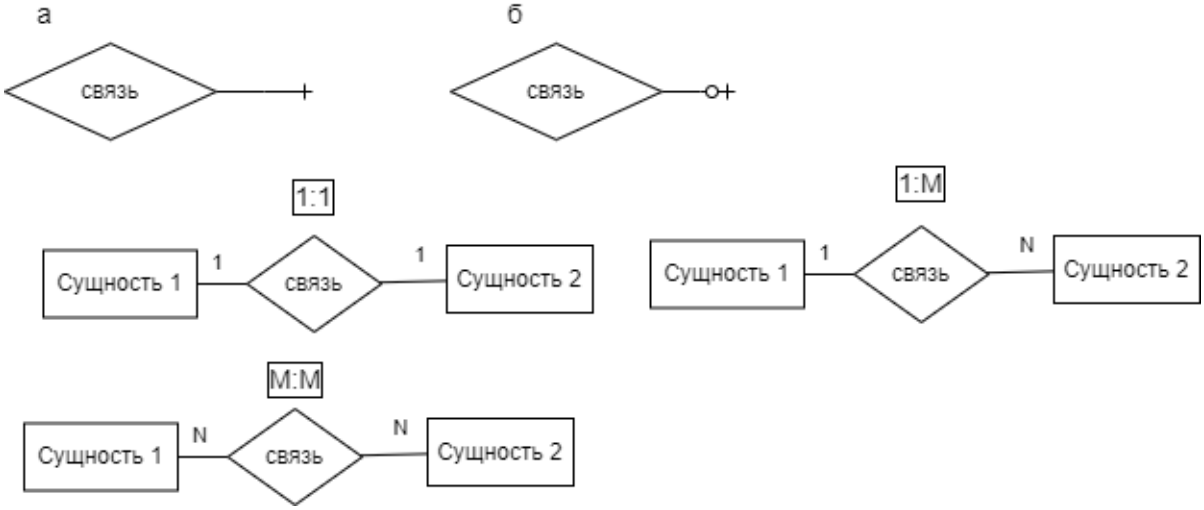


Рис. 5. Отображение типов связей в нотации «птичья лапка»: а – обязательная б –не обязательная связь

Для работы с ER-диаграммами существует достаточно много средств, среди них встречаются CASE-средства (например, DB Designer, Oracle SQL Developer Data Modeler) и универсальные средства создания диаграмм (Draw.io, Visio, Lucidchart)

Правила создания диаграммы сущность-связь.

- Названия сущностей уникальны в рамках диаграммы
- Названия атрибутов (свойств) уникальны в рамках диаграммы

- Каждая сущность должна иметь ключ (ключевые атрибуты)

Методика анализа предметной области

Шаги при анализе предметной области:

1. Выделение из описания задания (технического задания, пользовательских историй, требований и т.д.) основных понятий предметной области и их значений. Понятия потом будут разделены на атрибуты и сущности
2. Определение связей между понятиями. Все понятия выписываем и соединяем линиями (или как-либо обозначаем связь) те, что логически непосредственно связаны.

Все связи, которые определили, являются или связью атрибута с его сущностью или связью, или связь между 2 и более сущностями. Обратите внимание, что связь может быть и у понятия с самим собой, и между 2 понятиями, и между большим числом понятий.

3. Определение мощности связи с этапа 2.

Это определение, с каким максимальным количеством экземпляров сущности на других концах связи может быть связан экземпляр заданной и наоборот.

4. Определение является ли понятие атрибутом или сущностью.

Для этого используются вопросы:

1)Существует ли самостоятельно. Если да – сущность. Если нет – не определено, так как может быть атрибутом и слабой сущностью.

2)Есть ли собственные свойства (атрибуты)? Если да – сущность. Если нет – не определено. Наименование чего-либо уже может быть свойством.

3) Участвует ли в связи многие-ко-многим или один-ко-многим со стороны многие? Если да– сущность.

5. Для каждого атрибута определить его принадлежность.

Если на этапе 2 атрибут был определен как связанный с одной сущностью, то это атрибут этой сущности. А если он связан с 2 или более понятиями(сущностями), то это атрибут связи этих сущностей (если она есть) или отражающей их связь ассоциированной сущности.

6. Доопределение атрибутов.

Возможно, выделенным сущностям придется добавлять атрибуты, не указанные в описании предметной области, так как многие вещи подразумеваются. Например, когда речь идет о сотруднике, студенте или человеке, то в явном виде редко указывается необходимость хранения полного имени. При наличии возможности о дополнительных атрибутах необходимо опросить специалистов предметной области и заказчиков.

7. Определение типов данных

На этапе концептуального проектирования достаточно определить дата или время, текст или число (целое или число с плавающей/фиксированной запятой).

Определение обязательности связей и атрибутов.

Обязательный атрибут— такой, который всегда будет заполнен у всех экземпляров сущности.

Пример анализа предметной области

Несмотря на то, что проектировать концептуальную модель можно на любом удобном языке в данном случае итоговый вариант будем рассматривать на английском. Это связано с удобством работы в СУБД с таблицами на английском языке.

Задание: База данных для управления секциями(кружками)студентов: секции, руководитель секций, дата вступления студента в секцию…

1. Выделение из описания задания основных понятий

База данных для управления секциями(кружками)студентов: **секции**, **руководитель секций**, **дата вступления студента** в **секцию**

Планируемые запросы

а. Выбрать **группы**, **студенты** которых посещают **секции**, в **названии** которых есть слово «*программирован*», но оно не последнее
б. Найти **группы** без **студентов**
в. Найти **преподавателя**, который ведет(руководит)**секцию** по *авиамоделированию* и еще одну.
г. Найти **группу**, совокупность **студентов** которой посещает все **секции** по *программированию*

д. Найти **секцию**, которая проводит **занятия** первой в ***этом месяце***.
е. **Преподавателя**, к которому в **секции** ходят только **студенты группы***4231*
ж. Найти **группу** с максимальным количеством **студентов**

Полужирным шрифтом выделены понятия, курсивом- значения. «Этот месяц» выделен полужирным курсивом, потому что и является отсылкой к значению, и говорит о наличии даты занятия.

Перечень понятий: секция, преподаватель-руководитель секции, дата вступления (студента в секцию), группа, студент, название секции, занятие, дата занятия. Проверяем связаны ли эти данные.

2. Определение связей между понятиями.

Название секции связно с секцией? (Настолько с ней). Дата занятия связана с занятием? (Да, только с занятием). Дата вступления связана с секцией? (Да). Дата вступления связана со студентом? (Да, только с секцией и студентом). Связан ли преподаватель и секция? (Да, руководит). Связан ли преподаватель со студентом? (Напрямую – нет.) Связан ли преподаватель с группой (Напрямую – нет.) Связан ли преподаватель с занятием? (Да, может проводить его). Связана ли секция со студентом? (Да, он в неё может ходить). Связана ли секция с занятием? (Да, занятие – встреча секции.) Связана ли секция с группой? (Напрямую нет, только через студента) Связано ли занятие с группой (Нет, секции не групповые). Связано ли занятие со студентом? (Наличие связи зависит от предметной области. Если мы хотим отмечать посещаемость секции, то связано. А если посещаемость нас не интересует, то связью можно пренебречь.). Результат отображения связей между понятиями представлен на рис. 6.

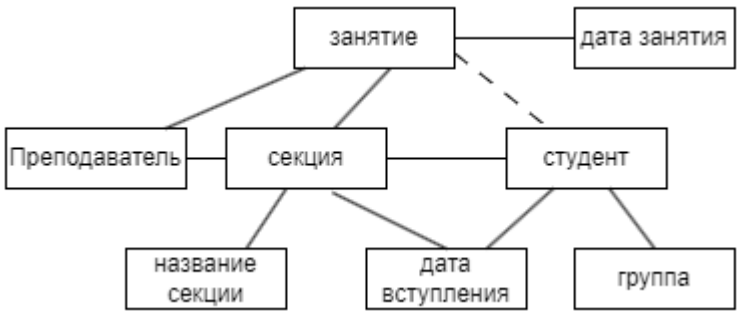


Рис.6. Шаг 2 концептуального проектирования: связи между понятиями.

Поскольку запросов, касающихся посещаемости секций нет, то связь между занятием и студентом в дальнейшем показывать не будем.

3. Определение мощности связи с этапа 2.

У секции может быть только одно название и в рамках одного вуза названия секций будут уникальны (принадлежать только 1 секции) — связь 1:1.

У студента может быть много секций и в одну секцию может ходить много студентов — связь М: М.Дат вступления для одной секции будет много, но в конкретную секцию можно записаться одному студенту только 1 раз (дата вступления конкретного студента в конкретную секцию) — связь 1: М. Дат вступления для одного студента будет много (в разные секции), но в конкретную секцию можно записаться только 1 раз — связь 1: М. У секции может быть много занятий, но одно конкретное только по одной секции— связь 1: М. У одного конкретного занятия, только один день(дата), но в этот день могут быть занятия у многих секций — связь 1: М. У секции один и только один руководитель, но преподаватель может руководить многими секциями — связь 1:М. Преподаватель может вести много занятий и теоретически одно занятие может вести сразу несколько преподавателей — связь М:М. Со связью группа-студент немного сложнее. Если мы берем данные с историей, то в разное время студент может учиться в разных группах (хотя бы бакалавриат и магистратура), а если берем текущие данные, то только в одной. В данном случае историей можно пренебречь, как незначительной для предметной области. В группе много студентов и студент учится только в одной группе— связь 1:М.

Отобразим эти связи на рис.7.

Рис.7. Шаг 3 концептуального проектирования: связи между понятиями.

4. Определение является ли понятие атрибутом или сущностью.

Преподаватель — сущность (существует самостоятельно).

Секция — сущность (существует самостоятельно).

Студент — сущность (существует самостоятельно).

Занятие— сущность (участвует в связи M:M, есть атрибут- дата).

Дата занятия— атрибут (не существует самостоятельно, нет атрибутов, не участвует со стороны многие).

Группа— сущность (существует самостоятельно, может быть без студентов).

Название секции — атрибут (не существует самостоятельно, нет атрибутов, не участвует со стороны многие).

Дата вступления — атрибут (не существует самостоятельно, нет атрибутов, не участвует со стороны многие).

5. Для каждого атрибута определить его принадлежность

Дата занятия — атрибут занятия (связь только с занятием). Дата вступления — атрибут связи студент-секция (связан с 2 взаимосвязанными сущностями студент и секция). Название секции — атрибут секции (связь только с секцией).

6. Доопределение атрибутов.

У преподавателя и студента явно должно появиться полное имя (на данном этапе составное из фамилии имени и отчества), у группы номер и год поступления (номера дублируются раз в 10 лет).

7. Определение типов данных

Название секции, имена студента и преподавателя явно текстовые. Конкретизируем их в тип varchar (строки с ограниченной длиной, у которой максимальная длина указывается в скобках), дата занятия и дата вступления студента в секцию —даты, год поступления—целое число. Номер группы может содержать буквы, поэтому он тоже varchar. Для каждой сущности также добавим суррогатные ключи типа INT(целое).

8. Определение обязательности связей и атрибутов.

Ключи обязательны всегда. Имена для студента и преподавателя явно обязательны, без них людей не бывает. Секция без названия тоже не

существует, как и группа без номера и года поступления. Занятие явно не проводится без указания даты. А вступления конкретного студента в секцию можно не знать, то есть он необязателен.

Связь преподаватель-секция необязательна для преподавателя, так как он может не курировать секций, но у секции обязателен руководитель. Связь - преподавателя занятие обязательна только со стороны занятия (его должен кто-то провести). Занятие всегда проходит для какой-то секции, но у секции может пока быть не назначено занятий. Также у секции может пока не быть студентов, а студент не обязан ходить на секции. При этом студент всегда в какой-либо группе, но группа может быть без студентов.

Итоговая ER –диаграмма в нотации птичья лапка с сущностями и атрибутами на английском языке отображена на рис.8.

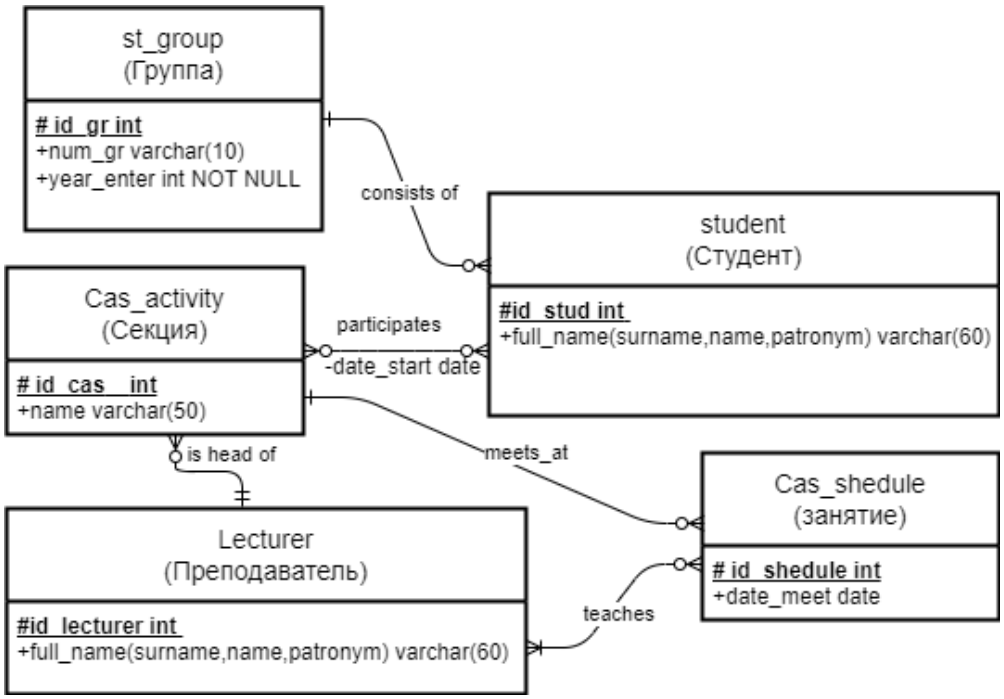


Рис.8 ER –диаграмма предметной области «База данных для управления секциями»

Для понимания имена сущностей продублированы на русском, хотя это не входит в нотацию.

Лабораторная работа №1 Разработка концептуальной модели предметной области

Цель работы: Получение умений и навыков построения концептуальной модели предметной области

Задание и последовательность выполнения работы

Спроектировать концептуальную модель предметной области(ER-диаграмму) в соответствии с вариантом задания. Структура модели должна обеспечивать хранение сведений, необходимых для выполнения запросов, указанных в варианте задания. Все сущности должны быть поименованы и иметь не менее одного атрибута, у которого должен быть проставлен тип данных и задано уникальное в рамках диаграммы имя. Все связи должны быть поименованы, у них должна быть проставлена мощность и обязательность.

Содержание отчета

- Текст задания;
- Концептуальная схема предметной области;
- Выводы о построении концептуальной модели в предметной области.

Контрольные вопросы и задания

- В чем отличие типа сущности и экземпляра сущности?
- Чем отличается сущность (тип сущности) от класса?
- В группе учится много студентов, но только один из них староста. Какой это тип связи?
- Как определить является ли атрибут обязательным? Обязателен ли номер паспорта для человека?
- Какие нотации для отображения ER-диаграммы существуют?
- Чем отличается сильная сущность от слабой? Приведите пример сильной и слабой сущности.

Логическое проектирование баз данных

Логическое проектирование в первую очередь проектирование базы в парадигме выбранной модели данных. На текущий момент можно выделить

следующие модели данных: реляционные, объектные (Объектно-ориентированные), объектно-реляционные, СУБД ключ-значение, документные, семейство столбцов (столбцовые), графовые.
Последние 4 модели относятся к NoSQL СУБД. Кроме определения моделей данных при проектировании базы данных необходимо учитывать её назначение. Выделяют 2 крупных направления назначений баз данных: аналитическое и транзакционное, отличающихся структурой. Первые рассчитаны на сложные статистические запросы для аналитики данных и плохо приспособлены к операциям изменения, вторые предназначены для манипуляции данными за время близкое к реальному. В рамках этого семестра и пособия будут рассматриваться транзакционные базы данных.

Модели данных и выбор применимой модели для различных предметных областей

Несмотря на появление множества различных моделей данных, реляционные и объектно-реляционные базы данных являются наиболее подходящими для работы с хорошо структурированными данными или критичными данными. А нереляционные модели занимают ниши со специфическими задачами.
Основные назначения различных моделями данных указаны в таблице 2.
Таблица 2. Назначения различных моделей данных СУБД

Модели данных	Назначение
Реляционные, объектно-реляционные	Работа с данными с четкой структурой , слабо подверженной изменениям
	Предметные области чувствительные к работе с транзакциями.
Столбцовая	Работа с большими данными, включая данные для машинного обучения, телеметрию интернета вещей и большой трафик.
	Обработка текстов на естественных языках
	Работа с временными рядами (данных с отметками времени, которые представляют собой измерения или события)*
Графовая	Сильно связанные данные, включая анализ спам-рассылок и любые предметные области, представимые в виде графов

	Рекомендательные системы
	Социальные сети
Ключ-значение	Кэширование данных , в том числе хранение информации о сессии, корзины заказа и так далее
	Организация блокировок (mutex)
	Работа с очередями
	Профили пользователей, предпочтения
Документная	Управление контентом (Сервисы блогосферы, которые подразумевают большое количество изображений, аудио и видеоматериалов.)[7]
	Интеграция разрозненных данных, обеспечивающая их единое представление
	Мобильные приложения
	Аналитика в реальном времени

* —Базы данных временных рядов (TSDB, Time Series DataBase) по модели данных чаще всего являются столбцовыми [8].

Построение логической реляционной модели для заданной концептуальной

Для получения логической реляционной модели из концептуальной достаточно применить к концептуальной модели ряд правил.

Если сущность **сильная**, то в логической модели создается отношение, которые включают все простые атрибуты. Если сущность **слабая**, то кроме простых атрибутов в результирующем отношении после преобразования связей с каждой сущностью-владельцем необходимо определить составной первичный ключ. **Многозначный** атрибут преобразуется в дополнительное отношение, представляющего многозначный атрибут, в который передаётся копия первичного ключа сущности-владельца для использования в качестве внешнего ключа. Преобразование связей немного сложнее. Правила преобразования зависят от мощности связи.

Двухсторонняя связь типа 1:M предполагает копирование первичного ключа со стороны «один» на сторону «многие» в качестве внешнего ключа.

На сторону "многие" передаются также все атрибуты связи. Пример такого преобразования приведен на рис. 9.

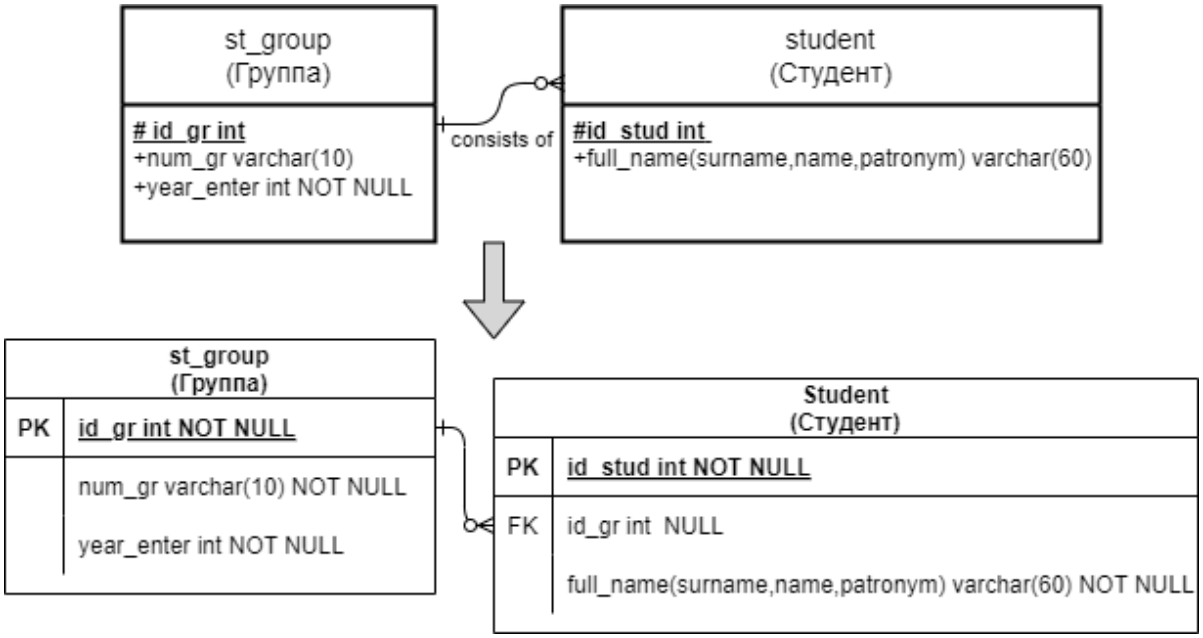


Рис. 9. Преобразование связи 1:M из концептуальной модели в логическую

В примере первичный ключ группы копируется в студента в качестве внешнего.

Двухсторонняя связь типа M:M, сложная связь (более 2 сущностей) раскрывается созданием дополнительной таблицы, представляющего связь, включающего все атрибуты связи и получающего копии первичных ключей связываемых таблиц. Пример преобразования связи M:M приведен на рис. 10.

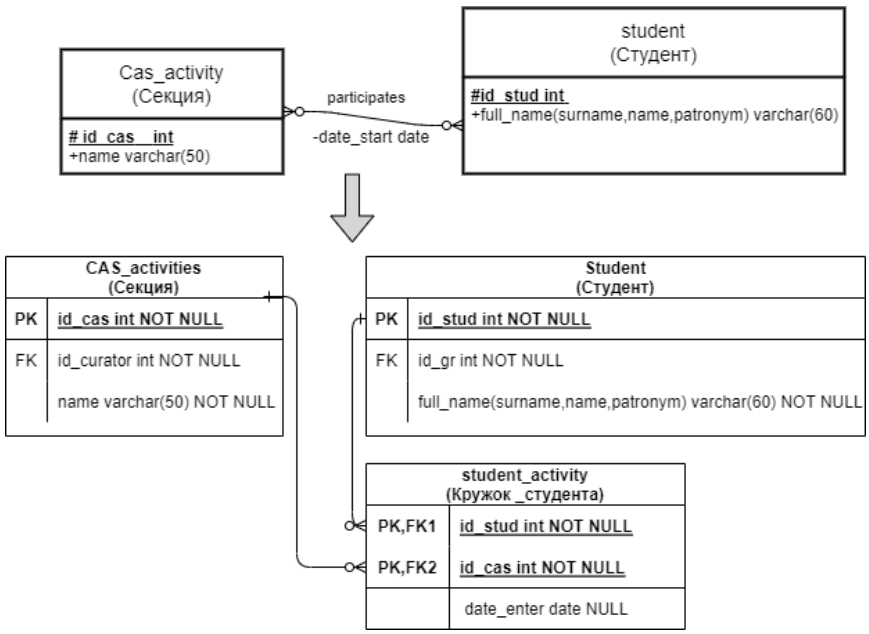


Рис.10. Преобразование связи 1:M из концептуальной модели в логическую

Для **двухсторонней связи типа 1:**1 способ преобразования зависит от обязательности связи. Если связь **обязательна** с обеих сторон, то происходит **объединение**2 сущностей в одно отношение. Пример приведен на рис. 11.

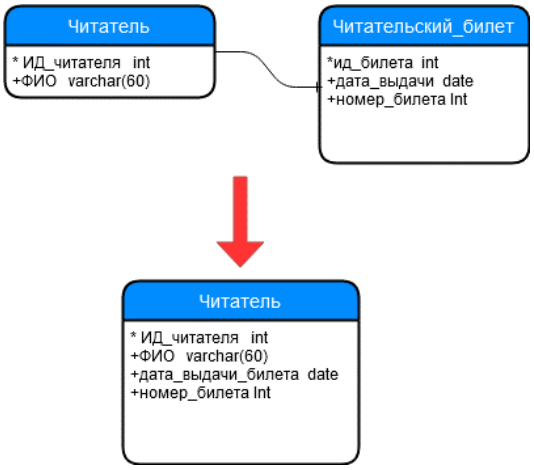


Рис. 11. Преобразование связи 1:1 из концептуальной модели в логическую для обязательных сторон

Для **двухсторонней связи типа 1:**1 если связь обязательна с одной стороны, то на «необязательную сторону» копируется первичный ключ с обязательной (для идентификации). Пример преобразований приведены на рис.12.

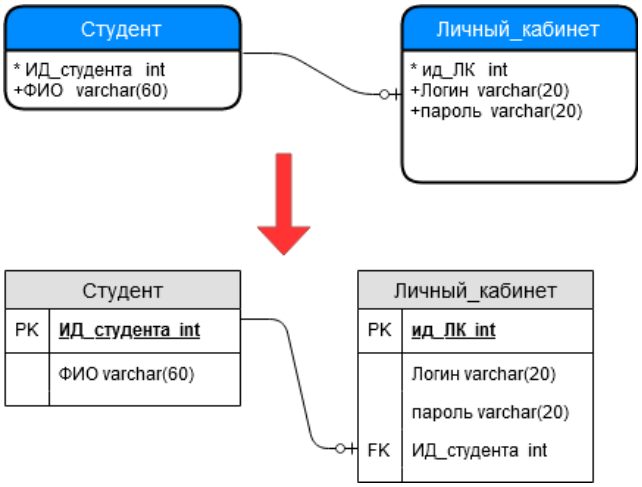


Рис. 12. Преобразование связи 1:1 из концептуальной модели в логическую для обязательной одной стороны

Нормализация баз данных

Нормализация данных – это процесс приведения модели к виду, позволяющему получить в дальнейшем структуру базы данных, в которой

устранена избыточность хранения и сведены к минимуму аномалии при добавлении, удалении, изменении данных.

Процесс нормализации проводится поэтапно. На каждом из этапов на структуру базы накладывается некоторое ограничение. Про базу с соответствующими ограничениями говорят, что она находится в одной из **нормальных форм.**

Выделяют 8 основных нормальных форм:

- Первая нормальная форма (1НФ)
- Вторая нормальная форма (2НФ)
- Третья нормальная форма (3НФ)
- Нормальная форма Бойса-Кодда(НФБК)
- Четвертая нормальная форма (4НФ)
- Пятая нормальная форма (5НФ)
- Шестая нормальная форма (6НФ)
- Доменно-ключевая нормальная форма (ДКНФ)

Каждая из нормальных форм включает требования предыдущей нормальной формы и накладывает свои ограничения/ требования к структуре базы. Каждая нормальная форма с одной стороны упорядочивает схему базы, но в то же время немного усложняет работу с ней (увеличивает количество таблиц-отношений). Поэтому, как правило, при работе ограничиваются использованием первых трех нормальных форм.

Для примера по первым 3 нормальным формам рассмотрим ненормализованную модель с группами и студентами. Она представлена на рис.13.

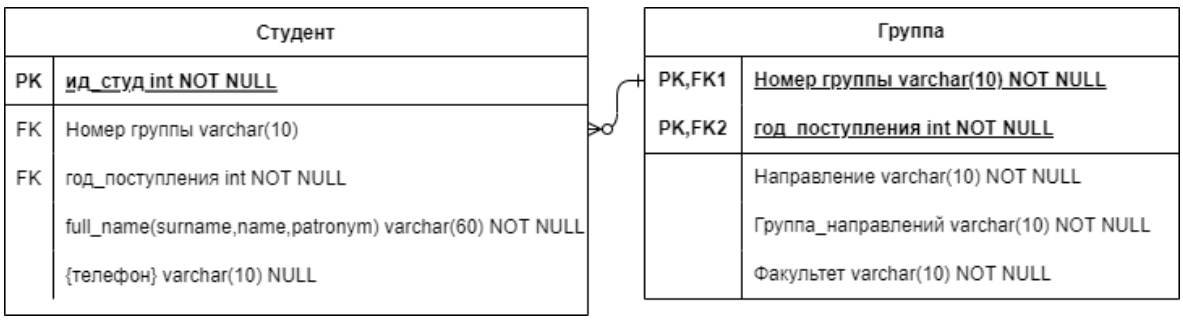


Рис 13. Логическая модель БД до приведения к первой нормальной форме.

В таблице группа указан составной первичный ключ из номера группы и года поступления, факультет, на котором учится группа, направление образовательной программы группы в виде кода (Направление: 09.03.04 «Программная инженерия») и укрупненная группа направлений (УГ) в виде кода (09 «Информатика и вычислительная техника»). В коде направления первые 2 цифры — код УГ.

Таблица (Отношение)находится в**первой нормальная форма (1НФ)** в котором на пересечении каждой строки и каждого столбца содержится одно и только одно атомарное значение.

Нарушают 1 НФ многозначные и составные атрибуты. Для приведения к 1НФ необходимо составные атрибуты раскрыть в простые, а многозначные — выделить в отдельную сущность, связанную связью 1:М. В модели примере составной атрибут— имя студента и многозначный—телефон студента. Результат приведения модели из примера к 1 НФ показан на рис. 14.

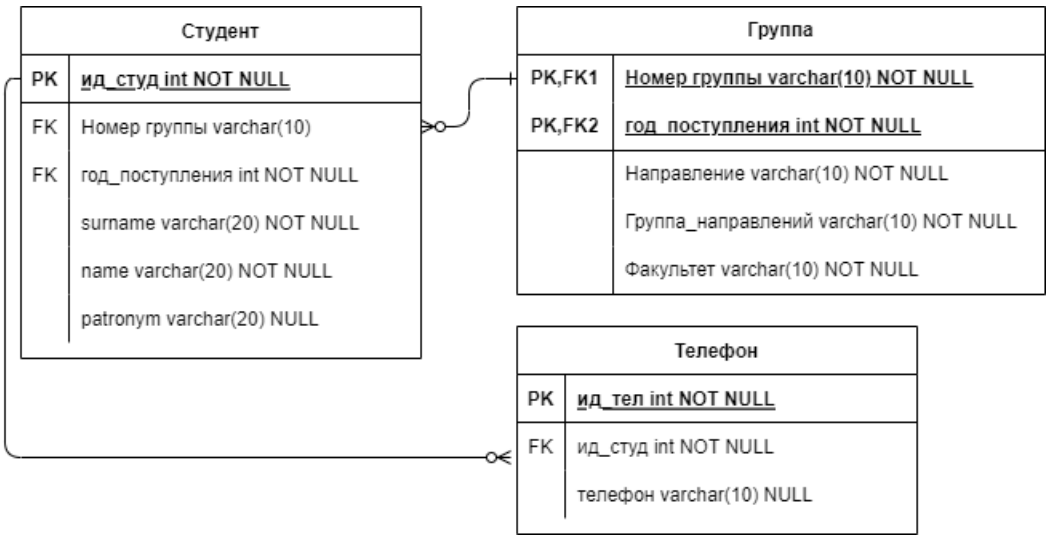


Рис 14. Логическая модель БД в приведения к первой нормальной форме

Вторая и третья нормальные формы требуют работы с понятием функциональной зависимости.

Функциональная зависимость. Поле В таблицы функционально зависит от поля А той же таблицы в том и только в том случае, когда в любой заданный момент времени для каждого из различных значений поля А обязательно существует

только одно из различных значений поля B. Отметим, что здесь допускается, что поля A и B могут быть составными.

Детерминант. Детерминантом функциональной зависимости называется атрибут или группа атрибутов, расположенная на диаграмме функциональной зависимости слева от стрелки. Для наглядности на рис. 15, где изображена зависимость, атрибут A является детерминантом зависимости.

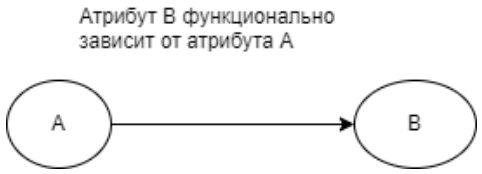


Рис. 15Диаграмма функциональной зависимости

Функциональная зависимость называется **тривиальной**, если она остается справедливой при любых условиях. Зависимость является тривиальной, если и только если в правой части выражения, определяющего зависимость, приведено подмножество (но необязательно собственное подмножество) множества, которое указано в левой части (детерминанте) выражения.

Полная функциональная зависимость. Поле B находится в полной функциональной зависимости от составного поля A, если оно функционально зависит от A и не зависит функционально от любого подмножества поля A.

Таблица(отношение) находится во **второй нормальной форме** (*2НФ*), если она удовлетворяет определению 1НФ и все ее поля, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом. То есть все поля должны зависеть от первичного ключа полностью, а не от его составной части. Для приведения к 2НФ необходимо вынести нарушающие зависимости в отдельные таблицы.

В приведенном примере факультет зависит от номера группы, но не от года поступления, что нарушает 2 НФ. Результат приведения к 2НФ приведен на рис. 16.

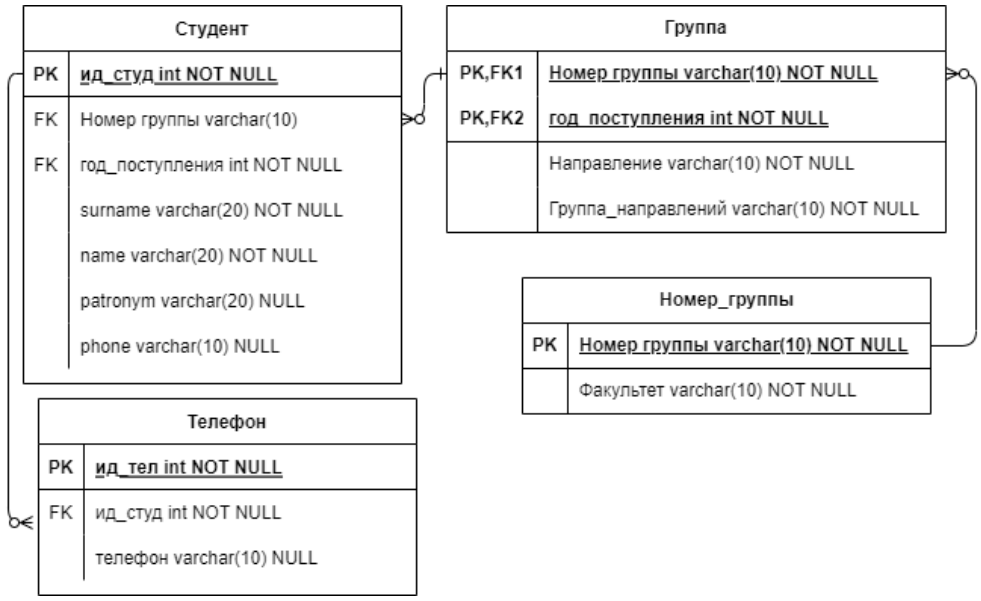


Рис 16. Логическая модель БД в приведения к 2НФ

Таблица находится в ***третьей нормальной форме*** (*3НФ*), если она удовлетворяет определению 2НФ и не одно из ее неключевых полей не зависит функционально от любого другого неключевого поля. Для приведения к 3НФ необходимо вынести нарушающие зависимости в отдельные таблицы. В приведенной модели во 2НФесть код направления однозначно определяет код УГ. Для приведения к 3НФ вынесем эту зависимость в таблицу направление. Результат приведения к 3 НФ отображен на рис. 17.

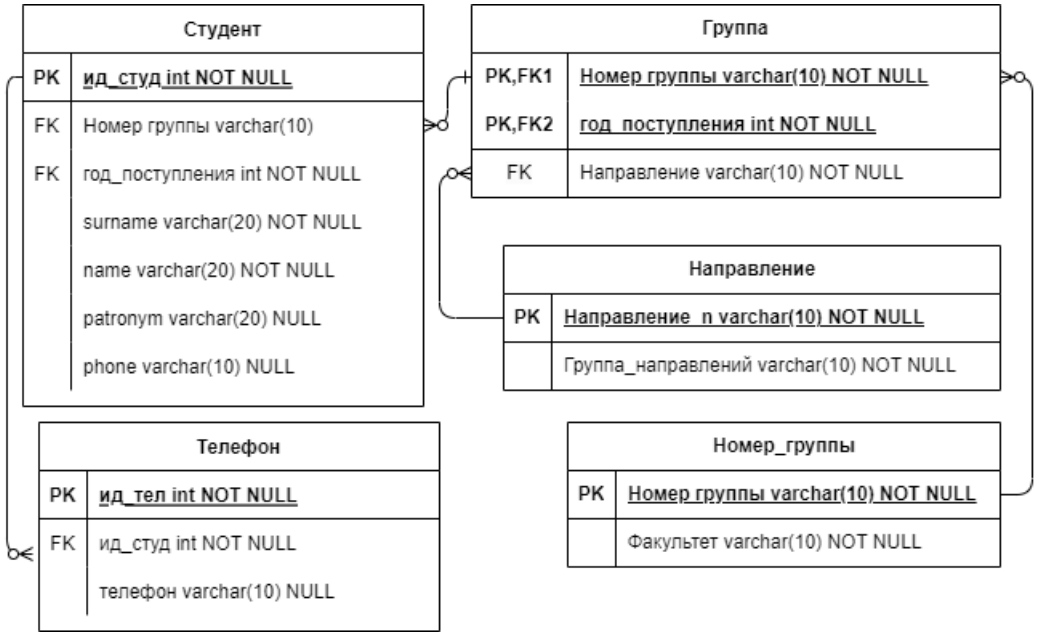


Рис 17. Логическая модель БД в приведения к 2НФ

Есть важный аспект при работе с зависимостями, связанный с тем, что использовано в базе данных в качестве ключей. Это связано с тем, что у суррогатных ключей по сути не может быть зависимостей и при приведении к нормальным формам можно пропустить нарушение 2НФ, перепутав с 3НФ.

Остальные нормальные формы рассмотрим теоретически.

Отношение находится в **Нормальная форма Бойса-Кодда (НФБК).** тогда и только тогда, когда каждый его детерминант является потенциальным ключом.

Различие между ЗНФ и НФБК заключается в том, что функциональная зависимость A → B допускается в отношении ЗНФ, если атрибут B является первичным ключом, а атрибут A не обязательно является потенциальным ключом. Тогда как в отношении НФБК эта зависимость допускается *только* тогда, когда атрибут A является потенциальным ключом. [5]

Многозначная зависимость. Представляет такую зависимость между атрибутами отношения (например, A, B и C), что каждое значение A представляет собой множество значений для B и множество значений для C. Однако множества значений для в и с не зависят друг от друга (A → B, A →C).

Многозначная зависимость может быть дополнительно определена как *тривиальная* или *нетривиальная.* Например, многозначная зависимость A → B некоторого отношения R определяется как тривиальная, если атрибут B является подмножеством атрибута A или A и B = R. И наоборот, многозначная зависимость определяется как нетривиальная, если ни то ни другое условие не выполняется.

Отношение находится в четвертой нормальной форме (4НФ), если оно находится в нормальной форме Бойса-Кодда и не содержит нетривиальных многозначных зависимостей.

Зависимость соединения без потерь это — свойство декомпозиции, которое гарантирует отсутствие фиктивных строк при восстановлении первоначального отношения с помощью операции естественного соединения.

Пятая нормальная форма (5НФ), которая также называется *проективно-соединительной нормальной формой,* или ПСНФ (Project-Join Normal Form -

PJNF), означает, что отношение в такой форме находится в 4 НФ и не имеет зависимостей соединения.

Шестая нормальная форма рассматривается в 2 вариантах: для обычных данных и для темпоральных (временных) данных. Отношение находится **в шестой нормальной форме (6НФ)** тогда и только тогда, когда единственные функциональные зависимости, которые выполняются в нем, являются тривиальными [9]. При этом для обычных данных разделяются элементы отношения, а при временных- время актуальности атрибутов.

Ограничение домена – это ограничение, предписывающее использование для определенного атрибута значений только из некоторого заданного домена (набора значений). **Ограничение ключа** – это ограничение, утверждающее, что некоторый атрибут или комбинация атрибутов представляет собой потенциальный ключ. Отношение Rнаходится в **доменно-ключевой нормальной форме**, тогда и только тогда, когда каждое ограничение, которое выполняется в R, подразумевается ограничениями домена и ключевыми ограничениями, которые выполняются в отношении

Пример построения логической реляционной модели и нормализации данных

По приведенным правилам для концептуальной модели «База данных для управления секциями», приведенной на рисунке 8 получается логическая модель, приведенная на рис. 18.

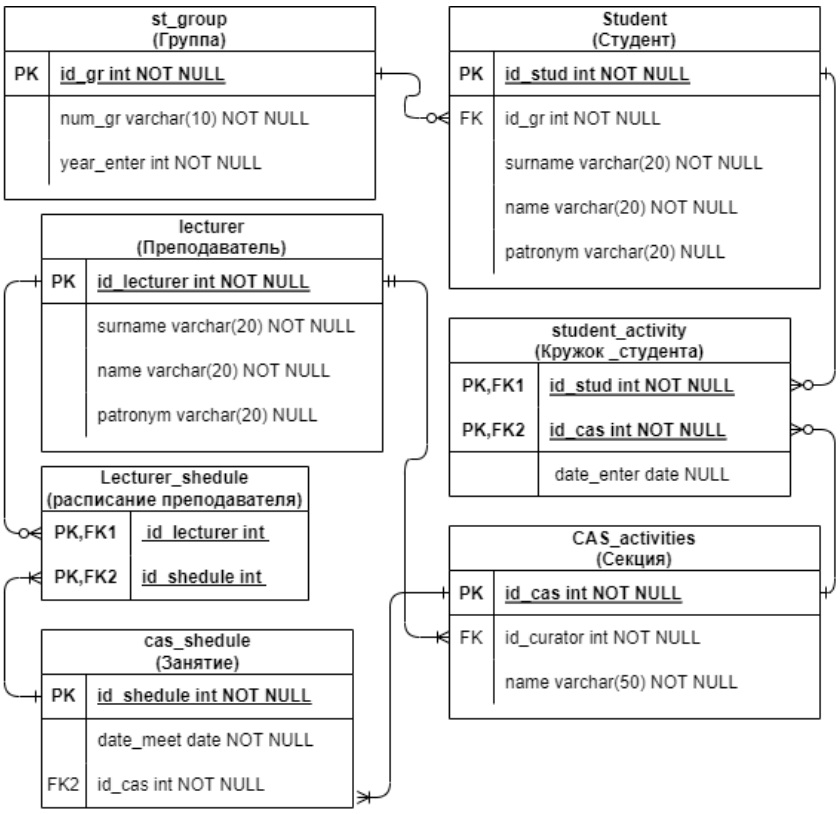


Рис. 18 Логическая модель «База данных для управления секциями»

Из необходимых действий по нормализации здесь только раскрытие составных атрибутов в именах

Контрольные вопросы и задания

- В каких случаях при переходе от концептуальной к логической модели добавляются новые таблицы?
- Почему при разработке БД ограничиваются приведением к 3 НФ?
- Какие последствия использования ненормализованной базы?
- Каким образом в реляционной базе может быть нарушена 1 НФ?
- Физическое проектирование баз данных

Физическая модель конкретизирует логическую, уточняя типы данных и методы хранения для конкретной СУБД. В том числе описывает ссылочную целостность и может описывать особенности индексации.

Ссылочная целостность

Пустое значение (Null) указывает, что значение атрибута в настоящий момент неизвестно или неприемлемо для этого кортежа.

Существует 2 вида целостности: на уровне сущности и на уровне ссылки. **Целостность на уровне сущности** требует, чтобы все элементы первичного ключа были уникальны, и никакая часть первичного ключа не была пустой (null). Это гарантирует, что каждая сущность (логический объект) будет иметь уникальную идентификацию, а значения внешнего ключа могут должным образом ссылаться на значения первичного ключа. Например, номер зачетной книжки не может иметь несколько дублирующихся значений и не может иметь пустое значение. То есть, все студенты уникально идентифицируются своей зачетной книжкой.

Целостность на уровне ссылки требует, чтобы внешний ключ имел или пустое значение (если только он не является частью первичного ключа данной таблицы), или значение, совпадающее со значением первичного ключа в связанной таблице. (Каждое непустое значение внешнего ключа должно ссылаться на существующее значение первичного ключа.)**.** Выполнение правила целостности на уровне ссылки делает невозможным удаление строки в одной таблице, где первичный ключ имеет обязательное соответствие со значением внешнего ключа в другой таблице или изменение такого первичного ключа. Например, студенту может быть не назначена (еще) Группа, но невозможно назначить студенту несуществующую группу.

Ссылочная целостность может поддерживаться 2 способами: декларативно и активно. При декларативной ссылочной целостности способы поддержания целостности задаются в виде ограничений и явно объявляются при определении таблиц. Она наименее гибкая, но не требует дополнительного кода. При активной (процедурной) ссылочной целостности, все действия по её поддержанию должны быть реализованы в триггерах- специальных хранимых процедурах- обработчиках событий. Для декларативной ссылочной целостности доступны несколько вариантов поддержания ссылочной целостности, приведенные в таблице 3.

Таблица 3. Действия в декларативной ссылочной целостности

Наименование типа поддержания	Код на SQL	Что происходит при удалении	Что происходит при обновлении
Ограничение	restrict	Не дает удалить данные из родительской таблицы, если с ними связаны какие-либо данные в дочерней (проверка сразу)	Не дает изменить первичный ключ из родительской таблицы, если с ним связаны какие-либо данные в дочерней (проверка сразу)
Ограничение	No action	Не дает удалить данные из родительской таблицы, если с ними связаны какие-либо данные в дочерней (проверка отложена)	Не дает изменить первичный ключ из родительской таблицы, если с ним связаны какие-либо данные в дочерней (проверка отложена)
Каскадирование	cascade	При удалении данных из родительской таблицы, удалятся связанные с ними данные в дочерней	При изменении первичного ключа из родительской таблицы, связанные с ним внешние ключи в дочерней изменятся на такое же значение
Установка	set null	При удалении данных из родительской таблицы, внешние ключи дочерней таблицы связанные с удаляемыми данными получат пустое значение(null)	При изменении первичного ключа из родительской таблицы, внешние ключи дочерней таблицы связанные с изменяемыми ключами получат пустое значение(null)
Установка	set default	При удалении данных из родительской таблицы, внешние ключи дочерней таблицы связанные с удаляемыми данными получат значение по умолчанию, которое должно быть задано в дочерней таблице	При изменении первичного ключа из родительской таблицы, внешние ключи дочерней таблицы связанные с изменяемыми ключами получат значение по умолчанию, которое должно быть задано в дочерней таблице

Особенности СУБД и индексация данных

Одной из частей проектирования физической модели является проектирование индексации. Существует множество типов индексов, поэтому определяем индекс по способу его использования, а не ориентируясь структурные свойства. Структура данных называется индексом, если:

это избыточная структура данных, она невидима для приложения, она предназначена для ускорения выбора данных по определенным критериям. [10].

Индекс уникален, если каждому индексированному значению соответствует ровно одна строка в таблице. В частности, и PostgreSQL, и MySQL автоматически создают уникальный индекс для поддержки каждого первичного

ключа или ограничения уникальности в таблице. Создание других индексов сопряжено с анализом часто используемых медленных запросов.

Лабораторная работа №2 Разработка физической модели базы данных с учетом декларативной ссылочной целостности

Цель работы: Получение навыков построения логической и физической моделей данных.

Задание и последовательность выполнения работы

1. Создать физическую модель базы данных, находящуюся в третьей нормальной форме в соответствии с заданным вариантом.
2. Описать ссылочную целостность БД в таблице и добавить её обоснование.

Таблица должна иметь вид, представленный для таблицы 4 или иметь формат, описанный ниже.

Таблица 4. Пример описания ссылочной целостности БД

Дочерняя таблица	Внешний ключ	Родительская таблица	ссылочная целостность при удалении	Описание ссылочной целостности при удалении	ссылочная целостность при обновлении	Описание ссылочной целостности при обновлении
Table1	Id_t2	Table2	Каскадируется	При удалении данных из Table2, удалятся все связанные данные из Table1	Каскадируется	При обновлении первичного ключа Table2, обновится внешний ключ из Table1

Описание может быть не в таблице, но должно содержать те же данные.

3. Описать возможные уникальные индексы в СУБД

Содержание отчета

- Цель работы
- Текст задания (вместе с текстом варианта задания);

– концептуальная модель БД;
– физическую модель БД;
– таблица с описанием ссылочной целостности;
– описание возможных уникальных индексов;
– Выводы о физическом проектировании для данной предметной области.

Контрольные вопросы и задания

– Какие действия предпринимаются для поддержания ссылочной целостности?
– В чем разница между активной и декларативной ссылочной целостностью?
– Что такое физическое проектирование базы данных?
– Почему для таблицы, раскрывающей связь многие-ко-многим нельзя ставить в декларативной ссылочной целостности установку пустого значения?

Введение в SQL. Язык определения данных

Состав языка SQL

В состав языка SQL *(Structured Query Language)*, входят Data Definition Language (DDL) — язык определения данных, Data Manipulation Language (DML) — язык манипулирования данными, Transaction Control Language (TCL)— язык управления транзакциями, Data Control Language (DCL)— язык управления доступом к данным.

Язык определения данных

DDL (Data Definition Language) - это группа операторов языка *SQL*, используемых для определения структуры базы данных и ее объектов, таких как таблицы, представления, индексы и процедуры. Наиболее распространенные DDL операторы приведены в таблице 5:

Таблица 5 – Операторы *DDL*

Create	Создает новый объект базы данных, такой как таблица, представление или индекс.
Alter	Используется для изменения существующего объекта базы данных.

Drop	Используется для удаления существующего объекта базы данных.
Truncate	Используется для удаления всех строк в таблице, но в отличие от оператора Drop он сохраняет структуру таблицы и индексы.
Rename	Используется для переименования существующего объекта базы данных.

Важно отметить, что *DDL* операторы выполняются немедленно и являются постоянными, то есть после создания, изменения или удаления объекта изменения невозможно отменить.

Создание таблицы (оператор create)

Для создания таблицы используется оператор *CREATE TABLE*. Синтаксис *CREATE TABLE* с основными параметрами [11,12]:

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ] TABLE [ IF NOT
EXISTS ] имя_таблицы ( [
{ имя_столбцатип_данных [ COMPRESSION метод_сжатия ] [ COLLATE
правило_сортировки ] [ ограничение_столбца [ ... ] ]
    | ограничение_таблицы
    | LIKE исходная_таблица [ вариант_копирования ... ] }
    [, ... ]
] )
[ INHERITS ( таблица_родитель [, ... ] ) ]
[ PARTITION BY { RANGE | LIST | HASH } ( { имя_столбца | ( выражение ) } [
COLLATE правило_сортировки ] [ класс_операторов ] [, ... ] ) ]
[ USINGметод ]
[ WITH ( параметр_хранения [= значение] [, ... ] ) | WITHOUT OIDS ]
[ ON COMMIT { PRESERVE ROWS | DELETE ROWS | DROP } ]
[ TABLESPACE табл_пространство ];
```

После оператора *CREATE TABLE* следует записать название таблицы и после него открыть круглые скобки. В скобках надо перечислить через запятую поля, которые будет содержать таблица. Для удобства чтения каждое поле должно находиться в отдельной строке. Имена могут содержать символы подчеркивания для большей наглядности. Максимальная длина названия и для таблицы, и для столбцов — 64 символа.

Каждому полю нужно прописать тип данных и ограничения. Ниже в таблице 6приведены наиболее часто используемые типы полей:

Таблица 6 – Наиболее часто используемые типы данных

Тип	Что содержит столбец	Пример
VARCHAR/CHAR/TEXT[1]	Хранит строку	'Маша', 'перышко', 'летучий корабль'
INT	Хранит целое число	1; 2; 356

DECIMAL	Хранит число с фиксированной запятой	1.5 или 356.12
FLOAT/DOUBLE	Хранит число с плавающей запятой	1.5 или 356.12
DATE	Хранит дату (день, месяц, год) в формате ГГГГ-ММ-ДД	'1999-03-03', '2017-07-25', '2023-10-27'
TIMESTAMP	Хранит дату и время(день, месяц, год, часы, минуты и секунды) в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС	'1999-03-03 01:36:12', '2017-07-25 17:16:15', '2023-10-27 11:57:34'

Примечание 1:

VARCHAR (N)- строка ограниченной длины, в неё можно поместить не более N символов.

CHAR(N) - строка фиксированной длины, в неё можно поместить ровно N символов. Если строка будет короче, то недостающие символы будут заменены пробелами

TEXT по стандарту относится к типу CHARACTER LARGE OBJECT (до 2 Гб теста), но хранимого не в самой строке и при этом ограничены (а в некоторых СУБД не возможны) поиск подстроки и создание индексов.

Для разных СУБД названия типов полей могут отличаться. Например, вместо *INT* может употребляться *INTEGER*. Тип *DATE* в зависимости от СУБД может хранить разную информацию — например, только день и месяц или день, месяц и время. Важно уточнять в документации к конкретной СУБД используемые типы данных.

Пример: Создадим таблицу для хранения данных о группах, где *id_gr* — уникальный номер, num_gr — номер группы, course — номер курса.

```
CREATE TABLE st_group(
 id_gr int,
 num_gr varchar(8),
coursesmallint);
```

Если повторно попытаться создать таблицу с помощью *CREATE TABLE*, а таблица уже есть в базе данных, команда выдаст ошибку. Поэтому перед созданием разумно проверить, содержит ли уже база данных таблицу с таким именем. Достаточно добавить в *CREATE TABLE IFNOTEXISTS*.

Пример:

```
GREATE TABLE IN NOT EXISTS student
```

Атрибуты (*ATTRIBUTES*) и ограничения (*CONSTRAINTS*)

PRIMARY KEY

Для обозначения первичного ключа используется специальная конструкция *PRIMARY KEY()*. Эта конструкция объявляет поле уникальным ключом. Например, номер паспорта для человека может быть неплохим первичным ключом, а вот имя человека — плохой ключ, ведь людей с одинаковыми именами очень много. Аналогично связка **имя и фамилия** тоже плохой первичный ключ. Поле, которое объявлено ключом, обязательно должно быть уникальным и не содержать пустых значений. То есть в терминах *SQL* первичный ключ — это поле со свойствами *UNIQUE* и *NOT NULL*.[11,12]

Пример:

```
CREATE TABLE st_group(
 id_gr int,
 num_gr varchar(8),
 course smallint,
 PRIMARY KEY (id_gr));
```

NOTNULL

Для запрета создания пустых полей используется оператор *NOTNULL*. Если нет явного указания *NOT NULL*, и этот столбец не *PRIMARY KEY*, то столбец позволяет хранить *NULL*, то есть хранение *NULL*— поведение по умолчанию. Для первичного ключа это ограничение можно не указывать, так как первичный ключ всегда гарантирует *NOT NULL*.[11,22]

Пример:

```
CREATE TABLE st_group(
 id_gr int NOT NULL,
 num_gr varchar(8),
 course smallint,
PRIMARY KEY (id_gr));
```

DEFAULT

С помощью данной команды можно указать значение по умолчанию, т.е. текст или число, которые будут сохранены, если не указано другое значение. Применяется не ко всем типам: *BLOB, TEXT, GEOMETRY и JSON* не

поддерживают это ограничение. Значение по умолчанию должно быть константой, функция или выражение недопустимы.

Для типа данных *BOOLEAN* можно использовать встроенные константы *FALSE* и *TRUE*. Вместо *DEFAULT(FALSE)*можно указать *DEFAULT(0)* - эти записи эквивалентны.

Пример:

```
CREATE TABLE st_group(
 id_gr int NOT NULL,
 num_gr varchar(8) DEFAULT NULL,
 course smallint DEFAULT 1
PRIMARY KEY (id_gr));
```

AUTO_INCREMENT

Каждый раз, когда в таблицу будет добавлена запись, значение этого столбца автоматически увеличится. На всю таблицу этот атрибут не применим, только к одному столбцу, причем этот столбец должен быть ключом. Рекомендуется использовать для целочисленных значений. Нельзя сочетатьс *DEFAULT*. [11]

Пример:

```
CREATE TABLE st_group(
 id_gr int NOT NULLAUTO_INCREMENT,
 num_gr varchar(8) DEFAULT NULL,
 course smallint DEFAULT 1
PRIMARY KEY (id_gr));
```

Данный атрибут применим в *MySQL*, но отсутствует в *PostgreSQL. В PostgreSQL* для этого используются следующие типы данных: *smallserial* , *Serial* и *bigserial*. Это числовые типы данных значение которых образуется путем автоинкремента значения предыдущей строки.[12]

Пример:

```
CREATE TABLE st_group(
 id_gr serial NOT NULL,
 num_gr varchar(8),
 course smallint,
PRIMARY KEY (id_gr));
```

UNIQUE

Для создания уникальных значений поля используется команда *UNIQUE*. Эта команда используется вместе с командой *NOT NULL*, потому что в ином случае будет допускать только одно значение *NULL*. [11,12]

Пример:

```
CREATE TABLE student (
 id_st int NOT NULL,
 surname varchar(20) DEFAULT NULL,
 name varchar(15) DEFAULT NULL,
 patronym varchar(25) DEFAULT NULL,
 id_gr int DEFAULT NULL,
 rate int UNIQUE NOT NULL,
 PRIMARYKEY (id_st),
);
```

CHECK

Команда *CHECK* позволяет добавлять условие, которому должна соответствовать строка. Позволяет установить дополнительную проверку данных для столбца или набора столбцов. Если значение проверяемого атрибута равно *NULL*, то строка не будет добавлена в таблицу, так как в результате работы условия будет получен неопределенный результат.

Пример:

```
Birthday DATE NOT NULL CHECK (birthday >’1900-01-01’)
```

Ниже приведен пример использования команды *CHECK* и определено ограничение даты рождения и допустимые форматы телефона через регулярное выражение.

```
CREATE TABLE student (
 id_st int NOT NULL,
 surname varchar(20) DEFAULT NULL,
 name varchar(15) DEFAULT NULL,
 patronym varchar(25) DEFAULT NULL,
 id_gr int DEFAULT NULL,
 rate int UNIQUE NOT NULL,
birthday DATE NOT NULL,
 PRIMARY KEY (id_st),
CONSTRAINT student_chk_birthday CHECK (birthday > '1900-01-01'));
```

Для добавления ограничений используется оператор *CONSTRAINT*, при этом, все названия уникальны, как и имена таблиц. Учитывая, что по умолчанию названия включают в себя и имя таблицы, рекомендуется придерживаться этого правила.

FOREIGN KEY(внешний ключ)

Внешний ключ является ссылкой на столбец или группу столбцов другой таблицы. Это тоже ограничение (*CONSTRAINT*), так как можно использовать только значения, для которых есть соответствие по внешнему ключу. Создает индекс. Таблицу с внешним ключом называют зависимой.

Синтаксис:

```
FOREIGNKEYимя_столбца1, имя_столбца2)
REFERENCES внешнее_имя_таблицы(внешнее_имя_чтолбца1,внешнее_имя_столбца2)
```

Сначала указывается выражение *FOREIGN KEY* и набор столбцов таблицы. Затем ключевое слово *REFERENCES* указывает на имя внешней таблицы и набор столбцов этой внешней таблицы. В конце можно добавить операторы *ON DELETE* и *ON UPDATE*, с помощью которых настраивается поведение при удалении или обновлении данных в родительской таблице. Это делать необязательно, так как предусмотрено поведение по умолчанию. Поведение по умолчанию запрещает удалять или изменять записи из внешней таблицы, если на эти записи есть ссылки по внешнему ключу.

Возможны 3 опции для *ON DELETE* и *ON UPDATE*:*CASCADE,.SET NULL,RESTRICT:*, описанные в разделе физического моделирования.[11,12]

Пример:

```
CREATE TABLE student (
 id_st int NOT NULL,
 surname varchar(20) DEFAULT NULL,
 name varchar(15) DEFAULT NULL,
 patronym varchar(25) DEFAULT NULL,
 id_gr int DEFAULT NULL,
 rate int DEFAULT NULL,
 PRIMARY KEY (id_st),
foreign key (id_gr) references st_group(id_gr) on delete cascade on update
restrict);
```

При *CREATE TABLE*, чтобы не усложнять описание столбца, рекомендуется указывать внешний ключ и все его атрибуты после перечисления создаваемых столбцов. Можно добавить внешний ключ, если таблица уже создана и в ней есть данные. Для внесения изменений в таблицу используем *ALTER TABLE*.

ALTER TABLE

SQL оператор *ALTER TABLE* используется для добавления, изменения, удаления столбцов в таблице или для переименования таблицы.

Синтаксис оператора *ALTER TABLE* для *PostgreSQL*выглядит следующим образом: [12]

```
ALTER TABLE название_таблицы [WITHCHECK|WITH NOCHECK]
{ ADD название_столбца тип_данных_столбца [атрибуты_столбца] |
DROP COLUMN название_столбца |
ALTER COLUMN название_столбца тип_данных_столбца [NULL|NOTNULL] |
ADD [CONSTRAINT] определение_ограничения |
DROP [CONSTRAINT] имя ограничения}
```

Синтаксис оператора *ALTER TABLE* для *MySQL*выглядит следующим образом: [11]

```
ALTER TABLE название_таблицы
{ ADD название_столбца тип_данных_столбца [атрибуты_столбца] |
  DROP COLUMN название_столбца |
  MODIFY COLUMN название_столбца тип_данных_столбца [атрибуты_столбца] |
  ALTER COLUMN название_столбца SETDEFAULTзначение_по_умолчанию |
  ADD[CONSTRAINT] определение_ограничения |
  DROP[CONSTRAINT] имя_ограничения}
```

Добавление столбца (ов) в таблицу

Синтаксис *SQL* оператора *ALTER TABLE* для добавления столбца в таблицу.

```
ALTER TABLE название_таблицы
ADD название_столбца тип_данных_столбца [атрибуты_столбца];
```

Примеры:

Добавление одного столбца в таблицу. *ALTER TABLE* добавит столбец с именем*name*в таблицу*student.*

```
ALTER TABLE student
ADD name varchar(15);
```

Аналогично происходит добавление нескольких столбцов.

```
ALTER TABLE student
ADD(name varchar(15),
    City varchar(45));
```

Изменить столбец(ы) в таблице

Синтаксис *ALTER TABLE* для изменения столбца в существующей таблице для *MySQL и PostgreSQL*различаются.

Примеры:

Изменение одного столбца.

Для*MySQL.*

```
ALTER TABLE student
 MODIFY name varchar(15) NOT NULL
```

Для*PostgreSQL*.

```
ALTER TABLE student
ALTER COLUMN name varchar(15),
ALTER COLUMN name SET NOT NULL;
```

Изменение нескольких столбцов происходит аналогично изменению одного столбца.

Для *MySQL*.

```
ALTER TABLE student
 MODIFY name varchar(15) NOT NULL,
 MODIFY city varchar(55);
```

Для *PostgreSQL*.

```
ALTER TABLE student
ALTER COLUMN name varchar(15),
ALTER COLUMN name SET NOT NULL,
ALTER COLUMN city varchar(55);
```

Удаление столбца(ов) в таблице

Синтаксис *ALTER TABLE* для удаления столбца в существующей таблице.

```
ALTER TABLE имя_таблицы
DROP COLUMN имя_столбца;
```

Пример:

```
ALTER TABLE student DROP COLUMN name;
```

Переименовать столбец в таблице

Синтаксис *ALTER TABLE* для переименования столбца в существующей таблице для *PostgreSQL и MySQL* различаются.

Для *PostgreSQL*.

```
ALTER TABLE имя_таблицы
RENAME COLUMN старое_имя_столбцаTOновое_имя_столбца
```

Для *MySQL*.

```
ALTER TABLE имя_таблицы
CHANGECOLUMNстарое_имя_столбцаTOновое_имя_столбца;
```

Пример

Для*PostgreSQL*.

```
ALTER TABLE student
RENAME COLUMN name TO s_name;
```

Для*MySQL*.

```
ALTER TABLE student
 CHANGE COLUMN name TO s_name;
```

Переименовать таблицу

Синтаксис *ALTER TABLE* для переименования таблицы.

```
ALTER TABLE имя_таблицы
RENAME TO новое_имя_таблицы
```

Пример:

```
ALTER TABLE student
RENAME TO s_student;
```

DROP TABLE

SQL оператор *DROP TABLE* позволяет удалить таблицу из базы данных.

Синтаксис для оператора *DROP TABLE* в *SQL*.[11,12]

```
DROP TABLE имя_таблицы;
```

Пример:

```
DROP TABLE student;
```

TRUNCATE

Данный оператор служит для быстрой очистки таблицы – удаляет все строки из нее.

TRUNCATE TABLE – удаляет все строки в таблице, не записывая в журнал удаление отдельных строк. Оператор *TRUNCATE TABLE* похож на оператор *DELETE* без предложения *WHERE*, однако *TRUNCATE TABLE* выполняется быстрее и требует меньших ресурсов системы и журналов транзакций.

TRUNCATE TABLE нельзя использовать если на таблицу ссылается ограничение *FOREIGN KEY*. Таблицу, имеющую внешний ключ, ссылающийся сам на себя, можно усечь. [11,12]

Пример:

```
TRUNCATE TABLE student
```

TRUNCATE TABLE обладает следующими преимуществами по сравнению с оператором *DELETE:*

- Используется меньший объем журнала транзакций.

Удаляет данные, освобождая страницы данных, используемые для хранения данных таблиц, и в журнал транзакций записывает только данные об освобождении страниц.

- Обычно используется меньшее количество блокировок.

Всегда блокирует таблицу (включая блокировку схемы (*SCH-M*)) и страницу, но не каждую строку.

- В таблице остается нулевое количество страниц, без исключений.

Оператор *TRUNCATE TABLE* удаляет все строки таблицы, но структура таблицы и ее столбцы, ограничения, индексы и т.п. сохраняются. Чтобы удалить не только данные таблицы, но и ее определение, следует использовать оператор *DROP TABLE*.

Если таблица содержит столбец идентификаторов, счетчик этого столбца сбрасывается до начального значения, определенного для этого столбца. Если начальное значение не задано, используется значение по умолчанию, равное 1. Чтобы сохранить столбец идентификаторов, используйте оператор *DELETE*.

Для *TRUNCATE TABLE* можно выполнить откат.

Нельзя использовать *TRUNCATE TABLE* в таблицах в следующих случаях:

- На таблицу ссылается ограничение *FOREIGN KEY*. Таблицу, имеющую внешний ключ, ссылающийся сам на себя, можно усечь.
- Таблица является частью индексированного представления.
- Таблица опубликована с использованием репликации транзакций или репликации слиянием.
- Это темпоральная таблица с управлением версиями.
- На таблицу ссылается ограничение *EDGE*.

Для таблиц с какими-либо из этих характеристик следует использовать инструкцию *DELETE*.

Rename

RENAME TABLE переименовывает одну или несколько таблиц.

Синтаксис:

```
RENAME TABLE старое_название_таблицы TO новое_название_таблицы;
```

Этот оператор эквивалентен следующему *ALTER TABLE* оператору:

```
ALTER TABLE старое_название_таблицы RENAME новое_название_таблицы;
```

RENAME TABLE в отличие от *ALTER TABLE*, можете переименовать несколько таблиц в одном запросе:

```
RENAME TABLE старое_название_таблицы TO новое_название_таблицы,
старое_название_таблицы2 TO новое_название_таблицы2,
старое_название_таблицы3 TO новое_название_таблицы3;
```

Лабораторная работа №3 Создание и модификация базы данных и таблиц базы данных

Цель работы: Получение умений и навыков создания и модификации таблиц на языке SQL.

Задание и последовательность выполнения работы

В соответствии с моделью, разработанной в предыдущей работе, создать базу данных. Продемонстрировать умение добавить и удалить столбец командой alter table.

Содержание отчета

- Цель работы
- Текст задания (вместе с текстом варианта задания);
- физическую модель БД;
- таблица с описанием ссылочной целостности;
- Выводы об особенностях создания таблиц разработанной модели данных в выбранной СУБД.

Контрольные вопросы и задания

- Что такое DDL и какие операторы относятся к DDL?
- Что можно изменить с помощью команды alter table?
- В чем разница между строкой фиксированной длины и ограниченной длины?
- Какие ограничения можно задать с помощью команды создания таблиц?
- Как задать ограничения с помощью команды создания таблиц?
- Можно ли создать внешний ключ, ссылающийся не на первичный ключ?
- Зачем нужно имя ограничения при задании ограничений, внешних ключей в частности?

Введение в SQL. Язык манипулирования данными

Основные команды по манипулированию данными и примеры их применения

DML (Data Manipulation Language) - это операторы языка *SQL* (язык структурированных запросов), которые используются для манипулирования данными в базе данных. *DML* операторы используются для вставки, обновления и удаления данных в базе данных. Некоторые из наиболее распространенных *DML*операторы представлены в таблице 7:

Таблица 7– Операторы *DML*

Select	Используется для получения данных из одной или нескольких таблиц базы данных.
Insert	Используется для вставки новых данных в таблицу.
Update	Используется для изменения существующих данных в таблице.
Delete	Используется для удаления данных из таблицы.
Merge	Используется для выполнения условных операций *INSERT, UPDATE* или *DELETE* строк в таблице.

DML операторы выполняются немедленно и могут быть отменены с помощью оператора отката. Важно отметить, что, хотя *DML* операторы используются для создания, изменения и удаления объектов базы данных, *DML* операторы используются для манипулирования данными внутри этих объектов.

Синтаксис оператора INSERT

Оператор *INSERT* используется для вставки новых данных в таблицу.

Данный оператор имеет 2 основные формы:[1,2]

```
INSERT INTO таблица(перечень_полей) VALUES(перечень_значений)
```

Вставка в таблицу новой строки значения полей которой формируются из перечисленных значений.

```
INSERT INTO таблица(перечень_полей) SELECT перечень_значений
FROM имя_таблицы
```

Вставка в таблицу новых строк, значения которых формируются из значений строк, возвращенных запросом.

Несколько заметок про *INSERT*:

- Порядок перечисления полей не имеет значения, важно только то, чтобы он совпадал с порядком значений, которые вы перечисляете в скобках после ключевого слова *VALUES*.
- Так же важно, чтобы при вставке были заданы значения для всех обязательных полей, которые помечены в таблице как *NOT NULL*.
- Можно не указывать поля у которых была указана опция *IDENTITY* или же поля у которых было задано значение по умолчанию при помощи *DEFAULT*, т.к. в качестве их значения подставится либо значение из счетчика, либо значение, указанное по умолчанию.
- В случаях, когда значение поля со счетчиком нужно задать явно используется опция *IDENTITY_INSERT*.

Синтаксис оператора UPDATE

SQL оператор *UPDATE* используется для обновления существующих записей в таблицах.

Синтаксис для оператора *UPDATE* при обновлении таблицы в *SQL*. Поля в [] не обязательны. [11,12]

```
UPDATE table
SET имя_столбца1 = значение,
имя_столбца2 = значение2,
...
[WHERE условие];
```

Синтаксис оператора *UPDATE* при обновлении таблицы данными из другой таблицы:

```
UPDATE имя_таблицы1
SET имя_столбца1 = (SELECT значение1
FROM имя_таблицы2
WHERE условие)
[WHERE условие];
```

Синтаксис оператора *UPDATE* при обновлении нескольких таблиц:

```
UPDATE имя_таблицы1, имя_таблицы2,
SET имя_столбца1 = значение1,
имя_столбца2 = значение2,
...
WHERE имя_таблицы1.имя_столбца = имя_таблицы2.имя_столбца
[AND условие];
```

Синтаксис оператора DELETE

SQL оператор *DELETE* используется для удаления одной или нескольких записей из таблицы. Синтаксис оператора *DELETE*:[11,12]

```
DELETEFROM имя_таблицы
```

```
[WHERE условие];
```

Если запустить оператор *DELETE* без условий в предложении *WHERE*, все записи из таблицы будут удалены.

Пример:

```
DELETE FROM student WHERE name= 'Александр';
```

Можно задать несколько условий в инструкции *DELETE* для этого используя либо условие *AND*, либо условие *OR*. Условие *AND* позволяет удалить запись, если все условия выполнены. Условие *OR* удаляет запись, если выполняется одно из условий.

```
DELETE FROM student WHERE id>50 AND name='Александр';
```

Можно удалить записи в одной таблице на основе значений в другой таблице. Поскольку нельзя перечислить более одной таблицы в предложении *FROM* при выполнении удаления, можно использовать предложение EXISTS.

```
DELETE FROM student
WHERE EXISTS (SELECT * FROM st_group
  WHERE student.id_gr= st_group.id_gr AND student.name= 'Александр');
```

Слияние данных (оператор MERGE)

MERGE – оператор, который можно использовать для выполнения условных операций *INSERT, UPDATE* или *DELETE* строк в таблице.

Синтаксис оператора *MERGE* для *PostgreSQL* выглядит следующим образом: [12]

```
[ WITHзапрос_WITH [, ...] ]
MERGEINTO [ ONLY ] имя_целевой_таблицы [ * ] [ [ AS ] целевой_псевдоним ]
USING источник_данных ON условие_соединения
предложение_when [...]
      t-
здесь источник_данных:
{ [ ONLY ] имя_исходной_таблицы [ * ] | ( исходный_запрос ) } [ [ AS ]
исходный_псевдоним ]
и предложение_when:
{ WHEN MATCHED [ AND условие ] THEN { изменение_при_объединении |
удаление_при_объединении | DO NOTHING } |
WHEN NOT MATCHED [ AND условие ] THEN { добавление_при_объединении | DO NOTHING
} }
и добавление_при_объединении:
INSERT [( имя_столбца [, ...] )]
[ OVERRIDING { SYSTEM | USER } VALUE ]
{ VALUES ( { выражение | DEFAULT } [, ...] ) | DEFAULT VALUES }
и изменение_при_объединении:
UPDATE SET { имя_столбца = { выражение | DEFAULT } |
             ( имя_столбца [, ...] ) = ( { выражение | DEFAULT } [, ...] ) } [,
...]
и удаление_при_объединении:
DELETE
```

Принцип работы: вставить/обновить или удалить строки в таблице *имя_целевой_таблицы* из *источник_данных* по условию.

Конструкция *MERGE* чем-то напоминает условный оператор *CASE*, она так же содержит блоки *WHEN*, при выполнении условий которых происходит то или иное действие. Модификация данных производится в таблице приемнике.

MERGE может иметь не больше двух предложений *WHEN MATCHED*. Если указаны два предложения, то первое предложение должно сопровождаться дополнительным условием. Для любой строки второе предложение *WHEN MATCHED* применяется только в том случае, если не применяется первое.

Если имеются два предложения *WHEN MATCHED*, одно должно указывать действие *UPDATE*, а другое - *DELETE*. Предложение *WHEN NOT MATCHED* используется для вставки строк из источника, не совпадающих со строками в изменяемой таблице согласно условию связи. Инструкция MERGE может иметь только одно предложение WHEN NOT MATCHED.

Пример: Добавить записи в таблицу *customers* с отсутствующими электронными адресами, обновить имя, если дата создания клиента меньше, чем соответствующая дата в *leads*, и удалить другие записи:

```
MERGE INTO customers AS c
USING leads AS l
ON c.email = l.email
WHEN NOT MATCHED THEN
  INSERT (name, email, created_at)
  VALUES (l.email, l.name, DEFAULT)
WHEN MATCHED AND c.created_at < l.created_at THEN UPDATE
    SET name = l.name
WHEN MATCHED THEN
DELETE;
```

В *MySQL*оператор *MERGE* отсутствует. Есть таблица *MERGE* (или таблица *MRG_MyISAM*) представляющая собой совокупность идентичных таблиц *MyISAM*, которые могут использоваться как одна таблица. [11]

Лабораторная работа №4 Заполнение таблиц и модификация данных

Цель работы: Получение умений и навыков манипулирования данными в реляционной базе данных

Задание и последовательность выполнения работы

1)Выполнить вставку тестовых данных в таблицы, созданные в ходе выполнения лабораторной работы 2.

В строках, вставляемых в таблицы, должны быть данные как удовлетворяющие, так и не удовлетворяющие условиям запросов, приведенных в варианте задания. (Для демонстрации этого необходимо в отчете создать таблицу , где будет указано задание на запрос, данные удовлетворяющие условиям запроса, данные не удовлетворяющие условиям запроса)

2)Необходимо привести свои примеры использования операторов update и delete и merge с описанием их назначения.

Содержание отчета

- Цель работы
- текст задания (вместе с текстом варианта задания);
- физическую модель БД;
- наборы данных, содержащихся в таблицах БД;
- таблица тестовых данных аналогичная таблице 8:

Таблица 8. Тестовые данные

Текст запроса	данные удовлетворяющие условиям запросов	данные не удовлетворяющие условиям запросов
a. станции в названии которых есть слово «площадь»	Таблица1 (станция) Площадь Ленина	Таблица1 (станция) Московские ворота

- примеры использования insert, update, delete и merge;
- скрипт полного заполнения базы;
- Выводы об особенностях манипулирования данными в выбранной СУБД.

Контрольные вопросы и задания

- Какие способы записи оператора вставки данных вы знаете?
- Что такое DMLи какие операторы относятся к DML?

– Что делает оператор merge?
– Когда ставятся кавычки у констант при вставке данных?
– Приведите пример синтаксиса команды обновления данных.
– Приведите пример синтаксиса команды удаления данных.

Язык SQL. Оператор выборки

Синтаксис оператора SELECT

Назначение оператора SELECT состоит в выборке и отображении данных одной или более таблиц базы данных. Оператор SELECT является чаще всего используемой командой языка SQL. Общий формат оператора SELECT имеет следующий вид:

```
SELECT [DISTINCT | ALL] .{ *| [columnExpression [AS newName]] [ ,  ... ]}
FROM TableName [alias][ ,..  ]
[ WHERE условиевыбора строк]
[GROUP BY условие группировки]
[HAVING условие выбора групп]
[ORDERBY условие сортировки]
```

Раздел *SELECT*. Устанавливается, какие столбцы должны присутствовать в выходных данных. Слово distinct позволяет убрать дублирующиеся строки из результата запроса.

Здесь параметр *columnExpression* может включать только следующие типы элементов:

• имена столбцов;

• агрегирующие функции;

• константы;

• выражения, включающие комбинации перечисленных выше элементов.

В противоположность списку столбцов * обозначает возврат всех столбцов всех таблиц запроса.

Параметр *TableName* является именем существующей в базе данных таблицы (или представления), к которой необходимо получить доступ. Необязательный параметр alias — это сокращение, псевдоним, устанавливаемое для имени таблицы *TableName.* Обработка элементов оператора SELECT выполняется в следующей последовательности.

• *FROM*. Определяются имена используемой таблицы или нескольких таблиц, перечисленные через запятую. Также в качестве источника данных в этом разделе могут быть таблицы, связанные различными видами соединения, представления и другие запросы.

• *WHERE*. Выполняется фильтрация строк объекта в соответствии с заданными условиями.

• *GROUPBY*. Образуются группы строк, имеющих одно и то же значение в указанном столбце.

• *HAVING*. Фильтруются группы строк объекта в соответствии с указанным условием, относящимся к результату агрегатной функции.

• *ORDERBY*. Определяется упорядоченность результатов выполнения оператора.

При этом каждому полю в списке сортировки может быть приписано *ASC* (по возрастанию- это параметр сортировки поля по умолчанию) или *DESC* (по убыванию)

Порядок конструкций в операторе *SELECT не может* быть изменен. Только две конструкции оператора — *SELECT и FROM* — являются обязательными, все остальные конструкции могут быть опущены.

Условия в конструкции *WHERE*

(применимы как к оператору *SELECT*, так и *UPDATE* или *INSERT*)

В приведенных выше примерах в результате выполнения операторов *SELECT* выбирались все строки указанной таблицы. Однако очень часто требуется тем или иным образом ограничить набор строк, помещаемых в результирующую таблицу запроса. Это достигается с помощью указания в запросе конструкции *WHERE*. Она состоит из ключевого слова *WHERE*, за которым следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов условий поиска

• Сравнение. Сравниваются результаты вычисления одного выражения с результатами вычисления другого выражения. (<,>,=,<>)

• Диапазон. Проверяется, попадает ли результат вычисления выражения в заданный диапазон значений. (имя_поля *BETWEEN* знач_1 *AND* знач_2)
• Принадлежность к множеству. Проверяется, принадлежит ли результат вычисления выражения к заданному множеству значений. (имя_поля in (знач_1,знач_2…, знач_n))
• Значение *NULL*. Проверяется, содержит ли данный столбец *NULL*(неопределенное значение) (имя_поля *ISNULL*).
• Соответствие шаблону. Проверяется, отвечает ли некоторое строковое значение заданному шаблону. (имя_поля *LIKE* шаблон)
• %. Символ процента представляет любую последовательность из нуля или более символов (поэтому часто именуется также *подстановочным символом).*
• _. Символ подчеркивания представляет любой отдельный символ. .
Все остальные символы в шаблоне представляют сами себя.
Пример
`st_group.num_grLIKE '4%'`. Этот шаблон означает, что первый символ значения обязательно должен быть символом 4, а все остальные символы не представляют интереса и не проверяются.
`st_group.num_gr LIKE '4_ _ _ _'`.Этот шаблон означает, что значение должно иметь длину, равную строго четырем символам, причем первым символом обязательно должен быть символ ' 4 ' . (пробелы между подчеркиваниями только для визуализации –их нет в шаблоне)
`patronym LIKE ' %ич'`. Этот шаблон определяет любую последовательностьсимволов длиной не менее двух символов, причем последними символами обязательно должен быть символы ич. (поиск мужчин по отчеству)
• `name LIKE` '%слово%'. Этот шаблон означает, что нас интересует любая последовательность символов, включающая подстроку «слово»;
Если требуемая строка должна включать также служебный символ, обычно применяемый в качестве символа подстановки, то следует определить с помощью конструкции ESCAPE "маскирующий" символ, который указывает,

что следующий за ним символ больше не имеет специального значения, и поместить его перед символом подстановки. Например, для проверки значений на соответствие литеральной строке 45%' можно воспользоваться таким предикатом: *LIKE '15#%' ESCAPE '#'*

Пример: Выберемвсехстудентовгруппы4432К по схеме, изображенной на рисунке 18.

```
select id_student, student.id_gr, surname, name, patronym from student, st_group
where student.id_gr =st_group.id_gr and number_gr='4432К'
```

Виды соединений в языке SQL

Виды соединений в SQL:

- Декартово произведение *CROSS JOIN*
- Внутреннее соединение *INNERJOIN* (часто просто JOIN)
- Внешние соединения:
- Левое соединение *LEFT JOIN*
- Правое соединение *RIGHTJOIN*
- Полное (внешнее) соединение*OUTER JOIN*

Соединение является подмножеством более общей комбинации данных двух таблиц, называемой *декартовым произведением.* Декартово произведение двух таблиц представляет собой другую таблицу, состоящую из всех возможных пар строк, входящих в состав обеих таблиц. Набор столбцов результирующей таблицы представляет собой все столбцы первой таблицы, за которыми следуют все столбцы второй таблицы. Если ввести запрос к двум таблицам без задания конструкции WHERE, результат выполнения запроса в среде SQL будет представлять собой декартово произведение этих таблиц.

Внутреннее соединение используется для связи таблиц по совпадающим значениям столбцов. Внешние соединения нужны в случае, когда значениям в одной таблице не всегда будут иметь соответствующие значения в другой (т.е. внешний ключ – пуст (NULL))

Свойства соединений:

Внутреннее соединение симметрично, т.е. от перемены местами таблиц в соединении ничего не изменится

```
SELECT tableName1.*, tableName2.*,  FROM tableName1INNER JOIN tableName2
```

Эквивалентно

```
SELECT tableName1.*, tableName2.*,  FROM tableName2 INNER JOIN tableName1
```

Полное внешнее соединение также симметрично. Левое и правое соединения симметричны друг по отношению к другу, т.е.

```
SELECT tableName1.*, tableName2.*,  FROM tableName1LEFT JOIN tableName2
```

Эквивалентно

```
SELECT tableName1.*, tableName2.*,  FROM tableName2RIGHT JOIN tableName1
```

Для примеров различных соединений используем таблицы группа и студент, изображенные на рис. 19

student

id_student	id_gr	surname	name	patronym
1	1	Петров	Петр	Петрович
2	2	Иванов	Иван	Иванович
3	*NULL*	Пупкин	Василий	Федорович

st_group

id_gr	number_gr
1	Z4431K
2	Z5432K
3	B5433

Рис.19. Данные для примеров

```
Select number_gr, id_student, surname,name, patronym from st_group inner join student on student.id_gr=st_group.id_gr
```

Результат такого запроса будет таким как изображено на рис.20.

st_group INNER JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович

Рис.20. Результат внутреннего соединения

```
Select number_gr, id_student, surname,name, patronym from st_group left join student on student.id_gr=st_group.id_gr
```

Результат такого запроса изображен на рис. 21.

st_group LEFT JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович
B5433	*NULL*	*NULL*	*NULL*	*NULL*

Рис.21. Результат левого соединения

```
Select number_gr, id_student, surname,name, patronym from st_group right join student on student.id_gr=st_group.id_gr
```

Результат такого запроса с правым соединением будет представлен на рис.22.

st_group RIGHT JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович
NULL	3	Пупкин	Василий	Федорович

Рис.22. Результат правого соединения

```
Select number_gr, id_student, surname,name, patronym from st_group outer join
student on student.id_gr=st_group.id_gr
```

Результат запроса с полным внешним соединением изображен на рис. 23.

st_group OUTER JOIN student

number_gr	id_student	surname	name	patronym
Z4431K	1	Петров	Петр	Петрович
Z5432K	2	Иванов	Иван	Иванович
NULL	3	Пупкин	Василий	Федорович
B5433	*NULL*	*NULL*	*NULL*	*NULL*

Рис.23. Результат полного внешнего соединения

Левое соеднение можно использовать для нахождения групп без студентов по схеме на рис. 18

```
Select number_gr, id_student, surname,name, patronym FROM st_group LEFT
JOIN student ON student.id_gr=st_group.id_gr

WHERE student.id_student is NULL;

или

Select number_gr, id_student, surname,name, patronym FROM student RIGHT
JOIN st_group ON student.id_gr=st_group.id_gr

WHERE student.id_student is NULL;
```

Агрегатные функции в операторе выборки языка SQL

Стандарт ISO содержит определение следующих пяти *агрегирующих функций:*

• *COUNT* — возвращает количество значений в указанном столбце;

• *SUM* — возвращает сумму значений в указанном столбце;

• *AVG* — возвращает среднее арифметическое значение в указанном столбце;

• *MIN* — возвращает минимальное значение в указанном столбце;

• *MAX* — возвращает максимальное значение в указанном столбце.

Все эти функции оперируют со значениями в единственном столбце таблицы и возвращают единственное значение. Функции *COUNT, MIN и MAX* применимы

как к числовым, так и к нечисловым полям, тогда как функции *SUM и AVG* могут использоваться только в случае числовых полей. За исключением *COUNT (*)* , при вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся непустым значениям столбца. Вариант *COUNT (*)* является особым случаем использования функции *COUNT* — его назначение состоит в подсчете всех строк в таблице, независимо от того, содержатся там пустые, повторяющиеся или любые другие значения.

Если до применения агрегирующей функции необходимо исключить повторяющиеся значения, следует перед именем столбца в определении функции поместить ключевое слово *DISTINCT*. Если список выборки *SELECT* содержит агрегирующую функцию, а в тексте запроса отсутствует конструкция GROUPBY, обеспечивающая объединение данных в группы, то ни один из элементов списка выборки SELECT не может включать каких-либо ссылок на столбцы, за исключением случая, когда этот столбец используется как параметр агрегирующей функции.

Группирование результатов (конструкция GROUPBY)

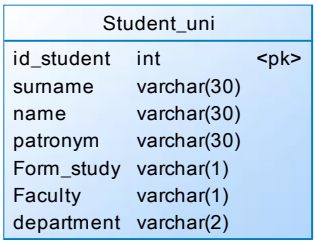
Все имена столбцов, приведенные в списке выборки *SELECT*, должны присутствовать и в конструкции *GROUPBY*, за исключением случаев, когда имя столбца используется только в агрегирующей функции. Если совместно с конструкцией *GROUPBY* используется конструкция *WHERE*, то она обрабатывается в первую очередь, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

конструкция HAVING

Конструкция *HAVING* предназначена для использования совместно с конструкцией *GROUPBY*.В основном *HAVING* действует так же, как выражение *WHERE* в операторе *SELECT*. Тем не менее, выражение *WHERE* применяется к столбцам и выражениям для отдельной строки, в то время как оператор *HAVING* применяется к результату действия команды *GROUPBY* (агрегатной функции).

Приведем пример использования агрегатных функций и группировки.

Допустим существует таблица Student_*Uni*, изображенная на рис. 24.



,

Рис.24. Отношение студент университета

Где From_study- форма обучения («О», «В» «З»), Faculty- факультет, department- кафедра.

Если мы хотим посчитать всех студентов вуза, необходимо написать запрос

```
SELECT count (id_student) as all_st from Student_Uni
```

, тогда в результате мы получим одно число равное общему количеству студентов.

Если мы хотим узнать сколько учится студентов на каждой форме обучения надо написать

```
SELECT count (id_student) as all_st, Form_study from Student_Uni GROUP BY Form_study
```

На самом деле сгруппировав по форме обучения, мы сделали равенство форм обучения критерием отнесения студента к той или иной группе(ячейке), в которой потом будет производиться подсчет. Т.е. разбили все множество студентов по форме обучения

```
SELECT count (id_student) as all_st, Faculty from Student_Uni GROUP BY Faculty
```

А если мы сгруппируем и по факультету, и по форме обучения

```
SELECT count (id_student) as all_st, Faculty, Form_study from Student_Uni GROUP BY Faculty, Form_study
```

То получим ячейку(группу) у которой одинаковы оба эти значения

Каждый раз добавляя в группировку поля мы уменьшаем ячейку, поэтому, например, запрос типа

```
SELECT count (id_student) as all_stfrom Student_Uni GROUP BYid_student
```

Выдаст и много строк с единицами (размер ячейки подсчета студентов- один студент) **Поэтому обратите внимание если у Вас группировка и агрегатная функция заданы для одного поля, то это, вероятно, ошибка**

Примеры реализации различных запросов
Запросы с использованием псевдонимов

Необходимость в таких запросах возникает, когда одна и та же таблица должна использоваться в разных качествах (кафедра, ведущая дисциплину у группы и кафедра, по которой группа выпускается), или когда нужно чтобы некоторый атрибут принимал 2 значения одновременно (группа, которая изучает и Дифференциальные уравнения и Объектно-ориентированное программирование)

Общий вид псевдонима на таблицу выглядит`select … from таблица as` **`псевдоним.`** AS иногда можно опустить.

Пример по предметной области учебного плана, отображенного на рис. 25.

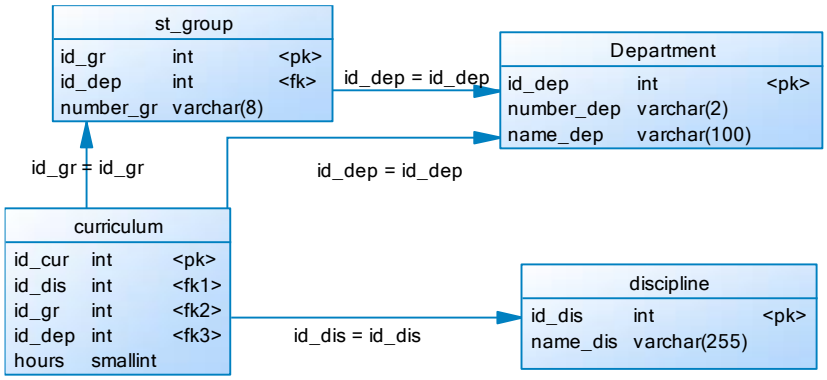


Рис. 25. Схема БД учебного плана

Написать запрос: Группы, у которых ведет дисциплины и 43 и 41 кафедра.

Для наглядности отобразим схему самого запроса на рис 26.

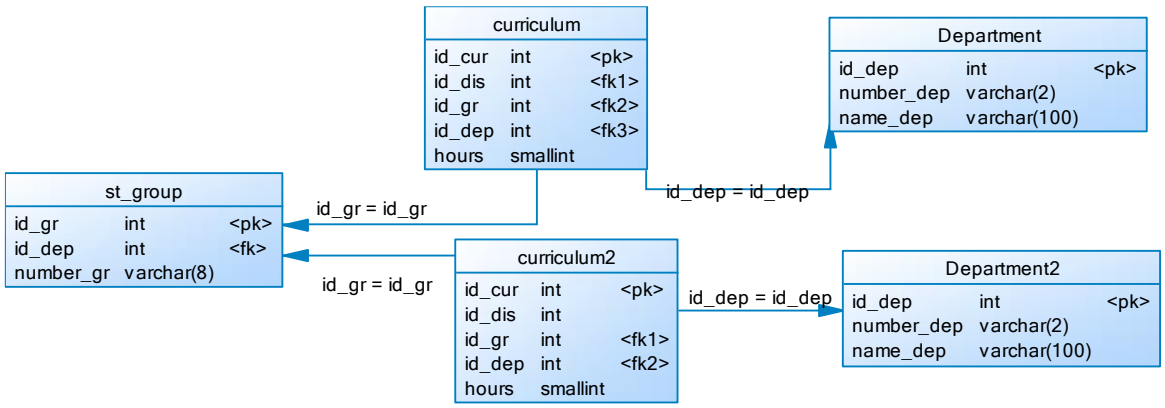


Рис.26. Схемазапросаспсевдонимами

```
Select distinct st_group.id_gr, number_gr from st_group
inner join curriculum on curriculum.id_gr= st_group.id_gr
inner join Department on Department.id_dep=curriculum.id_dep
inner join curriculum as curriculum2 on curriculum2.id_gr= st_group.id_gr
inner join Department as Department2 on Department2.id_dep=curriculum2.id_dep
where Department.number_dep='43' and Department2.number_dep='41'
```

При необходимости не 2 конкретных кафедр, а 2 различных последяя строка будет выглядеть

```
where Department.number_dep< Department2.number_dep
```

Лабораторная работа №5 Разработка SQL запросов: виды соединений и шаблоны

Цель работы: Получение навыков создания запросов без подзапросов, используя различные виды соединений.

Задание и последовательность выполнения работы

1)Реализовать запросы а) .. в), указанные в варианте задания. .

- Запрос на поиск по шаблону (поиск подстроки)(выполнить с единственным оператором like)

- Запрос на использование одной таблицы несколько раз (псевдонимы).

- Запрос на использование внешних соединений.

Все запросы НЕ должны содержать вложенных запросов или агрегатных функций (используйте псевдонимы и различные виды соединений вместо них).

Содержание отчета

- Цель работы
- текст задания (вместе с текстом варианта задания);
- физическую модель БД;

– элементы таблицы тестовых данных из предыдущей работы, касающиеся запросов а-в:
– текст запросов на SQL;
– наборы данных, возвращаемые запросами (скриншоты);
– выводы об особенностях реализации простых запросов на выборку.

Контрольные вопросы и задания

– В чем отличие левого соединения от правого?
– В чем отличие левого соединения от внутреннего? Когда каждое из них применяется?
– Какие подстановочные символы используются при поиске по шаблону?
– Что такое _ (подчеркивание) при поиске по шаблону?
– Что такое эскейп символ при поиске по шаблону?
– Что такое псевдонимы и для чего они используются в запросах?

Язык SQL. Запросы с подзапросами

Принципы построения запросов с подзапросами

Здесь мы обсудим использование законченных операторов *SELECT*, внедренных в тело другого оператора *SELECT*. *Внешний* (второй) оператор *SELECT* использует результат выполнения *внутреннего* (первого) оператора для определения содержания окончательного результата всей операции. Внутренние запросы могут находиться в конструкциях *WHERE* и *HAVING* внешнего оператора *SELECT* — в этом случае они получают название *подзапросов,* или *вложенных запросов.* Кроме того, внутренние операторы *SELECT* могут использоваться в операторах *INSERT*, *UPDATE* и *DELETE* .

Также возможно нахождение подзапроса в конструкции FROM , где он должен получить псевдоним после скобок и с ним можно работать как с любой таблицей.

Существуют три типа подзапросов.

• *Скалярный подзапрос* возвращает значение, выбираемое из пересечения одного столбца с одной строкой, т.е. единственное значение. В принципе

скалярный подзапрос может использоваться везде, где требуется указать единственное значение.

• *Строковый подзапрос* возвращает значения нескольких столбцов таблицы, но в виде единственной строки. Строковый подзапрос может использоваться везде, где применяется конструктор строковых значений, — обычно это предикаты.

• *Табличный подзапрос* возвращает значения одного или нескольких столбцов таблицы, размещенные в более чем одной строке. Табличный подзапрос может использоваться везде, где допускается указывать таблицу, например как операнд предиката IN.

Пример скалярного подзапроса

Вывести группу, в которой учится столько же человек, как и в группе Z4431

```
Select st_group.* from st_group left join student on
student.id_gr=st_group.id_gr
Group by student.id_gr
Having count (id_student)=
(select count(*) from st_group gr
   left join student st on st.id_gr=gr.id_gr where  number_gr=' Z4431')
```

Left join, т.к. в группе может не быть студентов

Табличный подзапрос

Вывести всех студентов из групп ,где учится Иванов

```
Select student.* from student where id_gr in

(select st_group.id_gr from from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname='Иванов')
```

К подзапросам применяются следующие правила и ограничения.

1. В подзапросах не должна использоваться конструкция *ORDERBY*, хотя она может присутствовать во внешнем операторе *SELECT*.

2. Список выборки *SELECT* подзапроса должен состоять из имен отдельных столбцов или составленных из них выражений, за исключением случая, когда в подзапросе используется ключевое слово *EXISTS*.

3. По умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в конструкции *FROM* подзапроса. Однако разрешается ссылаться и на столбцы таблицы, указанной в конструкции *FROM* внешнего

запроса, для чего используются уточненные имена столбцов (как описано ниже).

4. Если подзапрос является одним из двух операндов, участвующих в операции сравнения, то подзапрос должен указываться в правой части этой операции. Например, приведенный ниже вариант записи запроса является некорректным, поскольку подзапрос размещен в левой части операции сравнения со значением столбца *salary*.

```
SELECT staffNo, fName, IName, position, salary
FROM Staff
WHERE (SELECT AVG(salary) FROM Staff) < salary;
```

Объединение, пересечение, разность запросов в языке SQL

• *Объединением* двух таблиц A и B называется таблица, содержащая все строки, которые имеются в первой таблице (A), во второй таблице (B) или в обеих этих таблицах одновременно,

• *Пересечением,* двух таблиц называется таблица, содержащая все строки, присутствующие в обеих исходных таблицах одновременно.

• *Разностью* двух таблиц A и B называется таблица, содержащая все строки, которые присутствуют в таблице A, но отсутствуют в таблице B.

Запросы должны иметь одинаковое количество столбцов, причем в соответствующих столбцах должны размещаться данные одного и того же типа и длины. Три операции над множествами, предусмотренные стандартом ISO, носят название *UNION*, *INTERSECT* и *EXCEPT* и позволяют комбинировать результаты выполнения двух и более запросов в единую результирующую таблицу. В каждом случае формат конструкции с операцией над множествами должен быть следующим:

Запрос1 operator .[ALL] Запрос2

Одни диалекты языка SQL не поддерживают операций *INTERSECT* и *EXCEPT* (MySQL), а в других вместо ключевого слова EXCEPT используется ключевое слово *MINUS*. Для примера выберем имена и фамилии преподавателей или студентов с непустым отчеством. (логическое или)

```
(SELECT surname, name FROM student
WHERE patronym IS NOT NULL)
UNION
```

```
(SELECT surname, name FROM teacher
WHERE patronym IS NOT NULL);
```

При использовании ALL после UNION в результате сохранятся дублирующие значения.

Выберем фамилии и имена и отчества студентов , которые не преподают (в магистратуре) (студенты минус преподаватели)(считаем, что полных тезок нет)

```
(SELECT surname, name, patronym FROM student )
EXCEPT
(SELECT surname, name, patronym FROM teacher);
```

Ключевые слова *ANY* и *ALL*

Ключевые слова *ANY* и *ALL* могут использоваться с подзапросами, возвращающими один столбец чисел. Если подзапросу будет предшествовать ключевое слово *ALL*, условие сравнения считается выполненным только в том случае, если оно выполняется для всех значений в результирующем столбце подзапроса. Если тексту подзапроса предшествует ключевое слово *ANY*, то условие сравнения будет считаться выполненным, если оно удовлетворяется хотя бы для какого-либо (одного или нескольких) значения в результирующем столбце подзапроса. Пример Найдите всех работников, чья зарплата превышает зарплату хотя бы одного работника отделения компании под номером 'вооз '.

```
SELECT staffNo, fName, IName, position, salary
FROM Staff WHERE salary >ANY(SELECT salaryFROM Staff
WHERE branchNo = 'B003');
```

Экзистенциальные запросы в языке SQL

Ключевые слова *EXISTS* и *NOT* EXISTS предназначены для использования только совместно с подзапросами или управляющими конструкциями. Результат их обработки представляет собой логическое значение *TRUE* или *FALSE*. Для ключевого слова *EXISTS* результат равен *TRUE* в том и только в том случае, если в возвращаемой подзапросом результирующей таблице присутствует хотя бы одна строка. Если результирующая таблица подзапроса пуста, результатом обработки ключевого слова *EXISTS* будет значение *FALSE*. Для ключевого слова NOT*EXISTS* используются правила обработки, обратные по отношению к ключевому слову *EXISTS*. Например, группа, где есть студенты, будет получена запросом

```
Select st_group.* from st_group where
```

```
exists (select id_student from student where student.id_gr=st_group.id_gr)
```

Часто условие NOTEXISTS используется для реализации разности.

Например: Группа без студентов может быть найдена запросом

```
Select st_group.* from st_group where
not exists (select id_student from student where student.id_gr=st_group.id_gr)
```

Обратите внимание, что в подзапросе сквозная видимость

Одно из назначений NOTEXISTS – реализация реляционного деления (так называемые запросы на «все»).

Рассмотрим на примере схемы базы данных участия студентов в кружках (секциях), представленном на рисунке 18.

Запрос звучит так: **Студенты, которые посещают все кружки.**

Для самой простой по смыслу реализации такого запроса посмотрим на него с точки зрения реляционного деления. В данном случаемы должны поделить студентов на кружки, то есть студент- это делимое, а все кружки - делитель.

такой запрос можно разделить на 3 части- запроса:

```
ANOT EXISTS
(B NOT EXISTS (C))
```

При этом каждая часть имеет свое назначение.

Часть А— это делимое, В— делитель, а С отвечают за связку между делимым и делителем (студент и все секции)

Запросы А и С как правило похожи с точностью до псевдонимов (сквозная видимость- псевдонимы обязательны), однако в С идет дополнительная связка с А по ключу делимого (в данном случае студента) и связка С и В по ключу делителя (в данном случае секции).Запрос В задает делитель.

Такой запрос будет выглядеть:

```
Select student.* from student
where not exists
(select * from CAS_activities
where not exists
(Select * from student student_activity as st_a
where student.id_st=st_a.id_stand st_a.id_cas= CAS_activities.id_cas))
```

Если запрос звучит, например

Студенты, которые посещают все кружки, связанные с математикой

То меняется здесь только делитель. Вместо «все кружки» он будет «все кружки, связанные с математикой»

Тогда запрос изменится так

```
Select student.* from student
where not exists
(select * from CAS_activities
where name like '%математ%'
and not exists
(Select * from student student_activity as st_a
where student.id_st=st_a.id_stand st_a.id_cas= CAS_activities.id_cas))
```

Запросы на разность

Запрос на разность можно реализовать несколькими способами:

С помощью except, not in, not exists, left join.Рассмотрим на примере запроса «Группы, где есть Иванов, но нет Петрова»

Здесь из групп с Ивановым необходимо вычесть группы, где есть Петров.

Пишем запросы вычитаемое и уменьшаемое.

Уменьшаемое: Группы, где естьИванов

```
select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname='Иванов'
```

Вычитаемое:Группы, гдеестьПетров

```
select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname=' Петров'
```

Реализация с помощью except

```
select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname='Иванов'

except

select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname=' Петров'
```

Реализация с помощью not in

```
select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname='Иванов'

 and st_group.id_gr not in

(select st_group.id_gr from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname=' Петров')
```

Реализация с помощью not exists

```
select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname='Иванов'

And not exists(
```

```
select *from st_group as gr2 inner join student on student.id_gr=gr2.id_gr
where student.surname=’ Петров’

And st_group.id_gr=gr2.id_gr)
```

Реализация с помощью left join

```
select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr

Left join

(select st_group.* from st_group inner join student on
student.id_gr=st_group.id_gr where student.surname=’ Петров’

)q on st_group.id_gr=q.id_gr

where student.surname=’Иванов’  and q.id_gr is NULL
```

Примеры реализации различных типов запросов с подзапросами

Группа с максимальным количеством студентов

```
Select st_group.number, count (id_st) as cnt from student  join st_group on
student.id_gr=st_group.id_gr group by student.id_gr having count (id_st)
=(select max(cnt) from

  (select count (id_st) as cnt from student group by id_gr)q)
```

Через ALL

```
Select st_group.number, count (id_st) as cnt from student  join st_group on
student.id_gr=st_group.id_gr  group by student.id_gr  having count (id_st)>
=ALL (select count (id_st) as cnt from student group by id_gr)
```

Через WITH

```
with q1 as (select st_group.id_gr, number_gr, count(id_st) as cnt from
st_group join student on student.id_gr=st_group.id_grgroup by st_group.id_gr,
number_gr)select q1.id_gr, q1.number_gr, cnt from q1where cnt=(select
max(cnt) from q1);
```

Лабораторная работа №6 Разработка SQL запросов: запросы с подзапросами

Цель работы: Получение навыков создания запросов с подзапросами.

Задание и последовательность выполнения работы

По аналогии с примерами, приведенными в п. 1 реализовать запросы г) .. ж), указанные в варианте задания. Один из запросов на максимум/минимум реализовать с помощью директивы all. Запрос на «все» (реляционное деление) реализовать с помощью 2 not exists и с помощью агрегатной функции. Запросы

на разность реализовать в 3 вариантах: Not in,except(MySQL не поддерживает, поэтому только синтаксис), с использованием левого/правого соединения

Содержание отчета

- Цель работы
- текст задания(вместе с текстом варианта задания);
- физическую модель БД;
- элементы таблицы тестовых данных из 4 работы, касающиеся запросов г-ж:
- текст запросов на SQL;
- наборы данных, возвращаемые запросами (скриншоты);
- выводы об особенностях реализации запросов с подзапросами.

Контрольные вопросы и задания

- Есть таблица студент, в которой хранится его ид, фамилия, имя, отчество, кафедра и форма обучения. Как посчитать студентов на каждой кафедре?
- Есть таблица студент, в которой хранится его ид, фамилия, имя, отчество , кафедра и форма обучения. Как посчитать студентов каждой формы обучения на каждой кафедре?
- В чем разница между count(id) и count(distinct id)?Когда применяется каждый вариант?
- В чем разница между count(id) и count(*)?Когда применяется каждый вариант?
- Как получить значение агрегатной функции от значения другой агрегатной функции?
- В каком месте запросов могут использоваться подзапросы?

Хранимые процедуры и функции

Хранимые подпрограммы (процедуры и функции)

Хранимые подпрограммы — это поименованные блоки на языке SQL и его процедурных расширениях хранящиеся на сервере в скомпилированном виде, которые можно вызвать по имени, и при этом они могут принимать различные параметры. Есть два типа хранимых подпрограмм: процедуры и функции. И процедуры, и функции могут принимать заданный им вызывающей программой набор параметров и выполнить ряд действий. Оба типа подпрограмм могут

изменять и возвращать данные передаются им в качестве параметра. Разница между процедурой и процедурой заключается в том, что функция всегда будет обязательно возвращать одно значение вызывающей стороне, в отличие от процедуры, которая может не возвращать значений вообще или возвращать одно, или более значений.

Язык SQLвесьма богатый, однако, не для всех сложных действий достаточно декларативных запросов. Поэтому были созданы процедурные расширения языка SQL. В некоторых СУБД их использование необходимо указывать явно (для PostgreSQL это PL/pgSQL). Для PostgreSQL хранимые процедуры и функции можно создавать на языке SQL, когда вся процедура или функция представляет собой последовательность запросов SQL, так и на PL/pgSQL, если необходимо использовать условия и управляющие конструкции, ветвления или условия. Функция на языке SQL для PostgreSQL вернется результат последнего запроса.

В отличие от индивидуально обрабатываемых сервером отдельных SQL запросов, Хранимые процедуры и функции компилируется один раз, и хранятся в комбинированном виде в базе данных на сервере. Это позволяет получить следующие преимущества:

- Дополнительные запросы клиента и сервером минимизируются.
- Промежуточные результаты, которые не нужны клиентскому приложению, не нужно передавать между сервером и клиентом. Что актуально для сложных действий, которые невозможно выполнить одним запросом
- Можно избежать лишнего парсинга запросов при повторном вызове процедуры.
- Повышение безопасности данных за счет ограничения возможностей непосредственного обращения к данным таблицы, разрешив пользователям манипулировать данными только с помощью хранимых процедур, при наличии пользователя соответствующих прав.

У серверного программирования в целом и хранимых подпрограмм в частности есть и недостатки:

- Отделение части бизнес-логики приложения для реализации в СУБД
- Сложность тестирования и отладки. Это связано с тем, что ошибки в коде процедуры не проявляют себя до времени выполнения (вызова) процедуры.
- Плохая переносимость между СУБД.

В хранимых процедурах и функциях могут использоваться запросы SQL на выборку, манипулирование данными и определение данных, и специальные управляющие конструкции, а также переменные для записи значений.

Хранимые процедуры и функции создаются командами *CREATE PROCEDURE* и *CREATE FUNCTION* и вызываются командами *CALL* (EXEC для MSSQLServer) и присвоением возвращенного значения или его выборкой (*SELECT*) соответственно. Сравнение процедур и функций в MySQL и в PostgreSQL приведено в таблице 9.

Таблица 9. Сравнение процедур и функций в MySQL и в PostgreSQL

Отличия хранимых подпрограмм	MySQL	PostgreSQL
Языки для написания хранимых подпрограмм	SQL со встроенными процедурными расширениями	SQL, процедурное расширение PL/pgSQL языка SQL, встроенные процедурные элементы других языков PL/Tcl PL/Perl and PL/Python (соответственно языков Tcl ,Perl ,Python)
обновление тела подпрограммы	Только удалить процедуру и добавить заново	Можно обновлять тело используя CREATE OR REPLACE
Идентификация подпрограмм	Совокупность 1)имя подпрограммы, 2)имя базы данных	Совокупность 1)имя подпрограммы, 2)имя базы данных и схемы 3)количество и тип параметров [6]
Режим передачи параметров процедуры	IN, OUT, INOUT	IN, OUT, INOUT,VARIADIC*
Режим передачи параметров функции	IN	IN, OUT, INOUT,VARIADIC*
Тип возвращаемого	Любой допустимый тип	Любой допустимый тип

значения функции	данных MySQL (один объект)	данных PostgreSQL, включая пользовательские типы, триггер, таблица, множество объектов заданного типа

Режим VARIADIC, за атрибутом в котором может следовать не более 1 OUT параметра, используется для передачи переменного количества аргументов в виде массива, при условии, что все эти аргументы имеют один и тот же тип данных.

Приведем общий упрощенный синтаксис создания процедур и функций .

Синтаксис создания процедуры на MySQL приведен ниже:

```
CREATE PROCEDURE имя_процедуры ([параметр_процедуры[,...]])
[характеристика ...] тело_подпрограммы
CREATE FUNCTION имя_функции ([параметр_функции[,...]])
RETURNS тип
[характеристика ...] тело_подпрограммы
параметр_процедуры:
 [ IN | OUT | INOUT ] имя_параметра тип
параметр_функции:
 имя_параметра тип
тип:
 Любой тип данных MySQL
характеристика:
 LANGUAGE SQL
 | [NOT] DETERMINISTIC
 | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
 | SQL SECURITY { DEFINER | INVOKER }
 | COMMENT 'string'
тело_подпрограммы:
 Правильное  SQL выражение.
```

Синтаксис создания процедуры PostreSQL

```
CREATE [ OR REPLACE ] PROCEDURE
name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [,
...] ] )
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
| sql_body
  } ...
```

Полный синтаксис можно посмотреть на сайте с официальной документацией[11,13].

Хранимые процедуры и функции могут иметь следующие назначения:

- Для защиты данных таблиц от несанкционированного доступа;
- Для управления основными и справочными данными;
- Для логического удаления данных;
- Для сложной аналитики в транзакционных базах данных;

- Для каскадного удаления всех зависимых данных вне зависимости от декларативной ссылочной целостности;
- Для пакетного выполнения заданий.

Основные данные — данные, предоставляющие контекст для сведений о бизнес-деятельности, представленные в форме относящихся к этой деятельности общепринятых абстрактных понятий. Они включают описания (определения и идентификаторы) деталей внутренних и внешних объектов, вовлеченных в бизнес-процессы, таких как клиенты, продукты, сотрудники, продавцы и контролируемые области (значения кодов)[14].

Справочные данные — это любые данные, используемые для определения характеристик или классификации других данных, или же для соотнесения данных внутри организации с внешними. [14]. В основном справочные данные состоят из кодов и их описаний, но могут иметь и более сложную структуру.

В рамках управления справочными данными нам может понадобиться вставлять справочные данные в отдельные таблицы при внесении данных с ними связанных. Аналогичные действия могут понадобиться для вставки данных в таблицы соответствующие слабым сущностям.

Физическим (жестким удалением) удалением называется удаление строки из базы данных полностью. В отличие от этого логическим или мягким удалением называется процесс, при котором строка с удаляемыми данными сохраняется в базе данных, но в дополнительном столбце она помечается как удаленная. Сравнение особенностей типов удаления приведено в таблице 10.

Таблица 10 Сравнение логического и физического удаления

Отличия	**Физическое (жесткое) удаление**	**Логическое (мягкое) удаление**
Описание	Строка удаляется из БД полностью, занятый участок памяти освобождается и становится доступным для дальнейшего использования.	Строка сохраняется в БД, но в служебной части она помечается как удаленная. Возможно хранение временной метки удаления
Доступ к данным	Удаленное данное не доступно	сохраняется история (удобно для аудита) и зависимые данные

Память	Данные занимают меньше памяти	Объем памяти, занимаемый данными, постоянно растет
Запросы	Запросы проще	В запросах условия на актуальность
Ссылочная целостность	Возможность поддержания ссылочной целостности декларативно	Ссылочная целостность только активная
Назначение	Хранение информации о текущем состоянии предметной области	Хранение истории состояния предметной области

Переменные в процедурных расширениях SQL

В хранимых процедурах переменные используются для временного хранения данных в теле подпрограмм, а также для хранения возвращенных процедурой или функцией данных в вызывающей среде. Особенности переменных диалектах SQL для различных СУБД могут отличаться.

Во всех СУБД в рамках процедурных расширений существуют локальные переменные с областью видимости в рамках блока, в котором объявлены.

Локальные переменные с именами без использования символов кавычек могут содержать основные латинские буквы, цифры 0–9, доллар, подчеркивание. Со специальными символами, в которые заключена переменная, возможна поддержка Unicode. Спецсимволами для MySQL являются машинописные обратные апострофы (грависы)``, для PostgreSQL двойные кавычки “”, для MS SQL Server квадратные скобки [].

В MS SQL Server, MySQL до 8 версии имена всех локальных переменных начинаются с символа @.

В MySQL начиная с 8 версии, PostgreSQL, Oracle локальные переменные имеют имя, начинающееся с любого допустимого символа.

Дополнительно в MySQL встречается понятие **пользовательских** переменных, имена которых начинаются с символа @ и область видимости глобальная (все процедуры, функции, триггеры и вне их) в рамках сессии заданного пользователя в данном подключении к серверу. Пользовательские переменные могут использоваться для отладки или для подстановки в качестве фактических параметров выходных при вызове процедур.

Пользовательские переменные не надо объявлять, при этом их значение можно запросить в любой момент, но если в данной сессии переменная еще не была инициализирована, то её значением будет null.

При написании процедур, связанных со вставкой в дочерние таблицы, может понадобиться вставка в родительские и получение значения внешнего ключа.

Методы получения нового значения ключа зависят от СУБД и от того, является ли ключ автоинкрементным.

В общем случае для неавтоинкрементного ключа ключ для новой записи вычисляется как максимальное значение существующих ключей +1. Однако необходимо учитывать, что если таблица пустая, то функция *max* вернет null-значение, поэтому необходимо использовать дополнительно функцию, которая вместо null-значения подставит другое значение, с которого мы будем начинать отсчет ключей в пустой таблице. Для этой цели можно использовать функцию *coalesce*, которая принимает неограниченное число аргументов и возвращает первый не *null* аргумент, но если в списке таких нет, то возвращает *null*. Эта функция из стандарта SQL-92, поэтому поддерживается практически всеми реляционными СУБД.В некоторых СУБД существуют дополнительно функции 2 аргументов, выполняющих аналогичную задачу, то есть проверяют первый аргумент на пустоту и возвращают альтернативное значение (второй аргумент), если первый равен *NULL*. В MySQL, SQLite это функция *ifnull*, MSSQLServer – *isnull*.

Получить не автоинкрементный ключ таблицы st_group в переменную *id_gr_new* и вставить данные для MySQL можно при помощи следующего кода:

```
set id_gr_new =(select ifnull(max(id_gr)+1,0) from st_group );
INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
```

Для PostgreSQL код будет выглядеть так:

```
id_gr_new := (select coalesce(max(id_gr)+1,0) from st_group );
INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
```

Получение автоинкрементного ключа для различных СУБД будет различаться сильнее.

В MySQL используется функция LAST_INSERT_ID(), в PostgreSQL можно использовать 2 разных способа: один из которых возврат значения оператором INSERT, а второй функции currval, в которую в качестве аргумента подается имя последовательности. Приведем пример кода вставки для той же таблицы групп st_group для автоинкрементного ключа.

Код для MySQL

```
INSERT INTO st_group(num_gr) VALUES (gr_num);
set id_gr_new =LAST_INSERT_ID();
```

Код для PostgreSQL

```
INSERT INTO st_group(num_gr) VALUES (gr_num);
SELECT currval('st_group_id_seq');
```

Или

```
INSERT INTO st_group(num_gr) VALUES (gr_num)RETURNING
```

Синтаксис создания процедуры

Пример кода вставки студента в не автоинкрементную таблицу с получением внешнего ключа для СУБД MySQL

```
1. delimiter //
2. CREATE PROCEDURE ins_stud_n_group (gr_num varchar(8),name_ varchar(15),
3. surname_ varchar(20),patronym_ varchar(25),out id_stud int)
4. BEGIN
5. declare id_gr_new int;
6. declare id_st_new int;
7. if exists(select * from st_group where num_gr=gr_num)
8. then select id_gr into id_gr_new from st_group where num_gr=gr_num;
9. else begin
10. set id_gr_new =(select ifnull(max(id_gr)+1,0) from st_group );
11. INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
12. end;
13. end if;
14. set id_st_new = (select ifnull(max(id_st)+1,0) from student);
15. set id_stud = id_st_new;
16. insert into student (id_st, surname, name,patronym,id_gr)
17. VALUES (id_st_new,surname_,name_,patronym_,id_gr_new);
18. END; //
19. delimiter ;
20.call ins_stud_n_group('4131', 'Иванов', 'Иван', 'Иванович', @id_st_new);
21. select @id_st_new;
```

В строке 1 изменяется стандартный разделитель на символы // , а в строке 19 по завершению кода создания процедуры возвращается к разделителю по умолчанию.

В строках 2 и 3 представлен заголовок хранимой процедуры с входными параметрами: номером группы и фамилией, именем и отчеством, и выходным

параметром — идентификатором добавленного студента. Строки 5 и 6 – объявление локальных переменных для хранения значений первичных ключей группы (*id_gr_new*) и студента (*id_st_new*).

В строке 7 проверяется наличие группы с заданным номером, в случае наличия которого в строке 8 первичный ключ этой группы записывается в переменную *id_gr_new*. Если группу не нашли, то в строке 10 вычисляется значение ключа для новой записи в таблице с не автоинкрементным ключом, а затем в строке 11 добавляется группа с вычисленным ключом.

В строке 14 вычисляем новое значение ключа для студента, по умолчанию считая, что в процедуре добавляем нового и в следующей 15 строке, записываем его в выходной параметр. для упрощения записи можно было не использовать локальную переменную для ключа студента, а сразу записать в возвращаемое значение. В 16 и 17 строках добавляем студента с найденными выше значениями ключа студента и группы. Необходимо обратить внимание, что после окончания тела процедуры в строке обязательно ставится разделитель команд, заданный в строке 1, сигнализируя завершение команды создания процедуры.

В строке 20 вызывается функция вставки студента *ins_stud_n_group,* где для возврата значения первичного ключа вставляемого студента используется пользовательская переменная *@id_st_new*. Затем значение этой переменной выводится для пользователя в строке 21.

В ситуации, когда таблица групп имеет автоинкрементный ключ, необходимо будет заменить строки 10 и 11 на следующие строки:

```
10. INSERT INTO st_group(num_gr) VALUES (gr_num);
11. SET id_gr_new =LAST_INSERT_ID();
```

И аналогично для вставки в таблицу студентов с автоинкрементным ключом.

Пример кода вставки студента в не автоинкрементную таблицу с получением внешнего ключа для СУБД PostgreSQL

```
1. CREATE OR REPLACE PROCEDURE
2.   ins_stud_n_group   (   gr_num   varchar(8),name_   varchar(15),   surname_
varchar(20),patronym_ varchar(25),out id_stud int )
3. AS $$
4. declare
5. id_gr_new int default null;
```

```
6. id_st_new int default null;
7. BEGIN
8. if exists (select * from st_group where num_gr=gr_num)
9. then select id_gr into id_gr_new from st_group where num_gr=gr_num;
10. else begin
11. id_gr_new := (select coalesce(max(id_gr)+1,0) from st_group );
12. INSERT INTO st_group(id_gr,num_gr) VALUES (id_gr_new,gr_num);
13. end;
14. end if;
15. id_st_new = (select coalesce(max(id_st)+1,0) from student);
16. id_stud = id_st_new;
17.  insert into student (id_st, surname, name,patronym,id_gr) VALUES
(id_st_new,surname_,name_,patronym_,id_gr_new);
18. END;$$
19. LANGUAGE plpgsql;
```

Листинг программы для PostgreSQL для автоинкрементных таблиц группы и студента приведен ниже

```
1. CREATE OR REPLACE PROCEDURE
2.  ins_stud_n_group_autoinc ( gr_num varchar(8),name_ varchar(15), surname_
varchar(20),patronym_ varchar(25),out id_stud int )
3. AS $$
4. declare
5. id_gr_new int default null;
6. BEGIN
7. if exists (select * from st_group where num_gr=gr_num)
8. then select id_gr into id_gr_new from st_group where num_gr=gr_num;
9. else begin
10.  INSERT INTO st_group(num_gr) VALUES (gr_num) returning id_gr into
id_gr_new;
11. end;
12. end if;
13.
14. insert into student ( surname, name,patronym,id_gr)
15. VALUES (surname_,name_,patronym_,id_gr_new)returning id_st into id_stud ;
16. END;$$
17. LANGUAGE plpgsql;
18.  call ins_stud_n_group_autoinc ('4131', 'Иванов', 'Иван', 'Иванович',
null);
```

Для каскадного удаления достаточно будет написать запросы на удаление с подзапросами, начиная от самого вложенного запроса.

Управляющие конструкции

Присвоение значения переменным отличается для MySQL и PostgreSQL сравнение приведено в таблице 11.

Таблица 11. Сравнение присвоения в MySQL и PostgreSQL

Конструкция	MySQL	PostgreSQL
Синтаксис присвоения	`SET variable = expr [, variable = expr]`	`variable { := | = } expression;`

Пример присвоения	`SET X = 10;`	`X=10;`
Синтаксис записи через запрос на выборку	`SELECT ... INTO` *`перечень_переменных`* `FROM ...;` *`перечень_переменных`*- список переменных, разделенных запятыми.	`SELECT` *`select_expressions`* `INTO [STRICT]` приёмник *`результа`* `FROM ...;` Только для PL/pgSQL
Пример записи через запрос	`select id_gr into id_gr_new from st_group where num_gr='4131';`	`select id_gr into id_gr_new from st_group where num_gr=gr_num;`

В PL/pgSQL приёмником результата может быть переменная записи (агрегата), переменная — строка таблицы или список простых переменных и полей записи/строки, разделенных запятыми.

Кроме того, в PostgreSQLв PL/pgSQL есть возможность записать значение в переменную через запрос изменения данных следующим синтаксисом:

`INSERT| DELETE | UPDATE... RETURNING` *`expressions`* `INTO [STRICT]` *`target`*`;`

Например:

```
INSERT INTO st_group(num_gr) VALUES (gr_num) returning id_gr into id_gr_new;
```

Опция STRICT требует, чтобы команда вернула ровно одну строку или сообщение об ошибке времени выполнения: *NO_DATA_FOUND* (нет строк) или *Too_MANY_ROWS* (более одной строки).

Условный оператор

Условный операторMySQL и PostgreSQL имеет почти одинаковый синтаксис. Синтаксис для PostgreSQL от синтаксиса MySQL отличается только написанием *ELSIF* вместо *ELSEIF*. В обоих случаях условный оператор имеет 2 необязательные части: ветку else и дополнительные ветки выбора else if и заканчивается обязательным end if. Синтаксис и пример условного оператора MySQL и PostgreSQLприведен в таблице 12.

Таблица 12 Условный оператор

	MySQL	PostgreSQL
Синтаксис условного	`IF`*`условие1`*`THEN`*`операторы1`* `[ELSEIF`*`условие2`*`THEN`*`оператор`	`IF`*`условие1`*`THEN`*`операторы1`* `[ELSIF`*`условие2`*`THEN`*`операторы`

оператора	*ы2*]... [ELSE*операторы3*] ENDIF	*2*] ... [ELSE *операторы3*] END IF

Пример использования условного оператора.
Процедура/функция, выводящая текстовую информацию о положительности аргумента.
	delimiter //
	create procedure test_if(numbe int)
	begin
	declare result varchar(20) default null;
	IF (numbe= 0) THEN
	set result = 'zero';
	ELSEIF numbe > 0 THEN
	set result = 'positive';
	ELSEIF numbe < 0 THEN
	set result = 'negative';
	ELSE
	set result = 'NULL';
	END IF;
	select result;
	end;//
	delimiter ;
	call test_if(1);
		create function test_if(numbe int) RETURNS varchar(20)
		LANGUAGE 'plpgsql'
		as $$
		declare result varchar(20) default null;
		begin
		IF (numbe= 0) THEN
		set result = 'zero';
		ELSIF numbe > 0 THEN
		result = 'positive';
		ELSIF numbe < 0 THEN
		result = 'negative';
		ELSE
		set result = 'NULL';
		END IF;
		return result;
		end;$$
		select test_if(1);

Циклы

Чаще всего циклы используются для поэлементной обработки данных или вставки данных, для получения нужны вычисления. Сравнение циклов в MySQL и PostgreSQL приведено в таблице 13

Таблица 13. Работа с циклами в СУБД

	MySQL	PostgreSQL
Цикл с	[*begin_label*:]WHILE *условие_продолжения_цикла* DO	[<<*label*>>] WHILE *условие_продолжения_цикла*

предусловием. Синтаксис	*операторы_тела_цикла* END WHILE[*end_label*]	LOOP *операторы_тела_цикла* END LOOP [*label*];

Пример использования цикла. Добавление данных по встречам секции раз в неделю в заданный день до заданной даты, day_of_week_num- номер дня недели от1до7, начиная с понедельника	create table cas_shedule (
	id_shedule int auto_increment not null primary key,
	id_cas int not null,
	date_meet date,
	foreign key (id_cas) references cas_activities (id_cas) on delete cascade on update restrict);
	
	delimiter //
	create procedure insert_dates_cas(id_cas_ int,day_start date,day_end date,day_of_week_num smallint)
	begin
	DECLARE dt DATE DEFAULT null;
	set day_of_week_num=day_of_week_ num-1;
	set dt= DATE_ADD(day_start, INTERVAL (7+day_of_week_num-WEEKDAY(day_start))mod 7 day);
	if exists (select * from cas_activities where id_cas=id_cas)and(dt is not null)and(day_end is not null) then
	begin
	WHILE dt <= day_end DO
	insert into cas_shedule (id_cas,date_meet) values(id_cas_,dt);
	SET dt = DATE_ADD(dt,INTERVAL 1 week);
	END WHILE;
	end;
	end if;
	end; //
	delimiter ;
	call insert_dates(1,'2023-09-01','2023-10-01',1);
		create table cas_shedule (
		id_shedule serial not null primary key,
		id_cas int not null,
		date_meet date,
		foreign key (id_cas) references cas_activities (id_cas) on delete cascade on update restrict);
		create procedure insert_dates_cas(id_cas_ integer,day_start date,day_end date,day_of_week_num smallint)LANGUAGE 'plpgsql'
		as $$
		DECLARE dt DATE DEFAULT null;
		begin
		dt=day_start+ mod(7+day_of_week_num-(extract(isodow from day_start)),7)::integer;
		if exists (select * from cas_activities where id_cas=id_cas)and(dt is not null)and(day_end is not null) then
		begin
		WHILE dt <= day_end LOOP
		insert into cas_shedule (id_cas,date_meet) values(id_cas_,dt);
		dt = date(DATE_ADD(dt,'1 week'::interval));
		END LOOP;
		end;
		end if;
		end;
		$$
		call insert_dates_cas(1,'2023-09-01'::date,'2023-10-01'::date,2::smallint);

Отдельно есть уникальные циклы для СУБД. В PostgreSQLподдерживается цикл со счетчиком следующего синтаксиса:

```
[ <<label>> ]
FOR перем_счетчик IN [ REVERSE ] нач_зн .. конеч_зн [ BY шаг ] LOOP
операторы_тела_цикла
END LOOP [ label ];
```

Пример использования цикла со счетчиком. Процедура добавление заданного количества встреч секции раз в неделю в заданный день

day_of_week_num- номер дня недели от1до7, начиная с понедельника.

```
create or replace procedure insert_dates_cas_for(id_cas_ integer,day_start
date,amount integer,day_of_week_num smallint)LANGUAGE 'plpgsql'
as $$
DECLARE dt DATE DEFAULT null;
declare i integer;
  begin
  dt=day_start+            mod(7+day_of_week_num-(extract(isodow            from
day_start)),7)::integer;
if exists (select * from cas_activities where id_cas=id_cas)and(dt is not
null)and(amount is not null) then
begin
FOR i IN  1 .. amount  BY 1
  LOOP
    insert into cas_shedule (id_cas,date_meet) values(id_cas_,dt);
  dt = date(DATE_ADD(dt,'1 week'::interval));
   END LOOP;
 end;
 end if;
end;
$$
call insert_dates_cas_for(2,'2023-09-01'::date,5,3::smallint);
```

Управление правами доступа

Для управления правами доступа в SQL существует ряд команд:

CREATEUSER — создать пользователя.

DROP USER — удалить пользователя.

CREATE ROLE — создать роль.

DROPROLE — удалить роль.

GRANT —Дает права на различные объекты роли и пользователю или назначает роли для пользователей.

REVOKE —отозвать права доступа у роли или пользователя или лишить пользователя роли

Команды общие для большей части СУБД, но отличающиеся особенностями синтаксиса.

PostgreSQL

CREATE USER — это псевдоним CREATE ROLE[13]. Единственное отличие состоит в том, что когда команда пишется CREATE USER, по умолчанию предполагается опция LOGIN, означающая, что роль или пользователь может

быть указан в качестве имени авторизации начального сеанса во время подключения клиента. тогда как NOLOGIN предполагается, когда команда пишется CREATE ROLE, что не разрешает роли вход в систему.

Пример работы с пользователями и правами.

Листинг создания пользователя, который может добавлять данные в базу только через процедуры, но не имеет доступа для редактирования самих таблиц дляMySQLприведен ниже

```
CREATE USER 'for_proc'@'localhost' IDENTIFIED BY '111';
GRANT select ON uni.* TO 'for_proc'@'localhost';
GRANT EXECUTE ON PROCEDURE uni.`ins_stud_n_group` TO 'for_proc'@'localhost';
SHOW GRANTS  FOR  'for_proc'@'localhost' ;
```

Вначале создается пользователь, затем ему предоставляются права на чтение всех объектов базы данных, затем разрешение на выполнение процедуры. Последняя команда выводит его права.

Если реализовывать не через права конкретного пользователя, а через введение ролей, которые потом выделяются конкретным пользователям путь немного длиннее.

```
1. CREATE ROLE 'deans_office';
2. GRANT select ON uni.* TO 'deans_office';
3. GRANT EXECUTE ON PROCEDURE uni.`ins_stud_n_group` TO 'deans_office';
4. CREATE USER 'deans_office_worker'@'localhost' IDENTIFIED BY '111';
5. GRANT 'deans_office' TO 'deans_office_worker'@'localhost';
6. SET DEFAULT ROLE ALL TO 'deans_office_worker'@'localhost';
7. SHOW GRANTS FOR 'deans_office_worker'@'localhost' using 'deans_office';
```

В строке 1 –создаем роль, в строках 2 и 3

Для создания роли в PostgreSQL используется команда CREATEROLE

```
CREATE ROLE name [ [ WITH ] option [ ... ] ]
```

```
CREATE ROLE deans_office;
 GRANT select ON ALL TABLES IN SCHEMA public TO deans_office;
GRANT EXECUTE ON PROCEDURE ins_stud_n_group TO deans_office;
GRANT EXECUTE ON PROCEDURE
ins_stud_n_group(varchar(8),varchar(15),varchar(20),varchar(25), int)  TO
deans_office;
CREATE USER deans_office_worker PASSWORD '111';
GRANT deans_office TO deans_office_worker;
 SET ROLE 'deans_office_worker';
SELECT *  FROM information_schema.routine_privileges
 WHERE grantee ='deans_office';
   SELECT * from information_schema.table_privileges
 WHERE grantee ='deans_office';
```

Однако если попробовать запустить процедуру будет ошибка доступа, так как у пользователя нет права на вставку в таблицу. Чтобы права при запуске

процедуры рассматривались владельца процедуры, необходимо при создании или изменении процедуры в конце после указания языка добавить SECURITY DEFINER.

логическоеудаление

```
alter table st_group add (is_deleted boolean default false, date_delete
datetime);
alter table student add (is_deleted boolean  default false, date_delete
datetime);
Delimiter //
create procedure logical_delete_group(id_group int)
begin
declare time_del datetime default now();
update st_group set is_deleted=true,date_delete=time_del where id_gr=id_group
;
update student set is_deleted=true,date_delete=time_del
where id_gr=id_group
   and is_deleted=false
   and id_st>0;
end;//
delimiter ;

call logical_delete_group(10);
```

Для PostgreSQLнесмотря на отсутствие управляющих конструкций, из-за наличия переменной требуется язык Plpgsql.

```
alter table st_group add is_deleted boolean default false;
alter table st_group add date_delete timestamp;
alter table student add is_deleted boolean default false;
alter table student add date_delete timestamp;
CREATE OR REPLACE PROCEDURE
logical_delete_group(id_group int)
AS $$
declare time_del timestamp default now();
BEGIN
update st_group set is_deleted=true,date_delete=time_del where id_gr=id_group
;
update student set is_deleted=true,date_delete=time_del
where id_gr=id_group
   and is_deleted=false
   and id_st>0;
END;$$LANGUAGE plpgsql;
```

Лабораторная работа №7 Хранимые процедуры. Управление доступом

Цель работы: Получения навыков по серверному программированию и управлению доступом в базе данных

Задание и последовательность выполнения работы

1)Создать в БД хранимые процедуры, реализующие:

— вставку с пополнением справочников (вставляется информация о студенте, если указанный номер группы отсутствует в БД, запись добавляется в таблицу с

перечнем групп) (получаем ссылку на внешний ключ по значению данного из родительской таблицы);

— хранимую функцию любого назначения, соответствующего предметной области.

— каскадное удаление: удаление всех зависимых данных (перед удалением записи о группе удаляются записи обо всех студентах этой группы и их секциях);

2)Создать пользователя, который обладает правами, указанными в варианте задания

Содержание отчета

- Цель работы
- Текст задания вместе с вариантом задания.
- Физическую модель БД.
- Назначение разработанных хранимых процедур или функций текстом
- Скрипт для создания хранимых процедур или функций;
- Скрипт создания пользователя, с возможностью изменять данные только через процедуры и функции
- SQL операторы и скриншоты наборов данных, иллюстрирующие работу процедур.
- SQL операторы и скриншоты наборов данных, отображающие права пользователя и демонстрирующие его работу с процедурамиработу.
- Выводы об использовании процедур и управление доступом в разработанной Вами базе данных

Контрольные вопросы и задания

- Что такое хранимая процедура?
- В чем отличие хранимых процедур и функций?
- В чем отличие локальной и пользовательской переменной в MySQL?
- Как вернуть значение из хранимой процедуры?
- Как получить новое значение ключа пари вставке данных в хранимой процедуре?

– Чем отличается логическое и физическое удаление?
– Как дать права пользователю на чтение таблицы?
– Как назначить роль пользователю?

Триггеры

Определение, классификация и назначение триггеров.

Триггер — это хранимая процедура, которая не вызывается непосредственно, а исполняется при наступлении определенного события (вставка, удаление, обновление строки).

Отличия триггеров от других хранимых процедур:

- Вызываются событием, нельзя вызвать вручную
- Нельзя вызвать из внешнего интерфейса (клиентского приложения)
- Не имеют параметров
- Не могут быть функциями, т.е. возвращать значения

Состав триггеров

- **событие —** операция в базе, вызвавшая триггер
- **Время** вызова триггера, относительно операции
- **условие—** это логическое выражение, которое должно принимать значение TRUE для того, чтобы было выполнен триггер
- **действие** — тело триггерной процедуры

Назначение триггеров (наиболее частое):

- Поддержание ссылочной целостности
- Передача пользователю предупреждения об ошибках или сообщений о данных
- Отладка (т.е. отслеживание ссылок на указанные переменные и/или контроль над изменениями состояния этих переменных).
- Аудит (например, регистрация информации о том, кто и когда внес те или иные изменения в определенные переменные отношения).
- Измерение производительности (например, регистрация времени наступления или трассировка указанных событий в базе данных).

- Проведение компенсирующих действий (например, каскадная организация удаления кортежа поставщика для удаления также соответствующих кортежей поставок).
- Логическое удаление

Триггеры по времени действия и назначению делятся на триггеры до, вместо и после. Триггеры **До и вместо** используются для каскадного удаления, обработки ошибок, сохранения старых значений, отладка, шифрование данных Триггеры **После** используются для логирования изменений, проведение компенсирующих действий (Удаление с очисткой справочника, расчет вычислимого поля)

Типы триггеров по способу обработки команд:

- Для каждой обработанной строки —FOREACHROW, которые запускаются для каждой строки таблицы, затронутой изменением;
- Для каждой обработанной команды—FOREACHSTATEMENT, которые запускаются для каждой из инструкций ***insert***, ***update***, ***delete***, применяемой к таблице.

Во время выполнения триггера и для MYSQL и PostgreSQL существуют две строки NEW и OLD, повторяющие структуру таблицы на которую поставлен триггер(в других диалектах SQL таблицы ***inserted*** и ***deleted)***. NEW содержит новые версии данных (вставленные оператором ***insert*** или измененные оператором ***update***). OLD содержит старые версии данных (удаленные оператором ***delete*** или подлежащие изменению оператором ***update***). Ссылка на указанные таблицы производится так же, как на основные таблицы БД.

Реализация триггеров для MySQL и PostgreSQL.

Поддержка триггеров в MySQL началась с версии 5.0.2. MySQL использует стандарт ANSI SQL:2003 для процедур и других функций.

Триггеры MYSQL

Синтаксис создания триггера

```
CREATE TRIGGER trigger_name trigger_time trigger_event
ON tbl_name FOR EACH ROW trigger_stmt
```

trigger_name — название триггера

trigger_time — Время срабатывания триггера. BEFORE — перед событием. AFTER — после события.

trigger_event — Событие:

insert — событие возбуждается операторами insert, data load, replace

update — событие возбуждается оператором update

delete — событие возбуждается операторами delete, replace.

Операторы DROP TABLE и TRUNCATE не активируют выполнение триггера

tbl_name — название таблицы

trigger_stmt выражение, которое выполняется при активации триггера

Пример триггера, реализующего ссылочную целостность (каскадное изменение)

Создадим простой триггер, который при удалении группы будет удалять всех её студентов:

```
Create trigger my_trigger
before delete on st_group FOR EACH ROW
Begin delete from student where id_gr =OLD.id_gr;
End//
```

Пример триггера для вычисленных полей

Еще один пример триггера после обновления с вычислимым столбцом для таблиц student и st_group с учетом наличия в группе столбца `stud_count`.

```
Create trigger my_trigger after update on student FOR EACH ROW
Begin
Update st_group Set stud_count=stud_count-1 where id_gr =OLD.id_gr;
Update st_group Set stud_count=stud_count+1where id_gr =NEW.id_gr; End//
```

Пример триггера бэкапа данных

Создадим еще одну таблицу, в которой будут храниться резервные копии строк из таблицы st_group, а затем триггеры бэкапа при обновлении и удалении

```
CREATE TABLE backup_group (
`id` INT( 11 ) UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
```

```
`row_id` INT( 11 ) UNSIGNED NOT NULL,
`num_gr` varchar(10) NOT NULL
) ENGINE = MYISAM
DELIMITER |
CREATE TRIGGER `update_gr` before update ON `st_group`
FOR EACH ROW BEGIN
  INSERT INTO backup_group Set row_id = OLD.id, num_gr = OLD. num_gr t;
END;|
CREATE TRIGGER `delete_test` before delete ON `test`
FOR EACH ROW BEGIN
  INSERT INTO backup_group Set row_id = OLD.id, num_gr = OLD. num_gr;
END |
DELIMITER ;
```

Пример триггера с сигналом об ошибке

```
create table trigger_test(id int not null);
delimiter //
create trigger trg_trigger_test_ins before insert on trigger_test for each row
begin
declare msg varchar(255);
if new.id < 0 thenset msg = concat('MyTriggerError: Trying to insert a negative
value in trigger_test: ', cast(new.id as char));
signal sqlstate '45000' set message_text = msg;
end if; end//
delimiter ;
```

Синтаксис DROP TRIGGER

```
DROP TRIGGER [schema_name.]trigger_name
```

Оператор удаляет триггер. Имя базы данных опционально.

Синтаксис создания триггера в PostgreSQL

Триггеры в PostgreSQL могут быть и На уровне строк(FOREACHROW) и На уровне оператора(FOREACHSTATEMENT).

Тело триггера в PostgreSQL представляет вызов хранимой функции, которая возвращает тип «триггер»:

```
CREATE OR REPLACE FUNCTION имя_функции() RETURNS trigger
```

Общие элементы синтаксиса создания триггера приведены в таблице 14.

Таблица 14. Синтаксис триггера

Элементы триггера	**Postgre**SQL

Общий синтаксис	CREATE [OR REPLACE] [CONSTRAINT] TRIGGER *trigger_name trigger_time trigger_event* ON *table_name* [FROM *referenced_table_name*] [NOT DEFERRABLE | [DEFERRABLE] [INITIALLY IMMEDIATE | INITIALLY DEFERRED]] [REFERENCING { { OLD | NEW } TABLE [AS] *transition_relation_name* } [...]] [FOR [EACH] { ROW | STATEMENT }] [WHEN (*condition*)] EXECUTE *trigger_body*
триггерное время trigger_time	{ BEFOR E| AFTER | INSTEAD OF }
триггерное событие trigger_event	{INSERT |DELETE|TRUNCATE|UPDATE [OF *column_name* [, ...]] } *event* [OR ...]
referenced_table_name	Имя другой таблицы, на которую ссылается ограничение. используется для ограничений внешнего ключа и не рекомендуется для обычного применения. допускается только для триггеров ограничений.
тело триггера	{ FUNCTION | PROCEDURE } *function_name* (*arguments*)

Пример реализации подсчёта PostgreSQL

```
CREATE OR REPLACE FUNCTION calc_stud_gr_after() RETURNS trigger
AS $$
BEGIN
update st_group set stud_count = stud_count + 1 where st_group.id_gr =
new.id_gr;
update st_group set stud_count = stud_count - 1 where st_group.id_gr =
old.id_gr;
RETURN NEW;
END;$$ LANGUAGE plpgsql;
CREATE TRIGGER calc_stud_gr_after AFTER UPDATE OF id_gr
 ON student
FOR EACH ROW EXECUTE PROCEDURE calc_stud_gr_after()
```

Пример реализации проверки PostgreSQL

```
CREATE OR REPLACE FUNCTION insert_existing_gr() RETURNS trigger
AS $$
BEGIN
IF EXISTS (SELECT * FROM st_group WHERE st_group.num_gr = NEW.num_gr)
THEN RAISE EXCEPTION 'Невозможно добавить группу %, так как группа с данным
номером уже существует', NEW.num_gr;
END IF;
RETURN NEW;
END;$$
LANGUAGE plpgsql;

CREATE TRIGGER insert_existing_gr1 BEFORE INSERT ON st_group
FOR EACH ROW EXECUTE PROCEDURE insert_existing_gr()
```

Синтаксис DROP TRIGGER

```
DROP TRIGGER [ IF EXISTS ] name ON table_name [ CASCADE | RESTRICT ]
```
Удаляет существующий триггер . При указании CASCADE удаляет зависимые от него объекты

Лабораторная работа №8 Триггеры. Обеспечение активной целостности данных базы данных

Цель работы: Получение умений и навыков по проектированию и созданию триггеров баз данных, включая их использование для поддержания активной ссылочной целостности.

Задание и последовательность выполнения работы

Реализовать для своей базы данных триггеры для всех событий (insert, delete, update) до и после.(6 триггеров) Часть из которых будет обеспечивать ссылочную целостность, остальные могут иметь другое назначение из других предложенных , но не менее 2 различных .

- Вычисление/поддержание в актуальном состоянии вычислимых (производных) атрибутов
- логирование (запись) изменений;
- проверка корректности проводимых действий.).

Вычислимые поля можно добавить при необходимости.

Содержание отчета

- Цель работы
- Текст задания вместе с вариантом задания.
- Физическую модель БД.
- Назначение разработанных триггеров текстом
- Скрипт для создания триггеров;
- SQL операторы и скриншоты наборов данных, иллюстрирующие работу триггеров.
- Выводы об использовании триггеров в разработанной Вами базе данных.

Контрольные вопросы и задания

- Что такое триггер?
- Чем триггеры отличаются от других хранимых процедур?
- Как внутри триггера обратится к добавленным и удаленным данным?

– Чем отличаются триггеры before и after по выполнению и назначению?
– Как вернуть из триггера значение?
– Как передать в триггер параметры?

Литература

1. Осипов Д. Л. Технологии проектирования баз данных. – М.: ДМК Пресс, 2019. –498 с.: ил.
2. Коннолли Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика: пер. с англ. / Т.Коннолли, К.Бегг. - М. и др.: Вильямс, 2017. - 1439 с
3. Шустова, Л. И. Базы данных : учебник / Л.И. Шустова, О.В. Тараканов. — Москва : ИНФРА-М, 2023. — 304 с. + Доп. материалы [Электронный ресурс]. — (Высшее образование: Бакалавриат). —URL: https://znanium.com/catalog/product/1986697 (дата обращения: 18.10.2023).
4. Valacich Joseph S. Modern Systems Analysis and Design 9th edition/ Valacich Joseph, George Joey F-— New York: Pearson, 2020. — 529 с.
5. Дейт, К. Введение в системы баз данных. Восьмое издание. – Москва.: Диалектика,2019— 1328 с.
6. Волк В. К. Базы данных. Проектирование, программирование, управление и администрирование : учебник /В. К. Волк.— Санкт-Петербург : Лань, 2020.
7. Садаладж П. Д. NoSQL: новая методология разработки нереляционных баз данных / Прамодкумар Дж. Садаладж, Мартин Фаулер ; [пер. с англ. и ред. Д. А. Клюшина]. - Москва: Вильямс, 2013
8. Time Series Databases and Their Importance https://expeed.com/time-series-databases-and-their-importance/(датаобращения: 19.10.2023)
9. Date, C.J.: Database Design and Relational Theory: Normal Forms and All That Jazz,. APRESS, Healsburg (2019)- 450 с.
10. Домбровская Г. Оптимизация запросов в PostgreSQL / Домбровская Г., Новиков Б., Бейликова А. , пер. с англ. Д. А. Беликова. – М.: ДМК Пресс, 2022. – 278 с.: ил.

11. Справочное руководство по MySQL 8.1 URL: https://dev.mysql.com/doc/refman/8.1/en/ (дата обращения 22.10.2023)
12. Документация на русском языке к PostgreSQL 15.4 URL: https://postgrespro.ru/docs/ (дата обращения 22.10.2023)
13. Официальная документация PostgreSQL 16 версии https://www.postgresql.org/docs/16/index.html(дата обращения 20.10.2023)
14. DAMA-DMBOK : Свод знаний по управлению данными. Второе издание / DamaInternational [пер. с англ. Г. Агафонова]. — Москва : Олимп–Бизнес, 2020. — 828 с

Приложение 1 Распределение баллов

Семестр 5

	Наименование лабораторной	Количество баллов	Предельный № недели сдачи
1.	Разработка концептуальной модели предметной области	5	3
2.	Разработка физической модели базы данных с учетом декларативной ссылочной целостности	3	5
3.	Создание и модификация базы данных и таблиц базы данных	3	7
4.	Заполнение таблиц и модификация данных	4	9
5.	Разработка SQLзапросов: виды соединений и шаблоны	5	11
6.	Разработка SQLзапросов: запросы с подзапросами	7	13
7.	Хранимые процедуры. Управление доступом	6	15
8.	Триггеры. Обеспечение активной целостности данных базы данных	7	17
	Итого	40	

Приложение 2 Варианты заданий лабораторных 1-8

1. Программа для рисования графов: название вершины, координаты левой верхней точки отображаемой вершины, и её размеры, автор графа,типы ребер графа(направленные и нет), вес ребер.
   а. Вершины, текст которых содержит слово «выход», но не заканчивается на него
   б. Пользователи-авторы, у которых графов
   в. Графы, в которых есть вершина , с петлёй (Петля в графе — ребро, инцидентное одной и той же вершине.)
   г. Графы, ширина которых превышает 200 пикселей. Ширина графа в пикселях (от максимальной сумма координаты по горизонтали с шириной отнять минимальную левую координату)
   д. авторы графов, с наибольшим количеством ребер
   е. Вершины, для которых есть исходящие ребра, ведущие ко всем остальным вершинам её графа
   ж. Авторы, у которых в графах нет вершин со словом «начало» в названии вершины
2. Садоводство: участки, владельцы с учетом совместной собственности, линии/номер участка, площадь стоимость постройки, тип построек ,электрические счетчики владельца участка
   а. номера участков владельцев с отчеством, заканчивающимся на «ич», и начинающиеся на букву «Б»
   б. участки, на которых зарегистрировано более 1 типа постройки
   в. Тип (типы) построек, которые отсутствуют на участках
   г. Владелец (владельцы) участка максимальной площади
   д. Владельцы участков числом типов построек больше среднего
   е. Владельцы, имеющие на всех своих участках электрические счетчики
   ж. Участки, на которых нет счетчиков, но есть туалеты

3. ТСЖ: квартира, владелец, количество комнат, площадь, жилая площадь, этаж, коммунальные платежи
   а. квартиры владельцев, отчества которых заканчиваются на «на» и начинаются на «А»
   б. владельцев, у которых есть трехкомнатные квартиры и квартиры -студии
   в. этажи, на которых нет квартир
   г. владельцы квартир с минимальной площадью
   д. тип платежа, который оплатило меньше всего людей
   е. этажи в квартирах, которых суммарно оплатили все типы платежей

ж. владельцы, у которых нет двухкомнатных квартир, но заплачены платежи за вывоз ТКО в июне прошлого года

4. парк: деревья ,породы, дата высадки, дата обрезки, расположение, аллеи
   а. аллеи, на которых встречаются тополя, которые имеют в названии уточнение кроме слова тополь
   б. деревья, стоящие на перекрёстке аллей: Тройной липовой аллеи и Театральной аллеи
   в. породы, не высаженные в парке
   г. аллея, на которой растут деревья, которые обрезали позже всех
   д. породы, деревьев которой меньше всего в парке с ид 1
   е. порода дерева, встречающаяся на всех аллеях заданного парка
   ж. аллея, на которой растут липы, но не высаживались деревья в 2021 году

5. расписание экзаменов: даты экзаменов, дисциплина, преподаватель, группа, аудитория, один экзамен может вести несколько преподавателей
   а. аудитории, которых проходят экзамены по дисциплинам, начинающиеся словом «математ», но это не единственная часть названия
   б. дисциплина(дисциплины), по которой экзамены проходят в и в аудитории 23-10 и в 23-16
   в. преподаватели, которые не принимают экзаменов
   г. Преподаватели, принимающие самые последние экзамены
   д. группа, у которой меньше среднего экзаменов
   е. аудитория, в которой проходят экзамены у всех групп 4 факультета
   ж. дата, в которую проходят экзамены только у группы 4131

6. поликлиника: прием, пациент, процедура/прием, врач, стоимость
   а. врачи, проводившие любые процедуры, с аппаратурой , связанной с лазером (лазер в любом месте названия)
   б. врач, проводивший одному пациенту и прием и процедуру в разных кабинетах
   в. процедуры, которые никому не делали
   г. процедуры, с наименьшей стоимостью среди всех процедур, которые проводил врач Иванов Иван Иванович
   д. кабинет, в котором проводилось больше среднего приемов
   е. пациент, которому проводились все процедуры, название которых начинается на «П»
   ж. врач, принимавший Иванова Петра, но не проводивший процедур с аппаратами магнитотерапии.

7. личный кабинет: дисциплина, работа, тип работы, студент, преподаватель, дата сдачи , баллы
   а. задания, начинающиеся со слова «разработка», но это не единственные слово в них
   б. дисциплина, по которой есть лабораторные и другой вид заданий
   в. задание, по которому не прикреплено ни одной работы
   г. студент, сдавший курсовую по ООП позже всех
   д. дисциплина, по которой в этом году прикрепили число работ больше среднего
   е. преподаватель, за которым закреплены все виды занятий группы 4332 по дисциплине «Объектно-ориентированное программирование»
   ж. группа, студенты которой не прикрепляли отчетов по лабораторным работам, но прикрепляли по курсовым работам

8. костюмерная театра: роль, спектакль, название костюма, деталь костюма, размер, автор модели, дата разработки
   а. авторы, разрабатывавшие костюмы, имеющие в названии слово «принцесса», но не начинающееся с него
   б. костюмы, в которых есть и плащ и брюки
   в. спектакль, которому пока не добавили ролей
   г. спектакли, к которым разрабатывались самые старые из костюмов
   д. автор, разработавший костюм с числом деталей меньше среднего
   е. роль, в костюме к которой есть все типы деталей
   ж. автор, не разрабатывавший костюмы к «Валькирии», но разрабатывавший костюмы в текущем году

9. охраняемые парковки: адрес парковки, машина, владелец, место, рег. номер машины, дата и время заезда, дата и время выезда
   а. все парковки, расположенные на улицах, в названии которых есть слово «Морская», но на него название не начинается
   б. владелец машины, у которого несколько машин разных марок
   в. район, в котором нет парковок
   г. парковка, открывающаяся позже всех
   д. улица, количество парковок на которой меньше среднего
   е. машина, которая стояла на всех парковках Центрального района
   ж. владелец, парковавшийся на Невском проспекте, но не парковавшийся в Адмиралтейском районе

10. База данных по теме «сбор в поездку» должна иметь структуру, позволяющую реализовать следующие запросы:
    а. Найти какие поездки за текущий год имели в названии слово «экскурсия», но не заканчиваются на него
    б. Найти категорию без предметов
    в. Найти в какой тип поездок брались и зажигалка, и нож
    г. Найти пользователя, совершившего самую последнюю поездку в прошлом году
    д. Какие категории вещей берутся в поездки всех типов
    е. Какие категории вещей не берутся в поездку типа «конференция», но берутся на в на экскурсии
    ж. Найти тип(ы) поездок с количеством поездок, меньше среднего.

11. База данных сериалов.
    а. Сериал, в названии которого есть слово дом, но оно не последнее
    б. Персонаж, которого в разных сезонах играют разные актеры
    в. Сериал, в котором не заявлено ни одного персонажа.
    г. Сезон и сериал этого сезона, с количеством эпизодов больше среднего
    д. жанр самого старого сериала
    е. сезон, в котором встречаются все персонажи сериала
    ж. Актер, который играл в исторических сериалах, но не играл в 2021 году

12. вакансии: название вакансии, организация работодатель, адрес работодателя, диапазон зарплаты, требования к образованию, Обязанности, график работы, требования обязательные, желательные, дата выставления вакансии
    а. вакансии, имеющие в названии «данны»(от слов данные/данных), но не заканчивающиеся на него
    б. график, не присутствующий ни в одной вакансии
    в. работодатели, выставившие вакансии и в Москве, и в Санкт-Петербурге
    г. вакансия с максимальной зарплатой по нижней границе вилки зарплат
    д. вакансии с количеством обязательных требований ниже среднего
    е. требование, присутствующее во всех вакансиях на переводчика (любые вакансии со словом перевод)
    ж. вакансии, которые есть в Санкт-Петербурге, но которых нет у Газпрома

13. калькулятор бюджета семьи: категория дохода (продажа, зарплата), категория расхода (еда, счета за КУ, здоровье …), статьи дохода и расхода, дата расхода/дохода. Категория- более общее понятие чем статья. Например категория- еда, а статьи в ней мясо, рыба, вкусное к чаю, а конкретный расход «печенье «курабье» к чаю 11.09»

а. доходы всех категорий название, которых (категорий) содержат слово «дар», но не заканчиваются на него
б. месяц, в котором были статьи расхода на транспорт и на здоровье у одного и того же пользователя
в. категория без доходов
г. категория, по которой были доходы в этом году максимальные по единоразовой величине
д. категория, по которой не было расходов в январе, но были в мае
е. категория расхода, по которой траты были у всех членов семьи
ж. месяц, в котором были расходы количества статей меньше среднего

14. Книга контактов: люди, метки (друзья, коллеги, семья), дни рождения, телефон, электронная почта, примечания к человеку или контакту, праздники, которые празднуют люди

а. родственники, у которых номер телефона заканчивается на 44
б. Люди (контакты) без электронной почты
в. люди, которые празднуют и 23 февраля и Рождество
г. самые старые люди среди празднующих 23 февраля
д. месяц, когда есть праздники у друзей, но нет дней рождения у коллег
е. метки, к которым относится количество людей больше среднего
ж. месяц, в котором есть дни рождения у людей в совокупности, относящихся ко всем меткам

15. вузы для абитуриента: город, вуз, факультеты, направления, направленности, ЕГЭ которые нужно сдать, дата начала/конца приемной кампании.

(Направление -09.03.04 «Программная инженерия», Направленность — его конкретизация «Разработка программно-информационных систем», именно направленность закреплена за кафедрой и соответственно факультетом)

а. направленности, в которых есть слово «систем», но оно не первое
б. Кафедра, не принимающая ни на одну направленность
в. направление, на которое надо сдавать математику и информатику

г. факультет, принимающий на количество направлений больше среднего
д. город, в котором есть все укрупненные группы направлений и специальностей(УГСН) (первые 2 цифры номера специальности, т.е у 09.03.04 УГСН=09, а у 02.03.03-02)
е. вуз, с последним по алфавиту названием
ж. направление, на которое не надо сдавать ЕГЭ по математике, но надо по иностранному языку

16. школьные экскурсии: тип (развлекательная/образовательная), дисциплины к которым имеет отношение образовательная экскурсия, стоимость с человека, список участников, ответственный за проведение учитель
а. экскурсии, в названии которых есть слово музей, но оно не первое
б. экскурсии, относящиеся к истории и географии
в. места, куда нет экскурсий
г. учащиеся, которые не ездили на экскурсии в мае 2024, но ездили в музей истории религии
д. учитель, отвечавший за число экскурсий меньше среднего
е. учителя, отвечающие за самые деневые экскурсии
ж. места, в которые ездили все учащиеся

17. собачий питомник: собака, владелец, порода, родители, медали с выставок, дата рождения
а. породы, названия которых содержат часть слова «терьер», но не начинается с него
б. собаки без медалей
в. собака(ки), чьи щенки получили одинаковые медали
г. владелец, у которого есть собаки всех пород на букву «п»
д. владелец, у которого есть овчарки, но нет любых собак 2021 года рождения
е. собака, имеющая больше среднего щенков
ж. владельцы самых старых, но еще живых собак-родителей

18. служба доставки: адрес доставки, контактное лицо, стоимость посылки, диапазон желаемого времени доставки. время доставки фактическое, вес посылки, отметка о доставке, фирма отправитель
а. все посылки, отправляемые в район, в названии которой есть начало «моск», но это не единственные буквы в нем
б. Улица, на которой нет адресов доставки

в. контактное лицо, получавшее посылки на улице Строителей и на проспекте Ленина
г. фирма, отправившая больше среднего посылок
д. посылка с весом меньше среднего
е. контактное лицо, получавшее посылки от всех фирм на букву Б
ж. контактное лицо, никогда не получавшее посылок в прошлом году, но получавшее посылки в феврале текущего

19. туристический путеводитель: город, достопримечательность, адрес, тип достопримечательности (памятник, архитектурный комплекс, природный комплекс), дата создания
а. достопримечательности, в которых есть слово «дом», но на него название не заканчивается
б. город без улиц
в. улица, на которой есть и памятники, и фонтаны
г. город, в котором нет архитектурных комплексов, но есть музеи на улице Ленина
д. город, в котором музеев меньше среднего
е. улица с самыми новыми достопримечательностями
ж. тип достопримечательностей, который есть во всех городах, где есть достопримечательности

20. метрополитен: линия, станция, время закрытия, время открытия, адрес выходов, время проезда между станциями, дата открытия станции
а. станции, в названии которых есть слово «проспект», но оно на него не заканчивается
б. станция без выходов
в. станция, у которой есть выходы на 2 разные улицы
г. линия с временем проезда меньше среднего
д. линия, на которой собраны все стации, на букву «н»
е. станция с самым ранним открытием
ж. линия, на которой нет пересадочных станций на линию 1, но есть выходы на улицу Ленина

21. Медпункт организации: сотрудник организации, данные флюорографии (результат, дата), данные прививок, обращения в медпункте
а. все сотрудники, отчество которых оканчивается на «на», а начинается на И
б. тип заболевания, от которого не делали прививок

в. сотрудники организации, делавшие в прошлом году прививки от кори , а в текущем от дифтерии
г. сотрудник, последним сделавший флюорографию в текущем году
д. подразделение, сотрудник которого сделаны все типы прививок
е. сотрудник с количеством обращений в медпункт меньше среднего
ж. сотрудник, делавший флюорографию в этом году, но не делавший в прошлом

22. Инвентаризация: ответственное лицо, оборудование, категории имущества (компьютерная техника, канцелярские товары, мебель, освещение, спец. оборудование и т.д.), помещения, дата поставки, дата списания
а. любые помещения, где стоит специализированные столы (оборудование, содержащее слова «стол» , но не начинается с него.
б. помещение, в котором пока не числится никакого имущества
в. помещение, в котором есть оборудование категорий и компьютерная техника и другого типа
г. категория имущества , списанного последним в прошлом году
д. мат. ответственное лицо(лица), ответственное за оборудование всех категорий
е. мат. ответственное лицо(лица), ответственное за количество имущества больше среднего
ж. мат ответственное лицо, ответственное за мебель, но не оборудование из помещения 131

23. Календарь корпоративных мероприятий: время дата, событие, статус события для участника (обязательное, рекомендуемое, нейтральное), должности и подразделения участников, комнаты проведения
а. Все мероприятия, текущего года, в названии которых есть слово «семинар», но на него название не заканчивается
б. Комната, где не проходит мероприятий
в. Комната, в которой проходило 2 разных мероприятия в один день в мае текущего года
г. Участники самого первого мероприятия мая 2023 года
д. Событие, с самым количеством не пришедших, но приглашенных на него участников больше среднего. (участники на мероприятие могут быть приглашены и при этом могут прийти/не прийти).
е. Мероприятие, в котором обязательно присутствие всех сотрудников второго отдела
ж. Сотрудник, не посещавший мероприятий в июле прошлого года, но посещавший мероприятия в комнате 11-11.

24. Распределение обязанностей: обязанность, название, частота выполнения, категория обязанности (домашняя, оплата счетов, посещение мероприятий, общественная)
    а. Люди, выполняющие обязанности, имеющие в названии слово «мытье», но не начинающееся на него
    б. Категория обязанностей без обязанностей
    в. Человек, у которого есть обязанности и ежедневные и с другой частотой выполнения.
    г. Обязанность Иванова Ивана Петровича, которая выполняется последней по дате (дню)
    д. Частота выполнения обязанностей, обязанности с которой выполняют все, кто вообще имеет обязанности
    е. Категория, в которой обязанностей больше среднего
    ж. Человек, который выполняет обязанности из категории «ремонт», но не выполняет ежемесячных обязанностей

25. Багтрекинг: проекты, баги, тестировщики, разработчики (тестировщики не исправляют)
    а. Баги, в названии которых содержится слово «памят», но оно не первое
    б. Баги, найденные в одном проекте , одним тестировщиком в разные дни
    в. Разработчик без задач
    г. Критический баг, который добавили последним
    д. Тестировщик, с количеством добавленных багов меньше среднего
    е. Разработчик, у которого нет багов исправленных не в срок в прошлом году
    ж. Тестировщик, работавший со всеми проектами, название которых начинается со слова «система»

26. Домашняя аптечка. Группа лекарств, срок(дата) годности.
    а. Найти лекарство, которое содержит в названии текст «норм», но не в конце
    б. Найти лекарство, у которого есть аналоги по действующему веществу как в форме крема, так и в форме геля.
    в. Найти группу лекарств, в которой нет ни одного лекарства
    г. Найти лекарство с самым близким сроком годности, но не истекшим сроком годности.
    д. Найти действующее вещество, с количеством лекарств больше среднего.
    е. Найти группы лекарств, для которых есть лекарства во всех формах.
    ж. Найти группу лекарств, в которой нет таблеток, но есть лекарства в других формах с истекшим сроком годности.

27. спортзал: абонементы, виды спорта, тренеры, тип абонемента, инвентарь к виду спорта

а. виды спорта, на которые есть абонементы со словом «утренний» в названии типа, но это не первое слово названия
б. тренер, который ведет занятия по дзюдо и по ушу
в. вид спорта, по которому нет инвентаря
г. абонемент (предлагаемый) на румбу с минимальным сроком действия в абонементе
д. тренер, который ведет меньше среднего видов спорта
е. инвентарь, который нужен на фитнесе, но не нужен в боксе
ж. совокупный абонемент, на все виды бальных танцев.

28. Создайте базу данных фанфикшна (фанфикшн – это фанатские фантазии выраженные в дополнениях, переработках и продолжениях культовых произведений.) для хранения следующих сведений: оригинальное произведение (фандом), автор фанфика (произведения по мотивам), наименование фанфика, дата его добавление и последней редакции, наименование и текст глав фанфика.
а. перечень фанфиков, с названием на букву б, добавленных автором с id=1 в прошлом году;
б. фанфики, написанные более чем по одному произведению (кроссоверы) ,вроде «Чужой против хищника»);
в. фанфик, в котором пока ни одной главы
г. самый старый фандом (фандомы);
д. фандом, по которому написано наибольшее количество фанфиков;
е. авторов, пишущих только произведения по «Звездным войнам»;
ж. авторов, пишущих по всем фандомам;

29. Создайте базу данных идиом и крылатых выражений для хранения следующих сведений: отдельные слова идиомы, идиома, толкование, пример использования, дата добавления идиомы (если есть), пользователь добавивший идиому в словарь
а) перечень идиом, содержащих слово «труд», но не начинающихся с него добавленных в текущем году;
б) пары идиом с одинаковыми словами;
в) слово в базе данных, не встречающееся в идиомах
г) пользователь, добавивший больше всего идиом;
д) идиомы, с наибольшим количеством примеров использования;
е) пользователи, добавлявшие идиомы со словом «время», но не добавлявшие идиом в 2023;
ж) год, когда идиомы добавляли все пользователи;