**Лабораторная работа**

РЕЛЯЦИОННЫЕ БД. ПОСТРОЕНИЕ ER-ДИАГРАММЫ И БАЗОВЫЕ ЗАПРОСЫ К БД

**Цель работы:** получение практических навыков построения ER-диаграмм для реляционной БД, а также получение практических навыков работы с языком SQL.

**Инструменты:** draw.io – свободное ПР для создания различных диаграмм; консольное приложение с утилитой ssh для возможности подключения к удаленному серверу (например putty, mobaXterm, gitbash).

**1. Теоретический раздел**

**1.1 Реляционные базы данных**

Реляционные базы данных представляют собой базы данных, которые используются для хранения и предоставления доступа к взаимосвязанным элементам информации. Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка, содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют ключом. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных.

**Реляционная модель**

В первых базах данных данные каждого приложения хранились в отдельной уникальной структуре. Если разработчик хотел создать приложение для использования таких данных, он должен был хорошо знать конкретную структуру, чтобы найти необходимые данные. Такой метод организации был неэффективен, сложен в обслуживании и затруднял оптимизацию эффективности приложений. Реляционная модель была разработана, чтобы устранить потребность в использовании разнообразных структур данных.

Она обеспечила стандартный способ представления данных и отправки запросов, которые могли быть использованы в любых приложениях. Разработчики уяснили, что таблицы являются ключевым преимуществом реляционных баз данных, так как обеспечивают интуитивно понятный, эффективный и гибкий способ хранения структурированной информации и получения к ней доступа.

Со временем, когда разработчики стали использовать язык структурированных запросов (SQL) для записи данных в базу и отправки запросов, стало очевидным и другое преимущество реляционной модели. Вот уже на протяжении многих лет SQL широко используется в качестве языка запросов в базах данных. Он основан на алгоритмах реляционной алгебры и четкой математической структуре, что обеспечивает простоту и эффективность при оптимизации любых запросов к базе данных. Для сравнения: при использовании других подходов приходится создавать отдельные, уникальные запросы.

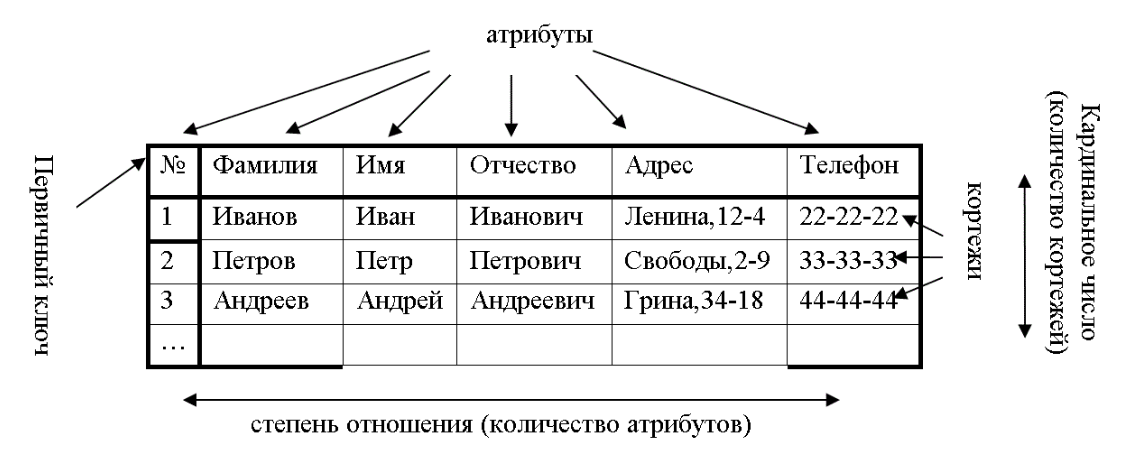


Рис. 1 Пример таблицы реляционной БД

**Ключи**

Когда речь идет о реляционной базе данных, ключи играют критическую роль в организации правильных и эффективных связей между таблицами. Основной задачей этого механизма является обеспечение целостных связей, которые помогают избежать аномалий и несогласованностей данных.

**Первичный ключ (PK – primary key)** – уникальный идентификатор записи в таблице. Он необходим для обеспечения уникальности каждой строки в конкретной таблице.

**Внешний ключ (FK – foreign key)** – ссылка на первичный ключ другой таблицы. Благодаря этому ходу обеспечивается взаимосвязь между двумя таблицами и поддерживается целостность данных.

**Связь** – благодаря внешним ключам записи в одной таблице могут ссылаться на записи другой таблицы, создавая структурированную и связную систему данных.

**Основные характеристики реляционных баз данных**

**Множество сущностей.** Объекты со строго определенным набором атрибутов, с помощью которых они связываются между собой, формируют понятную и простую для восприятия структуру.

**Табличный форма.** Такой формат гарантирует высокий уровень структурированности с жесткими логическими взаимосвязями, минимальный уровень избыточности данных, их согласованность и целостность.

**Язык SQL.** SQL является стандартизированным средством общения пользователя с базой данных. Он очень формальный, что делает его удобным и простым в изучении. SQL гарантирует точный результат даже при сложном многоуровневом запросе.

**Масштабирование по вертикали.** Реляционные базы данных хорошо масштабируются по вертикали. Но это значит, что по мере накопления информации в какой-то момент ее обработка потребует больших аппаратных ресурсов и финансовых затрат.

**Масштабирование по горизонтали**. Горизонтальное масштабирование, подразумевающее распределение таблиц данных по множеству серверов, является слабой стороной реляционных баз данных. С разрастанием системы появляются задержки в обновлении данных. В какие-то моменты нарушается принцип целостности данных, что может негативно отразиться на пользовательском опыте.

**Наличие требований к параметрам данных.** Реляционные базы данных умеют работать только со структурированными данными. Но современный цифровой мир полон неструктурированных данных (например, фото и видео), к которым нельзя применять принципы реляционной модели.

**Применение реляционных баз данных**

Реляционные базы данных — самые распространенные базы данных в мире. Компании самого разного размера и профиля используют их для обслуживания своих информационных систем. Такой подход удобен, чтобы:

* отслеживать торговые транзакции в интернете;
* обрабатывать критически важные данные банковских клиентов;
* хранить и обслуживать картотеки на промышленных предприятиях;
* вести учет в медицине и образовании.

То есть реляционные базы данных применимы везде, где важнейшим приоритетом является целостность и безопасность данных и где встречаются высокоструктурированные данные, соответствующие строгой, предсказуемой и предопределенной схеме.

**1.2 ER-диаграммы**

Схема «сущность-связь» (также ERD или ER-диаграмма) — это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. ER-диаграммы чаще всего применяются для проектирования и отладки реляционных баз данных в сфере образования, исследования и разработки программного обеспечения и информационных систем для бизнеса. ER-диаграммы (или ER-модели) полагаются на стандартный набор символов, включая прямоугольники, ромбы, овалы и соединительные линии, для отображения сущностей, их атрибутов и связей. Эти диаграммы устроены по тому же принципу, что и грамматические структуры: сущности выполняют роль существительных, а связи — глаголов.

В ER-моделях и моделях данных обычно выделяют до трех уровней детализации:

* *Концептуальная модель данных.* Схема наивысшего уровня с минимальным количеством подробностей. Достоинство этого подхода заключается в возможности отобразить общую структуру модели и всю архитектуру системы. Менее масштабные системы могут обойтись и без этой модели. В этом случае можно сразу переходить к логической модели.
* *Логическая модель данных*. Cодержит более подробную информацию, нежели концептуальная модель. На этом уровне определяются более подробные операционные и транзакционные сущности. Логическая модель не зависит от технологии, в которой она будет применяться. На этом уровне детализируют данные из концептуальной модели: к сущностям добавляют характеристики — атрибуты. Например, на логическом уровне описывают характеристики сущности «Транспорт»: марка и модель автомобиля, количество лошадиных сил, пробег, грузоподъёмность.
* *Физическая модель данных.* На основе каждой логической модели данных можно составить одну или две физических модели. В последних должно присутствовать достаточно технических подробностей для составления и внедрения самой базы данных. На этом уровне описывают, как будет организована работа с данными: выбирают тип базы, её содержание и где данные будут хранить. Например, выбирают реляционный тип базы данных и [СУБД](https://practicum.yandex.ru/blog/chto-takoe-subd/) для работы с ней, перечисляют таблицы в базе и определяют, что она будет храниться на внутреннем сервере компании.

В модели есть три типа связей:

* **«Один-к-одному»** — один экземпляр сущности связан только с одним экземпляром другой сущности. Например, пассажир рейса и его место в самолете.
* **«Один-ко-многим»** — один экземпляр сущности связан со множеством экземпляров другой сущности. Например, у одного пассажира может быть несколько единиц багажа, при этом каждая единица багажа может быть связана только с одним пассажиром.
* **«Многие-ко-многим»** — множество экземпляров одной сущности связаны со множеством экземпляров другой сущности. Например, аэропорт обслуживает несколько авиакомпаний. При этом каждая авиакомпания может обслуживаться в нескольких аэропортах.

ER-диаграммы — «родственники» схем структуры данных (DSD – data structure diagramm), где вместо связей между самими сущностями отображаются отношения между элементами внутри них. ER-диаграммы часто используются в сочетании с диаграммами DFD (data flow diagramm), которые схематично показывают движение потоков информации в рамках процесса или системы.

**Применение ER-диаграмм**

Модели «сущность-связь» традиционно используют для разработки программного обеспечения. При этом для метода нет конкретной области разработки: для создания любого ПО нужно работать с данными и транслировать их пользователям. Поэтому ER-модели строят и для интернет-магазина, и для корпоративного портала компании.   
  
Обычно ER-модель создают в двух случаях:

* когда перед началом проекта ещё не понятно, с какими данными предстоит работать;
* когда нужно создать новую базу данных или добавить таблицу в уже существующую.

Чем больше в системе сущностей и связей, тем важнее построить ER-модель до начала разработки ПО.  
  
На практике над простыми системами можно работать без концептуальной ER-модели. Например, программа для выдачи талонов электронной очереди — простая система, в которой всего две сущности — номер окна и номер очереди.

**Символы и нотации ERдиаграмм**

ER-модель — это общее представление данных, ER-диаграмма — представление модели, а нотация — графический язык для представления модели.  
  
Объясним на примере анатомии человека. Устройство человеческого организма — это модель. Её можно описать текстом, изобразить на картинке, перечислить все органы в таблице. Всё это разные представления одной и той же модели. Символы, с помощью которых описывают модель, — это нотации.  
  
Для того чтобы построить ER-диаграмму, можно использовать разные нотации. Три самые известные из них:  
  
**1. Нотация IDEF1X**. Её относят к фундаментальным, но на практике давно не используют, потому что есть более удобные варианты.  
  
**2. Нотация Чена**. Классическая нотация, которая состоит из простых символов — прямоугольников, овалов и линий. Из-за этого нотацию часто используют для концептуальных моделей, которые презентуют заказчику. Человеку, который далёк от аналитики данных, проще разобраться в понятных диаграммах со знакомыми символами.  
  
**3. Нотация Мартина**. Её ещё называют «воронья лапка» (от англ. Crow's Foot). Она компактнее нотации Чена, поэтому её используют для построения ER-моделей логического уровня, когда нужно описать в модели все атрибуты сущностей.  
  
В нотациях Чена и Мартина есть одинаковые элементы: сущности, атрибуты и связи. Но эти элементы диаграмм обозначают разными символами.

**Нотация Мартина**

Сущность изображается прямоугольником, внутри которого указано ее имя жирным шрифтом и список ее атрибутов (идентифицирующий атрибут подчеркнут/выделен жирным), а связь – линией, название которой располагается над ней и ее вид в месте соединения с сущностью определяет кардинальность связи (рис. 1)

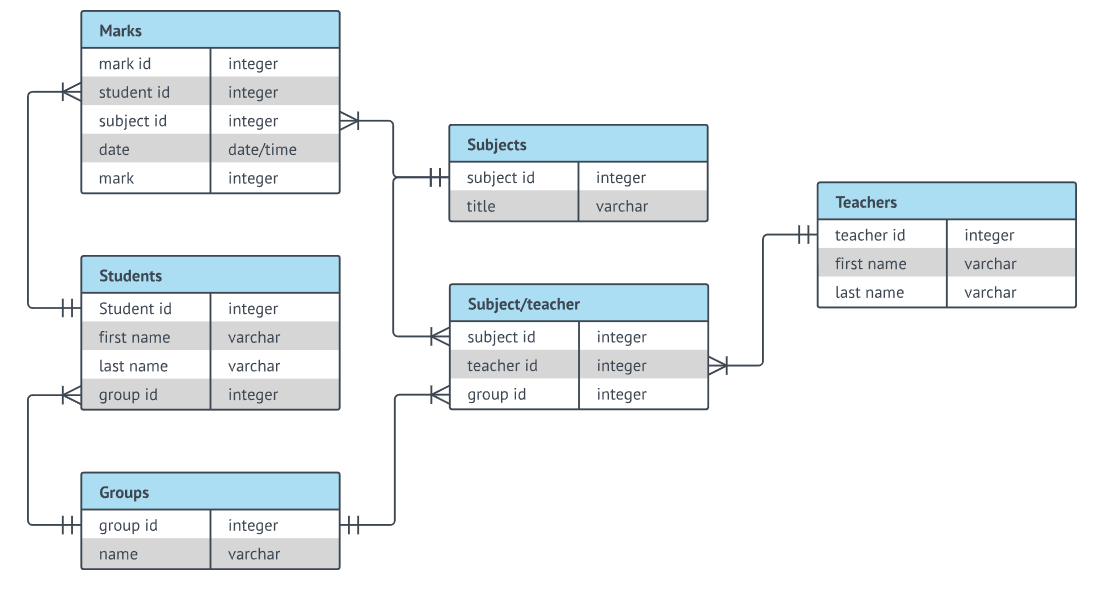
****

Рис. 1 Пример нотации Мартина

В нотации Мартина для обозначения связей между сущностями используются следующие элементы:

****

**Как создать простую ERдиаграмму**

Вот пошаговый алгоритм для создания простой ER-диаграммы «сущность-связь»:

1. Определить сущности. Чтобы собрать все сущности будущего проекта, системные аналитики общаются с заказчиком и будущими пользователями ПО: сотрудниками или клиентами компании. Например, если нужно разработать ПО для ветеринарной клиники, системный аналитик проведёт интервью с руководителем клиники, сотрудниками, врачами и клиентами, которые будут записываться на приём.  
   На этом этапе обычно создают концептуальную модель и согласовывают её с заказчиком.
2. Определить атрибуты. Системный аналитик детализирует информацию, собранную во время интервью, и описывает характеристики сущностей. Если данных не хватает, нужно повторно опросить заинтересованных лиц.
3. Определить связи между сущностями. На этом этапе выясняют, какие сущности связаны между собой. Например, пациенты и медицинская карта, филиал клиники и врачи, которые ведут приём.
4. Определить типы и характеристики связей. Например, пациенты и медицинская карта — это связь «один-к-одному», врач и день приёма — «один-ко-многим».   
     
   Затем ищут идентифицирующие связи между сущностями и определяют, какая из сущностей родительская. Допустим, у клиники есть филиалы — A, B и C. В каждом филиале есть кабинеты под номерами от 1 до 5. Это значит, что нельзя использовать номер кабинета без уточнения, в каком филиале он находится. Филиал — родительская сущность, а связь между филиалом и кабинетом — идентифицирующая.
5. Проверить ER-модель. После завершения работы над ER-моделью системный аналитик проверяет, нет ли в ней лишних сущностей, дубликатов данных и косвенных связей между данными в одной таблице. Такую проверку называют нормализацией данных.

**1.3 Язык SQL**

**SQL** (аббр. от англ. Structured Query Language — «язык структурированных запросов») — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

Является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. В общем случае SQL (без ряда современных расширений) считается языком программирования неполным по Тьюрингу, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

* создание в базе данных новой таблицы;
* добавление в таблицу новых записей;
* изменение записей;
* удаление записей;
* выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);
* изменение структур таблиц.

**Элементы**

Язык SQL представляет собой совокупность операторов, инструкций, вычисляемых функций.

Согласно общепринятому стилю программирования, операторы (и другие зарезервированные слова) в SQL обычно рекомендуется писать прописными буквами.

Операторы SQL делятся на:

операторы определения данных (Data Definition Language, DDL):

* CREATE создаёт объект базы данных (саму базу, таблицу, представление, пользователя и так далее),
* ALTER изменяет объект,
* DROP удаляет объект;

операторы манипуляции данными (Data Manipulation Language, DML):

* SELECT выбирает данные, удовлетворяющие заданным условиям,
* INSERT добавляет новые данные,
* UPDATE изменяет существующие данные,
* DELETE удаляет данные;

операторы определения доступа к данным (Data Control Language, DCL):

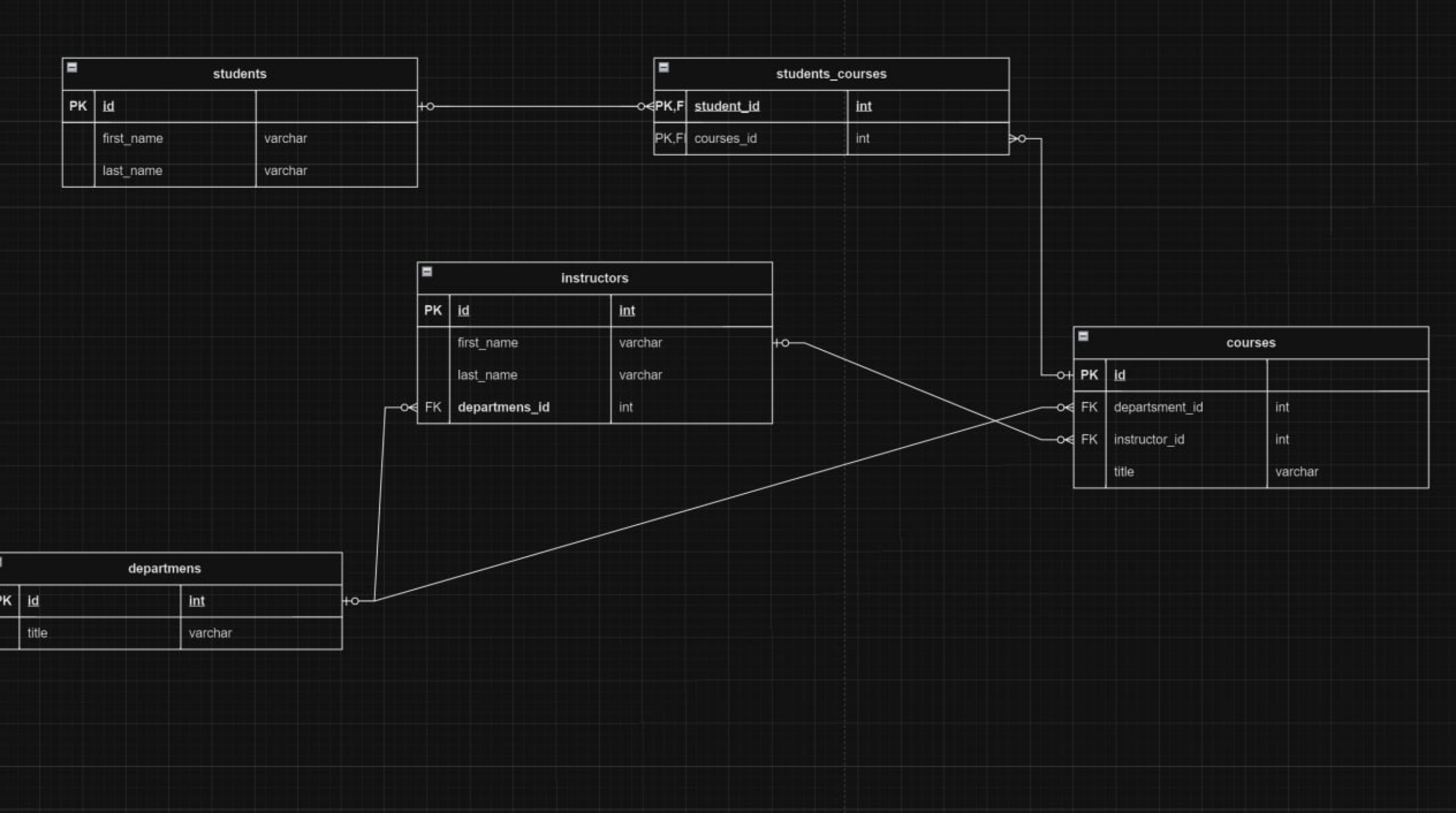
* GRANT предоставляет пользователю (группе) разрешения на определённые операции с объектом,
* REVOKE отзывает ранее выданные разрешения,
* DENY задаёт запрет, имеющий приоритет над разрешением;

операторы управления транзакциями (Transaction Control Language, TCL):

* COMMIT применяет транзакцию,
* ROLLBACK откатывает все изменения, сделанные в контексте текущей транзакции,
* SAVEPOINT делит транзакцию на более мелкие участки.

**2. Практические задания**

**2.1 Составить ER-диаграмму в нотации Мартина для следующей базы данных:**

В университете множество факультетов (department). Каждый факультет может предложить некоторое ненулевое количество предметов (course). В одном факультете может работать множество преподавателей (instructor), при этом каждый преподаватель может работать только на одном факультете. Каждый преподаватель может вести несколько курсов, либо не вести вообще. Курс может вести только один преподаватель. Студенты (student) могут выбрать любое количество предметов, при этом предмет может быть назначен разному количеству студентов.

**2.2 Работа с базой данных через с помощью SQL**

Для работы с базой данных подключитесь к удаленному серверу. Адрес сервера, данные учетной записи узнать у преподавателя.

Команда для подключения к серверу:

*ssh user@host*

При выполнении команды будет запрошен пароль. Введите его.

После входа на сервер необходимо подключиться к серверу MySql:

*mysql –u rudn\_student –p*

данная команда после выполнения запросит пароль для входа, введите его.

**Начало работы**

Перед началом работы необходимо выбрать базу данных. Для вывода списка доступных баз выполните команду:

*show databases;*

В результатах выполнения команды должна фигурировать БД с именем *rudn\_student\_db.* Выберите ее для дальнейшей работы*:*

*USE rudn\_student\_db;*

Следующим шагом просмотрите список таблиц, которые содержит в себе данная база, путем выполнения команды:

*SHOW TABLES;*

Результат выполнения команды занесите в раздел «Результаты выполнения команд» в «Результаты 1»

После того, как получите список таблиц, просмотрите, какие записи содержат в себе таблицы: departments, courses, instructors с помощью команды SELECT \* FROM <table\_name>, выполненной для каждой таблицы. Результаты занесите в «Результаты выполнения команд» в «Результаты 2»

**JOIN**

После просмотра содержимого таблиц можно заметить, что, например, таблица courses содержит идентификаторы преподавателей и факультетов. Данные идентификаторы нужны для использования в качестве внешних ключей для сохранения целостности данных в базе. Но для человека это не очень информативный вывод. Данные были бы более читаемыми, если вместо идентификаторов были выведены имена и фамилии преподавателей напротив предмета, который они ведут. Для этого нужно воспользоваться объединением таблиц с помощью оператора JOIN при выполнении команды SELECT:

*SELECT `instructors`.`first\_name`, `instructors`.`last\_name`, `courses`.`id` AS `course\_id`, `courses`.`title` FROM instructors JOIN courses ON `courses`.`instructor\_id` = `instructors`.`id`;*

Результаты занесите в «Результаты выполнения команд» в «Результаты 3»

**Вставка новых записей**

Теперь назначим студентам различные курсы, но перед этим посмотрим, есть ли в базе данных студенты и назначенные на них курсы через выполнение SELECT на таблицах students и students\_courses.

После того, как убедились в том, что нет студентов в таблице students, добавим их с помощью команды INSERT:

!!! Внимание!!! Вставьте свои имя и фамилию при выполнении команды

*INSERT INTO students (first\_name, last\_name) VALUES (‘<student\_name>’, ‘<student\_last\_name>’);*

После выполнения команды, выполните команду SELECT на таблице *students* и посмотрите список студентов.

Теперь необходимо назначить студентам курсы. Выберите из списка курсов 3 шт. и назначьте их на свои записи путем выполнения команды:

*INSERT INTO students\_courses (student\_id, course\_id) VALUES (… , …);*

Дождитесь, пока все выполняющие лабораторную работу выполнят команду и назначат себе курсы.

После того, как таблица *students\_courses* будет заполнена, выполните на ней SELECT и убедитесь, что в ней теперь много записей. Найдите записи только по себе, выполнив SELECT с дополнительной инструкцией WHERE, которая поможет отфильтровать только нужные значения:

*SELECT \* FROM students\_courses WHERE student\_id = <id>;*

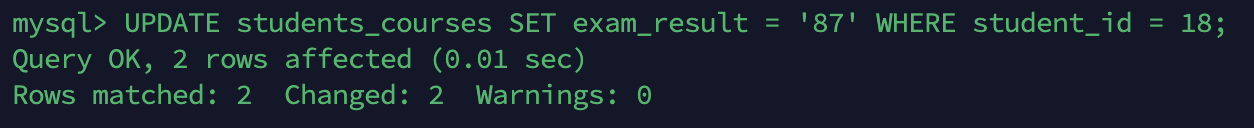
**Задача.** Выберите на свое усмотрение любого студента из students и посмотрите, какие курсы он себе назначил. Напишите запрос к БД таким образом, чтобы вывод был в формате, удобном для чтения человеку (без id). Подсказка: Используйте JOIN и WHERE.

**Задача.** Проверьте, есть ли курсы, которые не выбраны ни одним из студентов.

Подсказка: используйте подзапрос, инструкцию DISTINCT, а также IN в сочетании с WHERE.

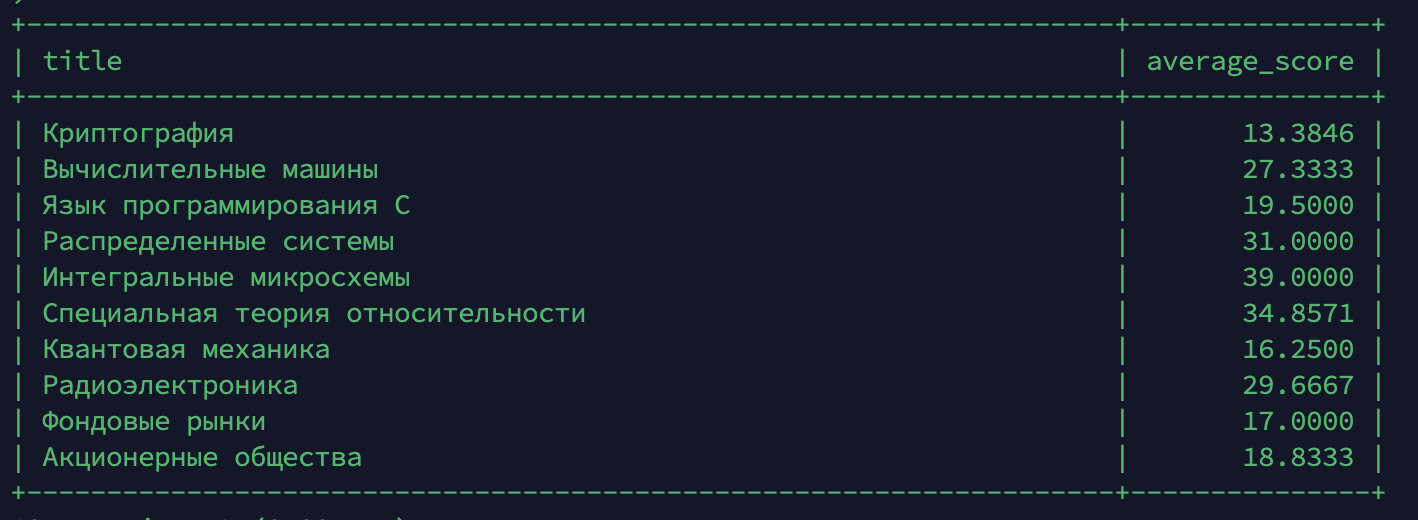
**Задача.** Напишите запрос, который покажет, какое количество студентов выбрало тот или иной курс.

Подсказка: используйте функцию COUNT() и инструкцию GROUP BY

\***Задача**. В таблице *students\_courses* для своих записей установите различные значения свойства *exam\_result.* Используйте инструкцию *UPDATE* для обновления строки. Не забывайте *WHERE* для выборки только своих записей.

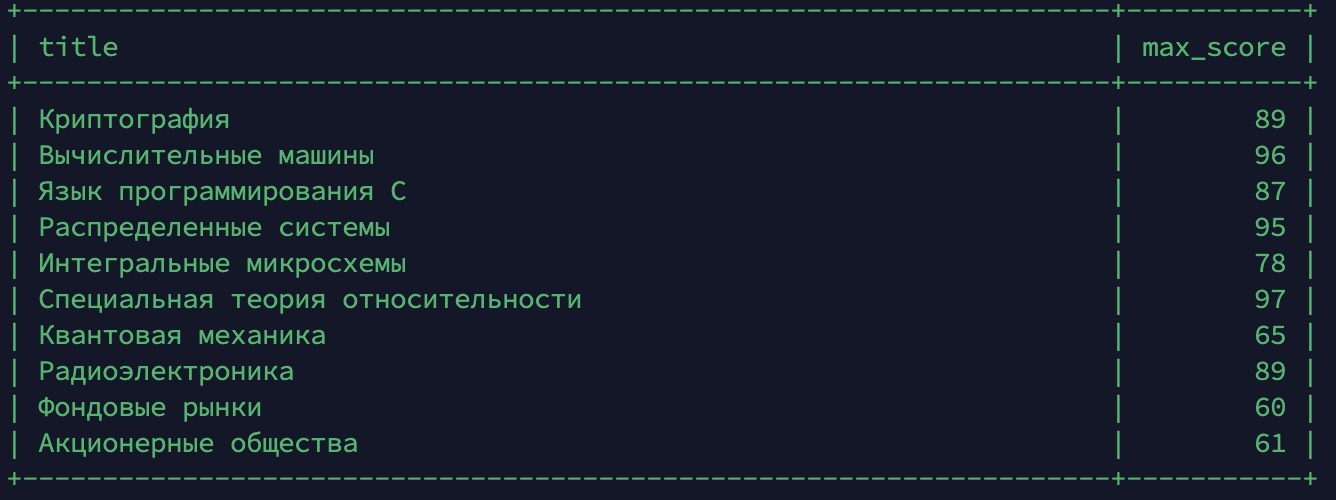
\***Задача.** Выведите в удобном формате среднее значение баллов за экзамен по каждому из предметов по всем студентам. Используйте функцию AVG(), JOIN, GROUP BY

```SELECT courses.title, AVG(students\_courses.exam\_result) AS average\_score FROM courses JOIN students\_courses ON courses.id = students\_courses.course\_id GROUP BY courses.title;```

Результат вывода:

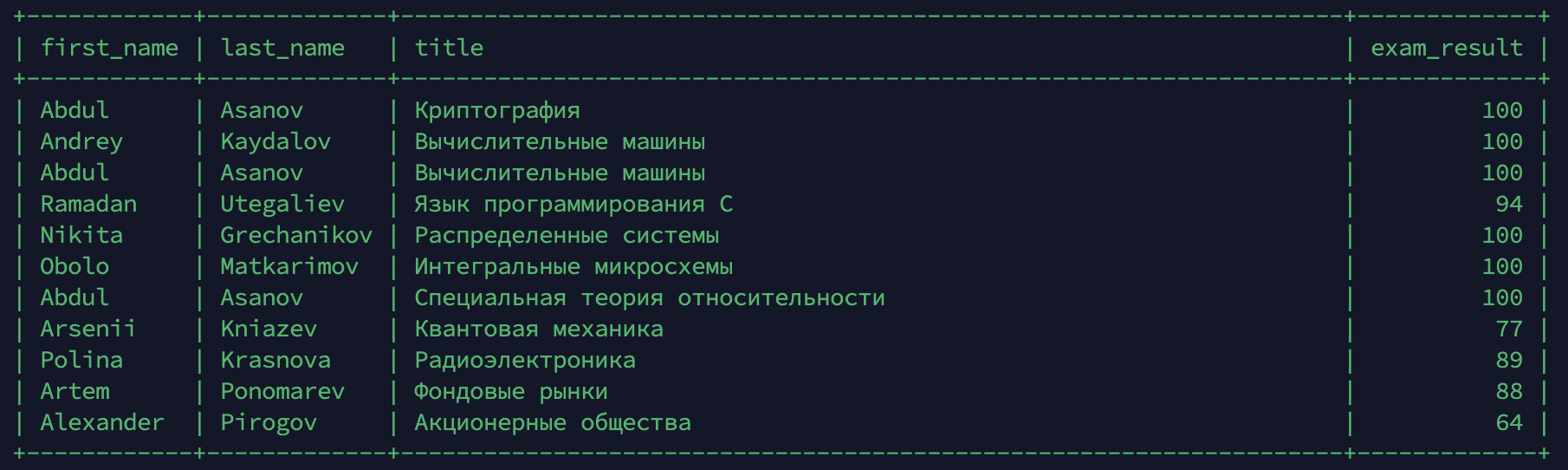
\***Задача.** Выведите в удобном формате максимальное значение балла за экзамен по каждому из предметов. Используйте функцию MAX(), JOIN, GROUP BY

```SELECT courses.title, MAX(students\_courses.exam\_result) AS max\_score FROM courses JOIN students\_courses ON courses.id = students\_courses.course\_id GROUP BY courses.title;```

Результат вывода:

\***Задача.** Выведите, кто из студентов набрал лучший балл по каждому из предметов.

```SELECT s.first\_name, s.last\_name, c.title, sc.exam\_result FROM students\_courses sc JOIN courses c ON sc.course\_id = c.id JOIN students s ON sc.student\_id = [s.id](http://s.id) WHERE (sc.course\_id, sc.exam\_result) IN (SELECT course\_id, MAX(exam\_result) FROM students\_courses GROUP BY course\_id);```

Результат вывода:

**Результаты выполнения команд:**

Результаты 1 (show tables)

Результаты 2 (select …)

Результаты 3 (join)