Министерство науки и высшего образования РФ

ФГБОУ ВО «Ульяновский государственный университет»

Факультет математики, информационных и авиационных технологий

Головин В.А.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ «ТЕХНОЛОГИИ ХРАНЕНИЯ И ОБРАБОТКИ БОЛЬШИХ ОБЪЕМОВ ДАННЫХ»

Ульяновск, 2019

Методические указания для самостоятельной работы студентов по дисциплине «Технологии хранения и обработки больших объемов данных» / составитель: В.А. Головин.- Ульяновск: УлГУ, 2019

Настоящие методические указания предназначены для студентов магистратуры по направлениям 01.04.02 «Прикладная математика и информатика» и 02.04.03 «Математическое обеспечение и администрирование информационных систем» всех форм обучения, изучающих дисциплину «Технологии хранения и обработки больших объемов данных». В работе приведены описание заданий лабораторной работы, комментарии и методические рекомендации по их выполнению, а также варианты лабораторной работы.

Студентам заочной формы обучения следует использовать данные методические указания при самостоятельном изучении дисциплины. Студентам очной формы обучения они будут полезны при подготовке к практическим занятиям и к экзамену по данной дисциплине. Рекомендованы к использованию ученым советом факультета математики, информационных и авиационных технологий УлГУ Протокол №2/19 от « 19» марта 2019 г.

1. ЛИТЕРАТУРА ДЛЯ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

1. Полякова Л. Н. Технологии ASP и ADO для организации доступа к базам данных : учеб.-метод. пособие / Л. Н. Полякова. - Ульяновск :УлГУ, 2004. ftp://10.2.96.134/Text/Polyakova.pdf.

2. Жуковский, О. И. Информационные технологии и анализ данных [Электронный ресурс] : учебное пособие / О. И. Жуковский. — Электрон. текстовые данные. — Томск : Томский государственный университет систем управления и радиоэлектроники, Эль Контент, 2014. — 130 c. — 978-5-4332-0158-3. — Режим доступа: http://www.iprbookshop.ru/72106.html

3. Воронова, Л. И. Machine Learning: регрессионные методы интеллектуального анализа данных [Электронный ресурс] : учебное пособие / Л. И. Воронова, В. И. Воронов. — Электрон. текстовые данные. — М. : Московский технический университет связи и информатики, 2018. — 82 c. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/81325.html

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ

2.1 Технологии big data: как анализируют большие данные

Большие данные мало просто собрать — их нужно как-то использовать, например, чтобы строить прогнозы развития бизнеса или проверять маркетинговые гипотезы. А для использования данные требуется структурировать и анализировать. Расскажем, какие существуют методы и технологии big data и как они помогают обрабатывать большие данные.

Краудсорсинг

Что это. Обычно анализом Big Data занимаются компьютеры, но иногда его поручают и людям. Для этих целей существует краудсорсинг — привлечение к решению какой-либо проблемы большой группы людей.

Как это работает. Предположим, у вас есть большой объем сырых данных. Например, записи о продажах магазинов, где товары часто записаны с ошибками и сокращениями. К примеру, дрель Dexter с аккумулятором на 10 мАч записана как «Дрель Декстр 10 мАч», «Дрель Dexter 10», «Дрель Dexter акк 10» и еще десятком других способов. Вы находите группу людей, которые готовы за деньги вручную просматривать таблицы и приводить такие наименования к одной форме.

Зачем и где применяют. Краудсорсинг хорош, если задача разовая и для ее решения нет смысла разрабатывать сложную систему искусственного интеллекта. Если анализировать большие данные нужно регулярно, система, основанная на Data Mining или машинном обучении, скорее всего, обойдется дешевле краудсорсинга. Кроме того, машины лучше справятся со сложным анализом, основанном на математических методах, например, со статистикой или имитационным моделированием.

Смешение и интеграция данных

Что это. Работа с big data часто связана со сбором разнородных данных из разных источников. Чтобы работать с этими данными, их нужно собрать воедино. Просто загрузить их в одну базу нельзя — разные источники могут выдавать данные в разных форматах и с разными параметрами. Тут и поможет смешение и интеграция данных — процесс приведения разнородной информации к единому виду.

Как это работает. Чтобы использовать данные из разных источников, используют следующие методы:

1. Приводят данные к единому формату: распознают текст с фотографий, конвертируют документы, переводят текст в цифры.
2. Дополняют данные. Если есть два источника данных об одном объекте, информацию от первого источника дополняют данными от второго, чтобы получить более полную картину.
3. Отсеивают избыточные данные: если какой-то источник собирает лишнюю информацию, недоступную для анализа, ее удаляют.

Зачем и где применяют. Смешение и интеграция данных нужны, если есть несколько разных источников данных, и нужно анализировать эти данные в комплексе.

Например, ваш магазин торгует офлайн, через маркетплейсы и просто через интернет. Чтобы получить полную информацию о продажах и спросе, надо собрать множество данных: кассовые чеки, товарные остатки на складе, интернет-заказы, заказы через маркетплейс и так далее. Все эти данные поступают из разных мест и обычно имеют разный формат. Чтобы работать с ними, их нужно привести к единому виду.

Традиционные методы интеграции данных в основном основаны на процессе ETL — извлечение, преобразование и загрузка. Данные получают из источников, очищают и загружают в хранилище. Специальные инструменты экосистемы больших данных от Hadoop до баз данных NoSQL также имеют собственный подход для извлечения, преобразования и загрузки данных. После интеграции большие данные подвергаются дальнейшим манипуляциям: анализу и так далее.

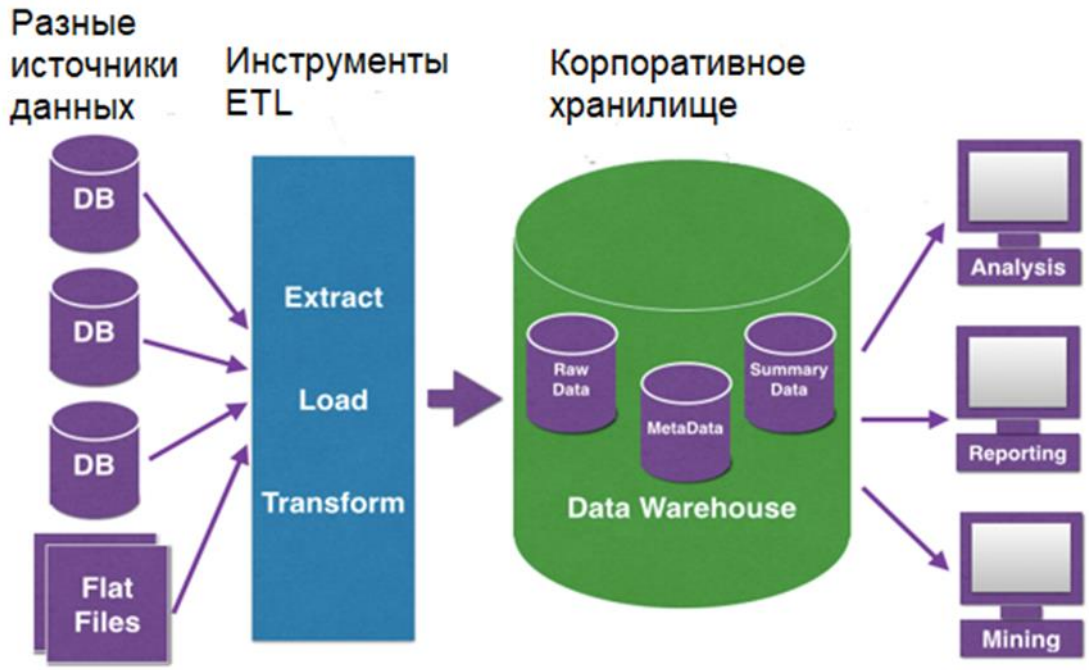


Схема выглядит примерно так: данные извлекают, очищают и обрабатывают, помещают в корпоративное хранилище данных, а потом забирают для анализа.

Машинное обучение и нейронные сети

Что это. Обычные компьютеры хорошо считают, но плохо справляются с некоторыми задачами, которые легко даются человеку. Например, вспомним пример выше: машине трудно понять, что «Дрель Декстр 10 мАч», «Дрель Dexter 10», «Дрель Dexter акк 10» — это одно и то же устройство. Чтобы машина могла мыслить как человек, требуется построить в ней структуру, похожую на человеческий мозг. Такими структурами и являются нейронные сети. Они состоят из множества искусственных нейронов, которые при обучении образуют связи и потом могут анализировать информацию.

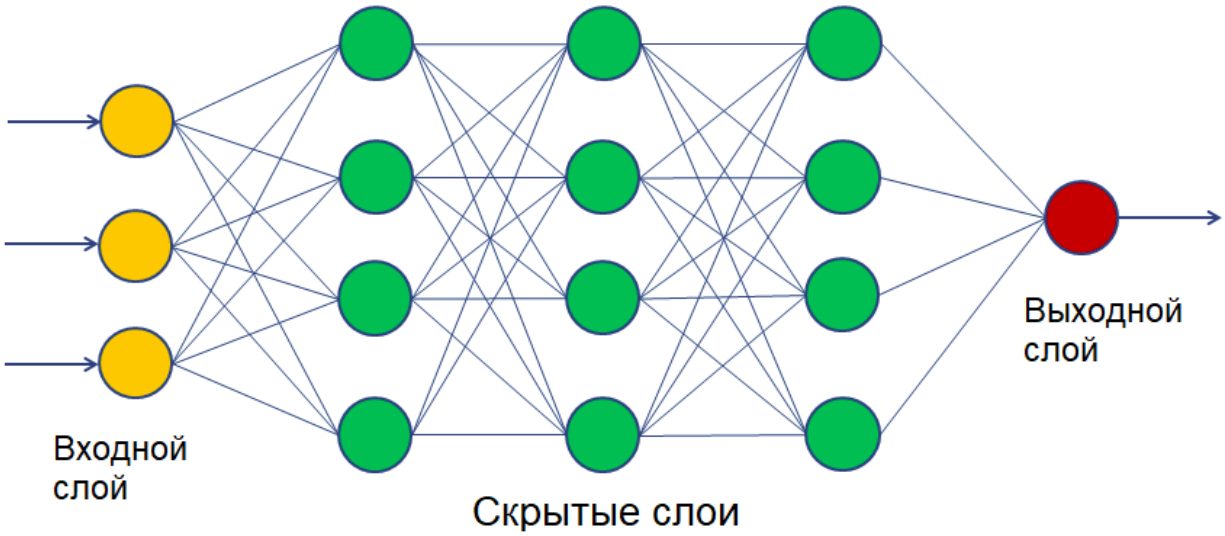
Как это работает. Нейронные сети работают по единому алгоритму — получают на входе данные, «прогоняют» их через сеть своих нейронов и на выходе выдают результат, например, относят входные данные к определенной группе.

Чтобы нейросеть работала, ее нужно сначала обучить — этот процесс называется машинным обучением.

Возьмем пример. Представим, что нужно научить нейросеть различать мужчин и женщин по фото. Для этого понадобится:

1. Построить нейросеть — запрограммировать искусственные нейроны воспринимать входные данные и создавать связи.
2. Передать нейросети очищенную выборку — базу лиц, однозначно отмеченных как женские или мужские. Так нейросеть поймет, по каким критериям отличать лица, то есть научится это делать.
3. Протестировать нейросеть — передать ей новую очищенную выборку, но не сообщать, какие лица мужские, а какие женские. Это поможет понять, как часто нейросеть ошибается, и приемлем ли для вас такой уровень ошибок.

После обучения и тестов можно использовать нейросеть для обработки big data.



Так выглядит простейшая нейросеть: информация подается на входной слой, обрабатывается внутри, а результат выдается через слой выхода.

Зачем и где применяют. Нейросети обычно используют, если нужно сортировать данные, классифицировать их и на основе входной информации принимать какие-то решения. Обычно нейросети используют для тех задач, с которыми справляется человек: распознать лицо, отсортировать фотографии, определить мошенническую банковскую операцию по ряду признаков. В таких задачах нейросеть заменяет десятки людей и позволяет быстрее принимать решения.

Предиктивная аналитика и big data

Что это. Часто нужно не просто анализировать и классифицировать старые данные, а делать на их основе прогнозы о будущем. Например, по продажам за прошлые 10 лет предположить, какими они будут в следующем году.

В таких прогнозах помогает предиктивная аналитика big data. Слово «предиктивный» образовано от английского «predict» — «предсказывать, прогнозировать», поэтому такую аналитику еще иногда называют прогнозной.

Как это работает. Задача предиктивной аналитики — выделить несколько параметров, которые влияют на данные. Например, мы хотим понять, продолжит ли крупный клиент сотрудничество с компанией.

Для этого изучаем базу прошлых клиентов и смотрим, какие «параметры» клиентов повлияли на их поведение. Это может быть объем покупок, дата последней сделки или даже неочевидные факторы вроде длительности общения с менеджерами. После этого с помощью математических функций или нейросетей строим модель, которая сможет определять вероятность отказа для каждого клиента и предупреждать об этом заранее.

Зачем и где применяют. Предиктивная аналитика нужна везде, где требуется строить прогнозы. Одними из первых ее начали использовать трейдеры, чтобы предсказывать колебания курсов на бирже. Сейчас такую аналитику используют в разных сферах, чтобы предсказывать:

- продажи и поведение клиентов в маркетинге;
- время доставки грузов в логистике;
- мошенничество в банковской и страховой сферах;
- рост компании и финансовые показатели в любых сферах.

На предприятиях и фабриках внедряют платформы индустриального интернета вещей: датчики собирают массивы данных о работе оборудования, а потом системы аналитики, в том числе на основе машинного обучения, обрабатывают их и предсказывают поломки и сроки технического обслуживания. Такие IoT-платформы можно развернуть в облаке: это снижает затраты на разработку, управление и эксплуатацию IoT-сервисов и решений.

Имитационное моделирование

Что это. Иногда возникает ситуация, в которой нужно посмотреть, как поведут себя одни показатели при изменении других. Например, как изменятся продажи, если повысить цену.

Ставить такие эксперименты в реальном мире неудобно — это дорого и может привести к серьезным убыткам. Поэтому чтобы не экспериментировать с реальным бизнесом, можно построить имитационную модель.

Как это работает. Представим, что мы хотим посмотреть, как разные факторы влияют на продажи магазина. Для этого берем данные: продажи, цены, количество клиентов и все остальное, имеющее отношение к магазину. На основе этих данных мы строим модель магазина. Потом вносим в нее изменения — повышаем и понижаем цены, меняем число продавцов, увеличиваем поток посетителей. Все эти изменения влияют на другие показатели — мы можем выбрать самые удачные нововведения и внедрить их в настоящем магазине.

Имитационное моделирование немного похоже на предиктивную аналитику. Только мы предсказываем будущее не по реальным, а по гипотетическим данным.

Имитационную модель можно построить и без big data. Но чем больше данных, тем точнее модель, так как она учитывает больше факторов.

Зачем и где применяют. Везде, где нужно проверять какие-нибудь гипотезы, но тестировать их на реальном бизнесе будет слишком дорого. Например, масштабное изменение цен на долгий срок может обрушить бизнес, так что перед таким шагом лучше провести тест на модели.

Важно помнить, что даже в масштабной модели часто бывают учтены не все факторы. Поэтому моделирование может дать неверный результат, переносить модель в реальность нужно с учетом всех рисков.

Статистический анализ

Что это. Суть статистики в том, чтобы собрать данные, посчитать их по определенным критериям и на выходе получить конкретный результат, обычно в процентах.

Одна из проблем статистики — недостоверные результаты на маленьких выборках. Например, из 20 000 человек 15 000 недовольны обслуживанием, но компания опросила только 100 — и в выборку попало 80 лояльных клиентов. Получится, что 80% опрошенных довольны обслуживанием, что не совпадает с реальностью.

Сделать статистику достовернее помогают большие данные. Чем больше информации вы собрали, тем точнее результат. Если вместо 100 клиентов опросить 10 000, результаты опроса уже можно считать достоверными.

Как это работает. Для получения точных статистических результатов используют разные методы. Вот некоторые из них:

1. Простой подсчет процентного соотношения.
2. Вычисление средних значений данных, иногда распределенных по группам.
3. Корреляционный анализ, который помогает выявить взаимосвязи и понять, как изменение одних данных повлияет на другие.
4. Метод динамических рядов, который оценивает интенсивность и частоту изменений данных с течением времени.

Зачем и где применяют. Везде, где для анализа данные нужно посчитать. Часто статистический анализ используют как часть других технологий — например, он необходим для имитационного моделирования или предиктивной аналитики.

Data mining

Что это. Big data — это большой массив разнородных данных. Чтобы они принесли пользу, в них нужно найти какие-то полезные закономерности: сходства, различия, общие категории и так далее. Процесс поиска таких закономерностей и называют data mining — добыча данных, или глубинный анализ данных.

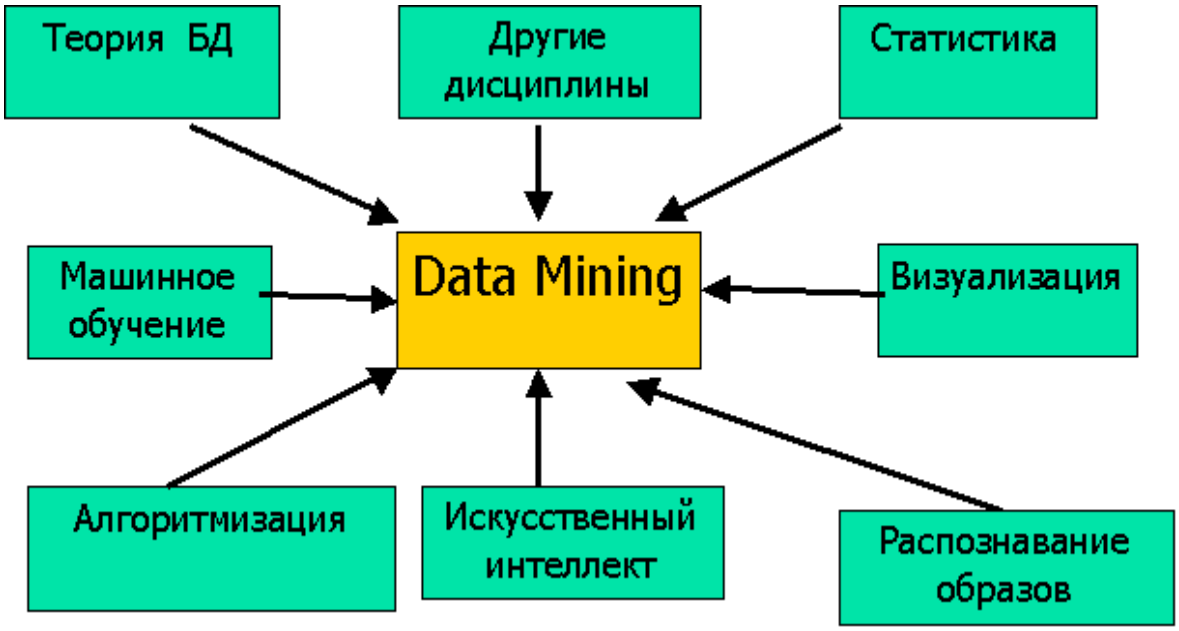
Как это работает. Мы берем большие данные и «добываем» из них новые полезные данные с помощью различных технологий: всевозможных методов классификации, моделирования и прогнозирования, основанные на применении деревьев решений, нейросетей, генетических алгоритмов и других методик. К методам data mining часто относят и статистические методы.

Data mining решает несколько основных задач:

1. Классификация — распределение данных по заранее известным классам.
2. Кластеризация — распределение данных на группы по степени похожести друг на друга. Например, составление разных портретов покупателей на основе их поведения в магазине.
3. Ассоциация — поиск повторяющихся образцов данных. Например, одинаковых наборов продуктов в чеках покупателей.

4. Регрессионный анализ — нахождение важных факторов, влияющих на какой-либо заданный параметр.
5. Анализ отклонений — выявление нетипичных данных, резко отличающихся от обычных.

Зачем и где применяют. Везде, где из больших данных нужно извлекать какие-то тенденции и закономерности. Решение большинства задач компании, связанных с данными, сводится к той или иной задаче data mining или их комбинации. Например, оценить риски можно с помощью регрессионного анализа, сегментировать покупателей с помощью кластеризации, предсказать спрос по выявлению ассоциаций в данных и так далее.



Data mining объединяет разные методики и технологии работы с данными.

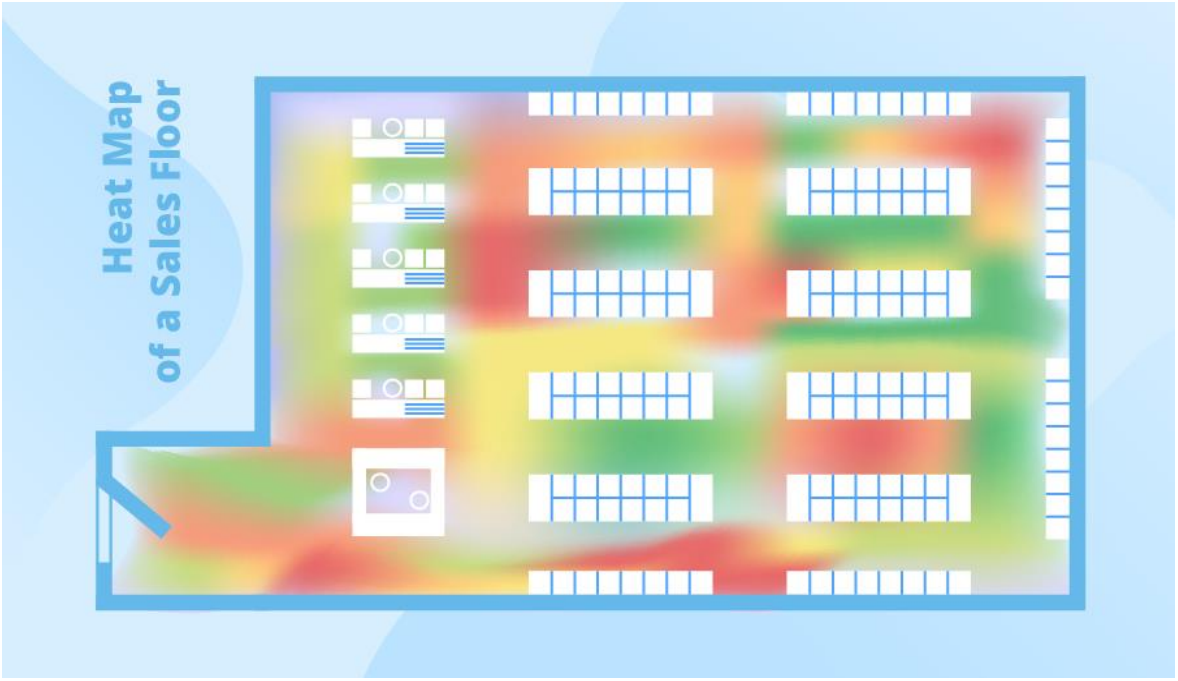
Визуализация аналитических данных

Что это. Чтобы результаты аналитики было удобнее оценивать и использовать, для работы с big data используют визуализацию данных. То есть представляют их в виде графиков, диаграмм, гистограмм, 3D-моделей, карт и пиктограмм.

Как это работает. Обычно визуализация — это конечный этап, демонстрация результатов анализа, проведенного другими способами. Например, вы построили имитационную модель и выводите результат ее работы в виде графика, который показывает колебание продаж в зависимости от изменений цены. Или сравнили продажи в разных регионах и визуализировали эти данные на карте, раскрасив регионы в разные цвета.

Обычно инструменты для анализа умеют и визуализировать данные, так как без визуализации результаты работы отобразить сложно. Для визуализации данных есть множество инструментов, например: Tableau, Qlik, Orange.

Зачем и где применяют. Везде, где с данными должны работать люди. Например, если нужно оценить результаты обработки или продемонстрировать их менеджеру или руководителю.



Пример визуализации поведения покупателей в магазине в виде тепловой карты.

Технологии для аналитики big data

1. Краудсорсинг — ручной анализ силами большого количества людей.
2. Смешение и интеграция данных — приведение данных из разных источников к одному виду, уточнение и дополнение данных.
3. Машинное обучение и нейронные сети — создание программ, которые умеют анализировать и принимать решения, выстраивая логические связи.
4. Предиктивная аналитика — предсказание будущего на основе собранных данных.
5. Имитационное моделирование — построение моделей на основе больших данных, которые помогают провести эксперимент в компьютерной реальности, без влияния на реальное положение вещей.
6. Статистический анализ — подсчет данных по формулам и выявление в них тенденций, сходств и закономерностей.
7. Data Mining — технология добычи новой значимой информации из большого объема данных.
8. Визуализация — представление больших данных и результатов их анализа в виде удобных графиков и схем, понятных человеку.

2.2 Структура учебной базы данных

Рассмотрим процесс построения учебной базы данных, которая ляжет в основу большинства примеров. Дадим еѐ вербальное описание.

> База данных автомастерская. В базе данных должны учитываться: дата, наименование, стоимость оказанной услуги, мастер, специализация мастера, владелец автомобиля, марка автомобиля, год выпуска автомобиля регистрационный номер автомобиля.

Для разработки структуры БД воспользуемся стандартной процедурой.

Определение типов сущностей

Перечислим все сущности, которые присутствуют в вербальном описании БД. Определим, являются ли эти сущности сильными или слабыми по следующему принципу:
Сильная сущность — независимая сущность.

Слабая сущность — зависимая сущность.
В результате для учебной базы данный получим:

- дата оказания услуги (слабая сущность)
- наименование услуги (слабая сущность)
- стоимость услуги (слабая сущность)
- мастер (**сильная сущность**)
- специализация мастера (слабая сущность)
- владелец автомобиля (**сильная сущность**)
- марка автомобиля (слабая сущность)
- регистрационный номер автомобиля (слабая сущность)
- год выпуска автомобиля (слабая сущность)

В связи с тем, что в вербальном описании часть сущностей могут присутствовать в неявном виде, проведём дополнительный анализ и выявим:

- услуга (**сильная сущность**)
- автомобиль (**сильная сущность**)
- журнал оказанных услуг (**сильная сущность**)

Даже если на данном этапе не будет выявлено ни одной неявной сущности, они будут выявлены позже (если таковые вообще имеются).

Важно отметить, что понятие «сильная» и «слабая» сущность всегда должны рассматриваться в некотором контексте. Скажем, если в вербальном определении речь идет о «владельце автомобиля», то «владелец» — это слабая сущность, так как это одно из свойств автомобиля; автомобиль в данном случае рассматривается как сильная сущность. Но если описание учебной базы данных дополнить фразой «у владельца автомобиля есть номер телефона», то в этом случае «владелец» является сильной сущностью, а «номер телефона» — слабой сущностью, то есть свойством «владельца автомобиля». В этом случае в рамках одной базы данных одна и та же сущность рассматривается и как сильная, и как слабая. Эта сущность должна быть отмечена как сильная.

Определение типов связей

Чаще всего рассматриваются только бинарные связи, то есть связи, которые объединяют пары сущностей. Обычно для построения связи используется следующий приѐм: все перечисленные выше сущности связываются между собой каким-нибудь глаголом. При этом не обязательно, что для каждой пары сущностей может быть образована связь.

Связь 1:1 (один к одному) возникает, когда одной родительской сущности соответствует одна дочерняя сущность, и, наоборот: одной дочерней сущности соответствует одна родительская сущность.

Связь 1:M (один ко многим) возникает, когда одной родительской сущности соответствует несколько дочерних сущностей, но не наоборот: одной дочерней сущности соответствует одна родительская сущность.

Связь M:M (много ко многим) возникает, когда одной родительской сущности соответствует несколько дочерних сущностей, и, наоборот: одной дочерней сущности соответствует несколько родительских сущностей. Выпишем связи, которые образуются в учебной базе данных.

- у услуги есть стоимость (1:M)
- у услуги есть название (1:1)
- услуга оказывается владельцу (M:M)
- у мастера есть ФИО (1:1)
- у мастера есть специализация (1:M)
- мастер оказывает услуги (M:M)
- у автомобиля есть цвет (1:M)
- у автомобиля есть марка (1:M)
- у автомобиля есть год выпуска (1:1)
- у автомобиля есть регистрационный номер (1:1)
- у владельца есть ФИО (1:1)
- у владельца есть адрес (1:1)
- у владельца есть номер телефона (1:1)
- у владельца есть дата рождения (1:1)

При проведении связей могут быть выявлены дополнительные сущности, которые могут сделать разрабатываемую БД более функциональной. Такими сущностями, например, являются: ФИО владельца, дата рождения владельца. Это позволит, скажем, поздравлять клиентов с днем рождения.

Поясним на конкретных примерах, почему в учебной базе данных проставлены те или иные типы связей.

- «у услуги есть название»: у услуги есть одно-единственное название, название соответствует одной-единственной услуге.
- «у автомобиля есть марка»: у автомобиля есть одна-единственная марка, но есть много автомобилей той же самой марки.
- «услуга оказывается владельцу»: владельцу оказывается много разных услуг, конкретная услуга может оказываться нескольким владельцам.

- «у автомобиля есть год выпуска». Рассмотрим эту связь подробнее. В принципе, здесь может быть проведена связь «один ко многим» по аналогии с маркой автомобиля, т.к. у автомобиля есть конкретный год выпуска, но есть много автомобилей с этим годом выпуска. Почему же в этом случае организуется связь «один к одному», а не «один ко многим? Дело в том, что марки автомобиля могут и должны быть вынесены в отдельную таблицу. В случае с годом выпуска в организации дополнительной таблицы нет смысла, поэтому используется связь один к одному.

Определение атрибутов и связывание их с типами сущностей

Атрибут — свойство сущности.

На одном из предыдущих этапов уже определено, какие из сущностей являются сильными, а какие слабыми. На основании этого распределения сущности должны образовать группы по следующему принципу — каждая слабая сущность становится атрибутом (свойством) сильной сущности. Из каждой сильной сущности формируется таблица, поля которой представляют собой атрибуты этой сильной сущности. Каждый атрибут может обладать дополнительными характеристиками.
Во-первых, необходимо выделить, какие из атрибутов являются простыми, а какие составными.

Атрибут называется простым (атомарным), если он не может быть разделён на составные части.

Составной атрибут может быть разделён на отдельные компоненты, которые в свою очередь могут быть рассмотрены как простые атрибуты.

Важно отметить, что составной атрибут является нежелательным при проектировании БД. Дело в том, что при заполнении значений составного атрибута достаточно сложно гарантировать, что пользователь будет строго следовать порядку перечисления компонент. Например, рассмотрим такой составной атрибут, как ФИО. Вполне очевидно, что сначала должна идти фамилия, затем имя и в последнюю очередь отчество. Однако практически невозможно отследить, что человек правильно внесёт эти данные. В результате нельзя будет, например, правильно отсортировать всех сотрудников по алфавиту. Невозможность контролирования порядка следования составных частей может привести к более серьезной проблеме: в базе данных рано или поздно появится дублирующаяся информация. Другими словами, может получиться несколько строк ссылающихся на один и тот же объект, например, «Иванов Иван Иванович» и «Иван Иванович Иванов». С точки зрения базы данных это два разных человека, а с точки зрения пользователя — один и тот же. Перечисленных проблем легко избежать, если разбить составной атрибут на необходимое количество простых.
Во-вторых, необходимо выделить потенциальные многозначные атрибуты.
Однозначный атрибут — атрибут, который содержит одно значение для одной сущности. Большинство атрибутов являются однозначными для каждого отдельного экземпляра этой сущности.
Многозначный атрибут — это атрибут, содержащий несколько значений для каждого экземпляра сущности. Многозначные атрибуты является нежелательными — по тем же причинам, что и составные. Ярким примером многозначного атрибута является атрибут «телефонный номер», т.к. номер может быть рабочий, домашний, сотовый. При этом не совсем понятно, каким образом все эти значения могут быть помещены в единственное поле. Самым простым способом устранения многозначного атрибута является разбивание многозначного атрибута на столько однозначных атрибутов, сколько потенциальных значений он может принять.

В-третьих, могут быть выделены производные атрибуты, т.е. атрибуты, которые могут быть получены из имеющихся по определённому алгоритму. Иногда, если алгоритм вычисления значения производного атрибута является достаточно долгим и дорогостоящим, целесообразно внести некоторую избыточности в структуру базы данных с целью повышения общей производительности.
Теперь определим типы используемых данных. Можно выделить следующие основные типы данных:

- строка (символ)
- битовый
- числа (целые, с фиксированной точкой, с плавающей точкой)
- деньги
- дата-время

Получим следующее разбиение на таблицы:

владелец

- ФИО (составной) [строка]
- пол (простой) [строка]
- адрес (составной) (многозначный) [строка]
- дата рождения (составной) [дата]
- телефон (составной) (многозначный) [строка]

услуга

- название (простой) [строка]
- цена (простой) [деньги]

специальность

- название (простой) [строка]

журнал оказанных услуг

- дата оказания услуги (составной) [дата-время]
- время, на оказание услуги (составной) [дата-время]

автомобиль

- регистрационный номер (составной) [строка]
- год выпуска (простой) [дата]
- цвет автомобиля (простой) [строка]
- марка автомобиля (простой) [строка]

марка автомобиля

- название (простой) [строка]

мастер

- ФИО (составной) [строка]
- специальность (простой) [строка]

Определение доменов атрибутов

Домен — это набор допустимых значений для одного или нескольких атрибутов. Другими словами, домен определяет все потенциальные значения, которые могут быть присвоены атрибуту. Одна из главных причин выделения доменов состоит в том, что домены помогают обеспечивать логическую целостность базы данных.
Приведём небольшой пример. Пол человека может быть записан или как «муж»/«жен», или как «м»/«ж». Для того, чтобы обеспечить однозначность написания, определяется домен. При попытке внести значение, которое не включено в домен, будет выдаваться ошибка о невозможности внесения данных.
Для учебной базы данных доменами могут быть выбраны:

- пол владельца
- дата рождения владельца
- год выпуска автомобиля
- дата оказания услуги
- время, затраченное на оказание услуги

Определение первичных ключей

Следующий важный момент при проектировании базы данных состоит в определении потенциальных и первичных ключей.

Потенциальным ключом называется атрибут или набор атрибутов, которые уникальным образом идентифицируют отдельные экземпляры типа сущности.

Потенциальный ключ обладает двумя свойствами:

- уникальность: ключ единственным образом идентифицирует экземпляр сущности;
- неприводимость: никакое допустимое подмножество ключа не обладает свойством уникальности.

Примером потенциального ключа могут служить табельный номер или номер паспорта.
Первичным ключом называется некоторый выбранный потенциальный ключ сущности.

При выборе первичного ключа среди нескольких потенциальных можно руководствоваться следующими рекомендациями:

- используйте потенциальный ключ с минимальным набором атрибутов;
- используйте тот потенциальный ключ, который имеет минимальную вероятность потери уникальности значений в будущем;
- по возможности используйте числовой потенциальный ключ;
- используйте потенциальный ключ, значения которого имеют минимальную длину (в случае текстовых атрибутов);

- остановите свой выбор на потенциальном ключе, с которым будет проще всего работать (с точки зрения пользователя).

Если сильная сущность не содержит ни одного подходящего потенциального ключа, то потенциальный ключ всегда может быть искусственным образом добавлен в виде некоторого уникального числового кода.
Перечислим первичные ключи из рассматриваемого примера.

- Для владельца это **код_владельца**
- Для автомобиля это **регистрационный_номер**
- Для услуги это **код_услуги**
- Для специальности это **код_специальности**
- Для журнала оказанных услуг это **номер_чека**
- Для марки автомобиля это **код_марки**
- Для мастера это **табельный_номер**

Определение внешних ключей

Внешний ключ — это атрибут или множество атрибутов внутри сущности, которое соответствует потенциальному ключу некоторой (может быть, той же самой) сущности.

Внешние ключи позволяют определять логическое связывание таблиц. Суть связывания состоит в установлении соответствия полей связи родительской и дочерней таблиц.
Перечислим внешние ключи с указанием таблиц, на которые они ссылаются.

В таблице **автомобили** три внешних ключа:

- **код_владельца** (таблица **владельцы**)
- **код_марки** (таблица **марки автомобилей**)
- **код_цвета** (таблица **цвета**)

В таблице **журнал_оказанных_услуг** тоже три внешних ключа:

- **регистрационный_номер** (таблица **автомобили**)
- **табельный_номер** (таблица **мастера**)
- **код_услуги** (таблица **услуги**)

И, наконец, в таблице **мастера** есть один внешний ключ **код_специальности**, ссылающийся на таблицу **специальности**.

Для удобства обычно принимают следующую схему именования таблиц и их столбцов: таблицы в своём название чаще всего содержат имена существительные во множественном числе, а столбцы — имена существительные в единственном числе.

Создание диаграммы «сущность-связь»

При создании диаграммы «сущность-связь» чаще всего используется какая-либо CASEсреда.

CASE-технология (Computer-AidedSoftware/SystemEngineering) представляет собой совокупность методологий анализа, проектирования, разработки и сопровождения сложных систем и поддерживается комплексом взаимосвязанных средств автоматизации.
CASE-технология обычно определяется как методология проектирования информационных систем плюс инструментальные средства, позволяющие наглядно моделировать предметную область, анализировать ее модель на всех этапах разработки и сопровождения информационной системы и разрабатывать приложения для пользователей.
CASE-технологии обеспечивают всех участников проекта, включая заказчиков, единым, строгим, наглядным и интуитивно понятным графическим языком, позволяющим получать обозримые компоненты с простой и ясной структурой.

При этом проектируемые объекты (программы или структура базы данных) представляются двумерными схемами (которые проще в использовании, чем многостраничные описания), позволяющими заказчику участвовать в процессе разработки, а разработчикам — общаться с экспертами предметной области, разделять деятельность системных аналитиков, проектировщиков и программистов, обеспечивая легкость сопровождения и внесения изменений в систему.
На рисунке приведена диаграмма, разработанная средствами MS Visio.

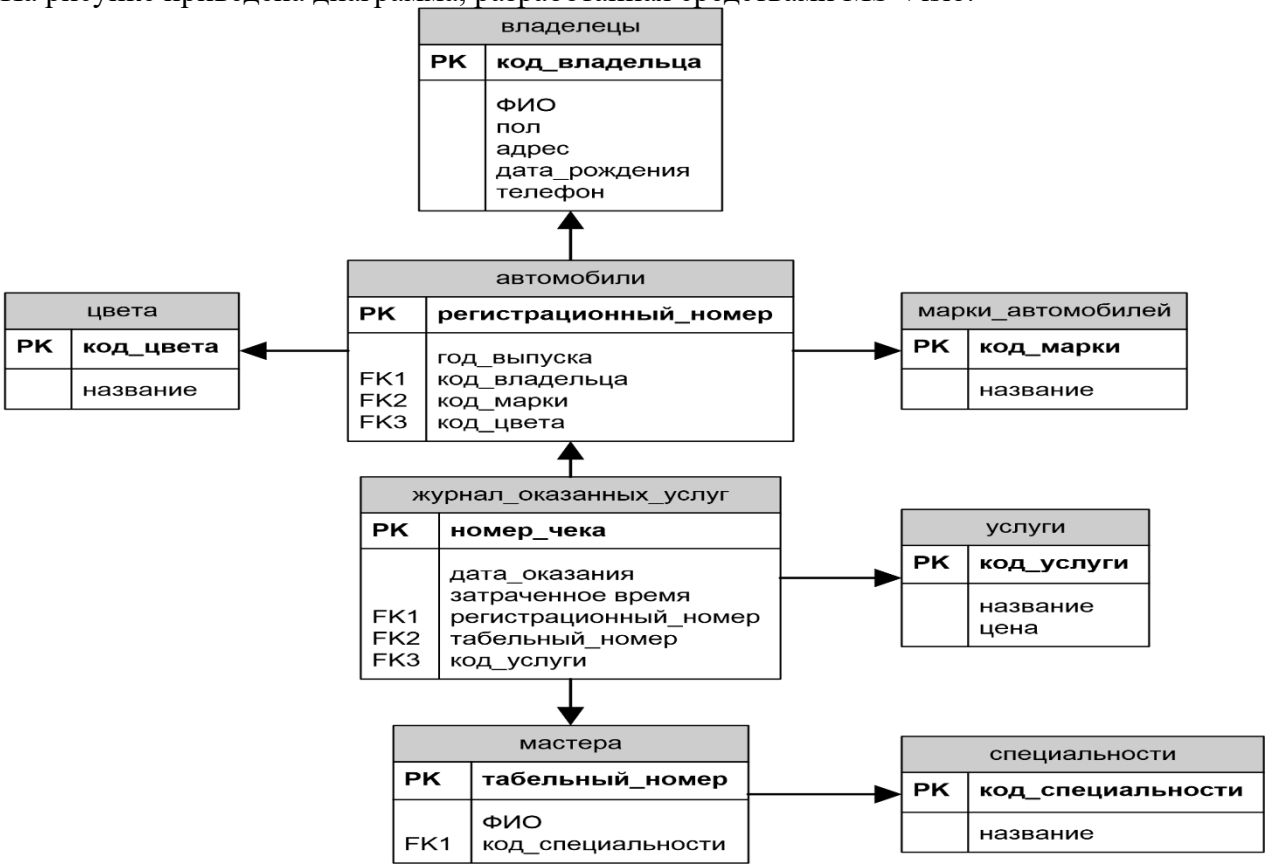


Рисунок 1. Диаграмма "сущность-связь" учебной базы данных

Создание таблиц

Прежде чем приступить к заполнению таблиц, необходимо их создать. Сделать это можно одним из двух способов.
Первый способ заключается в использовании визуальных инструментов, которые предлагаются некоторыми средами разработки баз данных.

Второй способ заключается в самостоятельном написании запросов по созданию таблиц. Остановимся на этом подробнее.

Один из возможных синтаксисов запроса на создание таблицы выглядит следующим образом:

```
createtable<имя таблицы>
(
<имя столбца 1><тип данных><ограничения>,   <имя столбца
2><тип данных><ограничения>,   ...,
<имя столбца n><тип данных><ограничения>,
<дополнительное ограничение 1>,   <дополнительное ограничение
2>,
  ...,
<дополнительное ограничение n> )
```

Перечислим основные ограничения, которые могут использоваться при создании таблиц:

- обязательное для заполнения поле (**notnull**);
- поле, значение которого должно удовлетворять условию (**check**);
- уникальное поле, значение которого не может повторяться (**unique**);
- ограничение первичного ключа (**CONSTRAINT <имя ограничения> PRIMARY KEY (<название столбца, на который накладывается ограничение>)**);
- ограничение внешнего ключа (**CONSTRAINT <имя ограничения> FOREIGN KEY (<название столбца, на который накладывается ограничение>) REFERENCES <имя таблицы на которую ссылается столбец>**).

Также перечислим основные типы данных, которые чаще всего используются при создании таблиц:

- целое число (**int**)
- вещественное число (**float**)
- целое число с фиксированным количеством знаков после запятой (**decimal**)
- деньги (**money**)
- дата и время (**datetime**)
- строка фиксированной длины (**char**)
- строка переменной длины (**varchar**)

Приведём примеры использования этого синтаксиса при создании учебной базы данных. Прежде чем начать, следует обратить внимание на правильный порядок создания таблиц. Например, нельзя создавать таблицу с внешними ключами до того, как будут созданы таблицы, на которые они ссылаются. Запишем запросы на создание таблиц учебной базы данных в нужном порядке.

Таблица**владельцы**:

```
create table clients
(
idint not null,   fio varchar(64) not null,
```

sex char(1) not null check (sex in ('м', 'ж')), address varchar(64),
phone varchar(16), birth date,
CONSTRAINT clients_pk PRIMARY KEY (id))

Таблица**цвета**:

```
create table colors
(
idint not null,
title varchar(64) unique not null,
 CONSTRAINT colors_pk PRIMARY KEY (id) )
```

Таблица **марки_автомобилей**:

```
createtablebrands
(
idint not null,
title varchar(64) unique not null,
 CONSTRAINT brands_pk PRIMARY KEY (id) )
```

Таблица**услуги**:

```
create table services
(
idint not null,
title varchar(64) unique not null, price decimal(10, 2) not null check
(price>0),
 CONSTRAINT services_pk PRIMARY KEY (id)
)
```

Таблица**специальности**:

```
create table professions
(
idint not null,
title varchar(64) unique not null,
 CONSTRAINT professions_pk PRIMARY KEY (id)
)
```

Таблица**мастера**:

```
create table masters
(
idint not null, fio varchar(64) not null,
profession_idint not null,
```

```
  CONSTRAINT masters_pk PRIMARY KEY (id),

  CONSTRAINT profession_fk FOREIGN KEY
    (profession_id) REFERENCES professions
)
```

Таблица **автомобили**:

```
create table cars
(
 id varchar(12) not null,   p_date date not null,
client_idint not null,   color_idint not null,
brand_idint not null,
  CONSTRAINT cars_pk PRIMARY KEY (id),

  CONSTRAINT client_fk FOREIGN KEY
    (client_id) REFERENCES clients,
  CONSTRAINT color_fk FOREIGN KEY
    (color_id) REFERENCES colors,
  CONSTRAINT brand_fk FOREIGN KEY
    (brand_id) REFERENCES brands
)
```

Таблица **журнал_оказанных_услуг**:

```
createtablecashbook
(   id int not null,   s_date date not null,   s_timeint not null
check (s_time>0),   car_id varchar(12) not null,   service_idint
not null,   master_idint not null,   CONSTRAINT
cashbook_pk PRIMARY KEY (id),

  CONSTRAINT car_fk FOREIGN KEY
    (car_id) REFERENCES cars,
  CONSTRAINT service_fk FOREIGN KEY
    (service_id) REFERENCES services,
  CONSTRAINT master_fk FOREIGN KEY     (master_id)
REFERENCES masters
)
```

Наполнение таблиц данными

Для наполнения таблиц данными может использоваться либо визуальный интерфейс, предоставляемый средой разработки базы данных, либо специальные операторы языка SQL.

Добавление данных

Общий синтаксис оператора добавления данных в таблицу:

```
INSERT INTO <имя таблицы>
(
<имя столбца 1>,   <имя
столбца 2>,
  ...,
<имя столбца n>
)
VALUES
(
<значение 1>,
<значение 2>,
  ...,
<значение n>
)
```

Модификация данных

Общий синтаксис оператора модификации данных в таблице:

```
UPDATE <имя таблицы> SET
<имя столбца 1> = <выражение 1>, <имя столбца
2> = <выражение 2>,
...,
<имя столбца n> = <выражение n>
WHERE <условие отбора модифицируемых записей>
```

Удаление данных

Общий синтаксис оператора удаления данных из таблицы:

```
DELETE FROM <имя таблицы>
WHERE <условие отбора удаляемых записей>
```

Построение запросов

Общий синтаксис построения запросов таков:

```
SELECT [ALL | DISTINCT]
{* | [имя_столбца [AS новое_имя]] [,...n]}
```

FROM имя_таблицы [AS] [псевдоним] [,...n]

[WHERE <критерии поиска>]

[GROUP BY имя_столбца [,...n]]

[HAVING <критерии выбора групп>]

[ORDER BY имя_столбца [,...n]]

Ключевое слово DISTINCT необходимо указывать в том случае, если требуется удалить все дублирующиеся строки из результата выполнения запроса.
Ключевое слово ALL используется по умолчанию и применяется в том случае, если нужны все строки, которые будут получены в результате выполнения запроса.
После ключевого слова SELECTидёт перечень столбцов, которые должны фигурировать в результате выполнения запроса. Каждый столбец может получить псевдоним (для чего необходимы псевдонимы, будет объяснено ниже). Если требуется получить полный перечень столбцов из таблиц, которые используются в запросе, указывается специальная мнемоника «*****», на место которой в момент выполнения запроса будет подставлен полный перечень столбцов.
После ключевого слова FROMидёт перечень таблиц, которые участвуют в запросе. Каждая таблица тоже может получить псевдоним.
После ключевого слова WHERE записывается перечень условий, согласно которым будет производиться отбор строк в результате выполнения запроса.
Результат выполнения запроса может быть сгруппирован; условия группировки указываются после ключевого слова GROUP BY.

Аналогично ключевому слову **WHERE**, ключевое слово **HAVING** используется для указывания дополнительных условий на данные, которые должны получиться в результате выполнения запроса. Однако, в отличие от **WHERE**, после ключевого слова HAVING указываются условия на отбор групп данных.
После ключевого слова ORDER BY указывается перечень столбцов, по которым будет осуществляться сортировка, а также принцип организации этой сортировки. Для сортировки в прямом порядке используется ключевое слово **ASC**, а для сортировки в обратном порядке — ключевое слово DESC.

Приведём примеры самых простых запросов.
Пусть мы хотим получить полный перечень всех клиентов, которые пользуются услугами автомастерской. Сделать это можно при помощи следующего запроса:

Пример 1. Получить полный перечень клиентов, которые пользуются услугами автомастерской.

```
SELECT * FROM clients
```

Усложним задание и напишем запрос, который должен выводить ФИО и адрес клиента. Для этого в запросе нужно указать те столбцы, которые нам необходимы.

Пример 2. Получить список фамилий и адресов клиентов, которые пользуются услугами автомастерской.

```
SELECT fio, address FROM clients
```

Теперь продемонстрируем пример использования ключевых слов DISTINCT и ALL. Когда ни одно из ключевых слов не указано, по умолчанию используется ALL. Если мы захотим написать запрос, который будет выводить перечень городов, в которых живут клиенты автомастерской, то следующий запрос несколько раз выведет «Ульяновск»:

Пример 3. Получить перечень городов, в которых живут клиенты, пользующиеся услугами автомастерской (неправильная реализация).

```
SELECT ALL address FROM clients
```

В этом случае мы дали команду на выбор всех строк из таблицы **clients** из столбца **address**. Если же нужен перечень городов без повторений, то следует использовать ключевое слово **DISTINCT**.

Пример 4. Получить перечень городов, в которых живут клиенты, пользующиеся услугами автомастерской (правильная реализация).

```
SELECT DISTINCT address FROM clients
```

При выполнении этого запроса будут получены только уникальные записи, что и требовалось в вербальной формулировке запроса.

Запросы с условиями

С помощью ключевого слова **WHERE** можно определять, какие строки данных из приведенных в списке **FROM** таблиц появятся в результате запроса. Рассмотрим пять основных типов условий поиска.

Сравнение

В языке SQL можно использовать следующие операторы сравнения:

=	Равенство
<	Меньше
>	Больше
<=	меньше или равно
>=	больше или равно
<>	не равно

Если нужно более сложное условие, оно может быть составлено при помощи логических операторов **AND**, **OR**, **NOT** и круглых скобок, определяющих очерèдность выполнения действий.

Пример 5. Вывести список услуг, цена на которые не превосходит 1000 рублей.

```
SELECT title FROM services where price<=1000
```

Пример 6. Вывести список услуг цена, на которые больше или равна 2000 и меньше или равна 10000.

```
SELECT title, price  FROM
services
where price>=2000 AND price<=10000
```

Пример 7. Вывести список клиентов, проживающих в Ульяновске или Саратове.

```
SELECT fio FROM clients where
address='Ульяновск' OR address='Саратов'
```

Диапазон

Оператор **BETWEEN** используется для поиска значения внутри некоторого интервала, задаваемого своими минимальным и максимальным значениями, при этом граничные значения включаются в диапазон поиска.

Пример 8. Вывести список услуг, цена на которые больше или равна 2000 и меньше или равна 10000.

```
SELECT title, price  FROM
services
where price BETWEEN 2000 AND 10000
```

Пример 9. Вывести список услуг, цена на которые не лежит в диапазоне от 2000 до 10000.

```
SELECT title, price  FROM
services
where price NOT BETWEEN 2000 AND 10000
```

Принадлежность множеству

Оператор **IN** используется для проверки того, равняется ли значение в ячейке таблицы одному из значений в предоставленном списке. При помощи оператора **IN** может быть достигнут тот же результат, что и в случае применения оператора **OR**, однако оператор **IN** выполняется быстрее.

Пример 10. Вывести список клиентов, проживающих в Ульяновске или Саратове.

```
SELECT fio FROM clients
where address IN ('Ульяновск', 'Саратов')
```

Оператор **IN** очень удобен в запросах с отрицательным условием:

Пример 11. Вывести список клиентов, проживающих не в Ульяновске и не в Саратове.

```
SELECT fio FROM clients
where address NOT IN ('Ульяновск', 'Саратов')
```

Список вариантов может быть не только статическим, т.е. жѐстко заданным на момент составления запроса, но и динамическим, т.е. формироваться при помощи подзапроса в момент выполнения запроса (см. раздел «Подзапросы»).

Соответствие шаблону

С помощью оператора **LIKE** можно сравнить выражение с заданным шаблоном, в котором допускается использование символов-заменителей:

- Символ **%** — вместо этого символа может быть подставлено любое количество произвольных символов.
- Символ _ заменяет один символ строки.
- [] — вместо символа строки будет подставлен один из возможных символов, указанных в квадратных скобках.
- [^] — вместо соответствующего символа строки будут подставлены все символы, кроме указанных в квадратных скобках.

Пример 12. Вывести список автомобилей, у которых первой цифрой номера является цифра 7 или цифра 3.

```
SELECT id FROM cars where id
LIKE '_[37]%'
```

Значение NULL

Оператор **IS NULL** используется для сравнения текущего значения со значением **NULL** — специальным значением, указывающим на отсутствие любого значения. NULL — это не то же самое, что знак пробела (пробел — допустимый символ) или ноль (**0** — допустимое число). NULL отличается и от строки нулевой длины (пустой строки).

Пример 13. Вывести список клиентов, у которых не указан номер телефона.

```
SELECT fio FROM clients where phone
IS NULL
```

Пример 14. Вывести список клиентов, у которых указан номер телефона.

```
SELECT fio, phone FROM clients where phone IS
NOT NULL
```

Сортировка результатов

При выполнении SQL запроса результирующие строки никак не упорядочены, кроме того, один и тот же запрос, будучи запущенным несколько раз подряд, может выдавать строки в разном порядке. Поэтому, если требуется, чтобы результат выполнения запроса был отсортирован по некоторому столбцу (группе столбцов), нужно использовать ключевое слово ORDER BY.

Пример 15. Вывести список клиентов, у которых указан номер телефона, при этом отсортировав записи по фамилии.

```
SELECT
fio, phone FROM clients where phone IS NOT NULL
ORDER BY fio ASC
```

Пример 16. Вывести список клиентов, сгруппировав их по городу проживания и отсортировав сначала по городу, а внутри города — по фамилии.

```
SELECT address, fio FROM clients
ORDER BY address ASC, fio ASC
```

Пример 17. Вывести список предоставляемых услуг, отсортировав их по цене (начиная с самых дорогостоящих).

```
SELECT title, price FROM services
ORDER BY price DESC
```

Соединение и объединение таблиц в запросе

Все SQL запросы, которые нам встречались до текущего момента, обращались в предложении FROM только к одной таблице. А как можно построить запрос, который будет обращаться к столбцам, расположенным в разных таблицах? Для этого в запросе

необходимо указать принцип соединения таблиц. Ведь если этого не сделать, то соединение таблиц будет строиться по принципу декартова произведения, а декартово произведение содержит в себе слишком много лишней информации.
Рассмотрим соединение таблиц по принципу декартового произведения.

Пусть нам необходимо составить таблицу, в которой бы первый столбец содержал фамилии владельцев, а второй — регистрационные номера их автомобилей. Эти данные разнесены по разным таблицам: **cars** и **clients**.

Если мы запишем запрос на выборку в следующем виде:

```
SELECT fio, id FROM cars, clients
```

то первое, с чем мы столкнёмся, будет сообщение о том, что этот запрос в принципе не сможет быть выполнен. Первая причина его ошибочности заключается в том, что столбец **id** является неоднозначным. Он присутствует в обеих таблицах, и интерпретатор SQL в момент исполнения запроса не сможет самостоятельно сделать выбор в пользу одной из таблиц и выдаст ошибку. Решить эту проблему можно за счёт использования полного формата именования столбцов:

```
<имя таблицы>.<имя столбца>
```

Для себя можно сформулировать достаточно простое правило: использование полного формата именования столбцов является более желательным, т.к. устраняет неоднозначности и неопределённости, и, кроме всего прочего, это упрощает процесс чтения SQL запроса.

При использовании полного формата именования столбцов запрос будет выглядеть следующим образом:

```
SELECT clients.fio, cars.id FROM cars, clients
```

Однако это не решит всех проблем. Дело в том, что если после предложения **FROM** имена таблиц перечислены через запятую без указания принципа их соединения, то они будут соединены по принципу декартового произведения, то есть к каждой строке первой таблицы будут по очереди приписаны все столбцы второй таблицы. В терминах нашего примера это будет означать, что к первому клиенту будут приписаны регистрационные номера всех автомобилей, затем ко второму клиенту будут приписаны регистрационные номера всех автомобилей и т.д. В результате будет получена таблица, количество строк в которой будет равняться произведению количеств строк двух исходных таблиц, а количество столбцов в результирующей таблице будет равнять сумме количеств столбцов исходных таблиц.
Конечно же, ненужные строчки могут быть отсечены при помощи предложения **WHERE**:

```
SELECT clients.fio, cars.id

FROM cars, clients

WHERE cars.client_id=clients.id
```

В результате выполнения этого запроса останутся только те строки, где фамилия клиента указана напротив регистрационного номера его автомобиля. Но, несмотря на то, что в результате будет небольшое количество строк, на момент соединения таблиц в предложении **FROM** количество строк соединения может быть довольно велико. Если бы в базе данных было 1000 клиентов и 1000 автомобилей, то в результате было бы 1000 строк. А вот в соединении до выполнения предложения **WHERE** было бы 1000000 строк, что достаточно много даже для современных компьютеров. Поэтому в чистом виде декартово произведение используется достаточно редко, а вместо него применяется внутреннее или внешнее соединение таблиц.

Внутреннее соединение

Для внутреннего соединения таблиц используется ключевое слово **INNER JOIN**. Запишем общий формат использования конструкции **INNER JOIN**:

```
SELECT … FROM
<имя таблицы 1> INNER JOIN <имя таблиц 2>
ON <условие соединения таблиц>
```

В простейшем случае условие соединения таблиц выглядит как равенство соответствующих столбцов.

Пример 18. Получить информацию о владельцах и регистрационных номерах их автомобилей (этот пример аналогичен приведѐнному выше, но строится не на основе декартового произведения).

```
SELECT clients.fio, cars.id
FROM cars INNER JOIN clients
ON cars.client_id=clients.id
```

В результате выполнения операции внутреннего соединения в соединение попадут только те строки, которые удовлетворяют условию соединения таблиц.
А как быть в том случае, если соединить нужно не две, а три таблицы? Принцип используется тот же, но при соединении большего количества таблиц становится важен порядок их следования. В середине должна оказаться таблица, которая будет связываться с двумя остальными.

Пример 19. Получить информацию о владельцах, регистрационных номерах и марках их автомобилей.

```
SELECT clients.fio, cars.id, brands.title
FROM clients INNER JOIN
(cars INNER JOIN brands ON cars.brand_id=brands.id)
ON cars.client_id=clients.id
```

В этом примере таблица **cars** склеивается с таблицей **clients**, плюс к этому таблица **cars** склеивается с таблицей **brands**. Поэтому она (таблица **cars**) должна оказаться посередине, ведь мы не можем напрямую соединить таблицы **clients** и **brands**. Иными словами, сначала произойдёт соединение таблиц **cars** и **brands**, потому что их соединение осуществляется в скобочках, а затем к полученному соединению будет подклеена таблица **clients**.
Альтернативный синтаксис записи этого запроса выглядит следующим образом:

```
SELECT clients.fio, cars.id, brands.title
FROM clients
INNER JOIN cars ON cars.client_id=clients.id
INNER JOIN brands ON cars.brand_id=brands.id
```

Но порядок следования таблиц важен и в этом случае, так, второе предложение **INNER JOIN** не может появиться раньше первого.
Перед тем, как перейти к понятию внешнего соединения таблиц, рассмотрим ещё один пример.

Пример 20. Получить информацию о датах обращения в автосервис владельцев автомобилей, при этом указав регистрационный номер и марку автомобиля. SELECT

cashbook.s_date, clients.fio, cars.id, brands.title

FROM cars

INNER JOIN cashbook ON cashbook.car_id=cars.id

INNER JOIN clients ON cars.client_id=clients.id

INNER JOIN brands ON cars.brand_id=brands.id

В результате будут выведены сведения только о тех клиентов и автомобилях, которым хотя бы раз оказывались некоторые услуги. Если ни одна услуга ни разу не оказывалась, то при внутреннем соединении строка, содержащая информацию об автомобиле, будет проигнорирована, потому как ей не будет найдено соответствие в таблице, содержащей информацию об указанных услугах.

Внешнее соединение

Если же мы хотим вывести и тех клиентов, которым ни разу ни одна услуга не оказывалась, то сделать это можно при помощи внешнего соединения. Чем отличается внешнее соединение от внутреннего? Во внутреннем соединении все таблицы являются равноправными, и не важно, как рассматривать операцию соединения: как приклеивание таблицы номер два к таблице номер один или как приклеивание таблицы номер один к таблице номер два. В случае же внешнего соединения одна из таблиц считается ведущей, а другая подчинённой. Поэтом рассматривают левое внешнее соединение, реализуемое при помощи конструкции **LEFT JOIN,** и правое внешнее соединение, реализуемое при помощи конструкции **RIGHT JOIN**. В первом случае главной является первая таблица, во втором случае вторая.
Важным отличием внешнего соединения от внутреннего является тот факт, что в соединение попадут все строки главной таблицы, даже если им не будет найдено соответствие в подчинённой таблице. А что же тогда будет выведено на том месте, где должны появиться связанные данные из второй таблицы? Там появится в прямом смысле слова «ничего», т.е.
значение NULL.

Пример 21. Получить информацию о датах обращения в автосервис владельцев автомобилей, при этом указав, регистрационный номер и марку автомобиля. Вывести информацию о владельце, даже если он ни разу не обращался в автосервис. SELECT

cashbook.s_date, clients.fio, cars.id, brands.title

FROM cars

LEFT JOIN cashbook ON cashbook.car_id=cars.id

INNER JOIN clients ON cars.client_id=clients.id

INNER JOIN brands ON cars.brand_id=brands.id

Использование ключевого слова UNION

Ключевое слово UNION тоже используется для объединения, но не таблиц, а результатов выполнения запросов на выборку. При этом объединяемые части должны быть совместимы, т.е. иметь одинаковое количество полей с совпадающими типами данных. Так как

результатом выполнения запроса на выборку данных является таблица, то этот результат также может именоваться таблицей наравне с исходными таблицами. Тогда в этой терминологии можно сказать, что объединением двух таблиц является таблица, содержащая все строки, которые имеются в первой таблице, во второй таблице или в обеих таблицах сразу.

Пример 22. Вывести список клиентов, проживающих в Ульяновске или Саратове.

```
SELECT fio FROM clients where
address='Ульяновск'
UNION
SELECT fio FROM clients where
address='Саратов'
```

В этом примере в первой части запроса формируется список клиентов, проживающих в Ульяновске, во второй части запроса формируется список клиентов, проживающих в Саратове, затем обе эти части объединяются в одну общую таблицу при помощи ключевого слова **UNION**. Объединение может быть корректно проведено, т.к. количество столбцов и их тип совпадают.

Разумеется, приведённый выше пример может быть написан при помощи более простых средств, но возможности конструкции **UNION** пока нельзя продемонстрировать в полном объеме — для этого требуется изучить построение вычисляемых полей.

Построение вычисляемых полей

Во всех запросах, рассмотренных выше, в результат выполнения запроса на выборку столбцы данных попадали в неизменном виде. Но вполне естественно, что при написании сложных запросов нам понадобится производить над ними некоторые вычисления, а также формировать новые столбцы. Все эти средства обеспечиваются языком SQL.
В общем случае для создания вычисляемого поля в предложении **SELECT** следует указать некоторое выражение языка SQL. В этих выражениях применяются арифметические операции сложения, вычитания, умножения и деления, а также встроенные функции языка SQL. При построении сложных выражений могут понадобиться скобки.

Ещѐ одним важным моментом, который возникает при построении вычисляемых полей, являются названия этих полей. Если в запросе на выборку присутствует столбец в неизменном виде, то в результате выполнения запроса этот столбец будет называться так же, как и в исходной таблице. А как быть, если этого столбца раньше не было, и он был сформирован в результате выполнения запроса? В этом случае необходимо вспомнить, что при определении общего синтаксиса SQL запроса было введено такое понятие как псевдоним. При помощи псевдонимов можно давать имена не только вновь сформированным столбцам, но и переименовывать существующие столбцы. В каком случае может понадобиться переименование существующего столбца? Например, нужно построить запрос, в котором должна фигурировать фамилия клиента (**fio**) и фамилии мастера (**fio**). При написании запроса можно разделить названия этих столбцов за счёт использования полного синтаксиса именования столбцов, но в результирующей таблице получится два разных столбца с одинаковыми названиями. Некоторые СУБД могут обеспечить автоматическое переименование столбцов в подобных ситуациях, например, первый столбец будет называться (**fio1**), а второй столбец (**fio2**), но это не всегда удобно. Поэтому лучше это сделать самостоятельно, указав соответствующий псевдоним.

Пример 23. Для каждой оказанной услуги указать дату оказания, количество затраченных часов, стоимость одного часа работы,

итоговую стоимость и регистрационный номер автомобиля, над которым проводилась работа. SELECT

cashbook.car_id AS регистрационный_номер_машины, services.title

AS вид_услуги, cashbook.s_date AS дата_оказания_услуги,

cashbook.s_time AS затраченное_время_в_часах, services.price AS

цена_одного_часа, services.price*cashbook.s_time AS итого FROM

services INNER JOIN cashbook ON services.id=cashbook.service_id

Важно ещё упомянуть тот факт, что при именовании столбцов использование символа пробела и символов национальных алфавитов является нежелательным, но допустимым. Для того чтобы содержащее пробелы имя столбца интерпретировалось как единое целое, оно может заключаться в квадратные скобки или символы кавычек.

Использование агрегатных функций

С помощью итоговых (агрегатных, или обобщающих) функций в рамках SQL-запроса можно получить обобщающие статистические сведения о множестве отобранных значений. Пользователю доступны следующие основные итоговые функции:

- **COUNT(Выражение)** — возвращает количество записей в выходном наборе SQLзапроса;
- **MIN/MAX(Выражение)** — возвращает наименьшее и наибольшее из множества значений в некотором поле запроса;
- **AVG(Выражение)** — возвращает среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей.
- **SUM(Выражение)** — возвращает сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Чаще всего в качестве выражения выступают отдельные столбцы, но также могут записываться и более сложные выражения, содержащие в качестве аргументов имена столбцов, принадлежащих различным таблицам.
Функции MIN, MAX и COUNT применимы как к числовым, так и к нечисловым полям. Функции AVG и SUM могут применяться только к числовым полям.
Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово **DISTINCT**. Оно не имеет смысла для функций **MIN** и **MAX**, однако его использование может повлиять на результаты выполнения остальных функций.

Очень важно отметить, что итоговые функции могут использоваться только в списке предложения **SELECT** и в составе предложения **HAVING**. Ещѐ одно правило использования итоговых функций состоит в том, что все столбцы, которые входят в предложение **SELECT** и не входят в итоговую функцию, должны войти в предложение **GROUP BY**, обеспечивая объединение данных в группы. Обратное правило звучит несколько иначе: в предложение GROUP BY могут входить столбцы, которые не входят в предложение SELECT.

Пример 24. Определить первого по алфавиту сотрудника автомастерской.

SELECT MIN(fio) AS первый_по_алфавиту FROM masters

При использовании функции COUNT можно использовать мнемонику *, т.к. в некоторых случаях неважно, на каком столбце вычисляется эта функция. В этом примере имеет

значение только количество строк в результирующем запросе, поэтому имя столбца может быть не указано.

Пример 25. Определить, сколько раз в автосервисе оказывались услуги автомобилю с регистрационным номером "x666xx99rus".

```
SELECT COUNT(*) AS количество_обращений
FROM cashbook
WHERE car_id='x666xx99rus'
```

Явное указание имени столбца для функции COUNT может быть актуальным только при использовании внешнего объединения таблиц, где количество значений в каждом столбце может быть различно.

Пример 26. Определить среднюю стоимость услуг, оказываемых в автосервисе.

```
SELECT AVG(price) AS средняя_цена
FROM services
```

Следующий пример показывает, каким образом аргументом итоговой функции может быть не отдельный столбец, а более сложное выражение.

Пример 27. Определить, на какую сумму были оказаны услуги автомобилю с регистрационным номером "x666xx99rus".

```
SELECT
SUM(cashbook.s_time*services.price) AS сумма FROM
cashbook INNER JOIN services
ON cashbook.service_id=services.id
WHERE car_id='x666xx99rus'
```

Ограничения на группировку данных

Запрос, в котором присутствует ключевое слово **GROUP BY**, называется группирующим запросом, поскольку в нем группируются данные, полученные в результате выполнения операции SELECT. Для каждой отдельной группы создается единственная суммарная строка.

Стандарт SQL требует, чтобы предложение SELECT и фраза **GROUP BY** были тесно связаны между собой. При наличии в операторе **SELECT** ключевого слова **GROUP BY** каждый элемент списка в предложении SELECT должен иметь единственное значение для всей группы.

Если совместно с **GROUP BY** используется предложение **WHERE**, то оно обрабатывается первым, а группированию подвергаются только те строки, которые удовлетворяют условию поиска.

Ещё одной подсказкой для использования ключевого слова GROUP BY являются следующие формулировки в вербальном описании запроса: «для каждого…», «для каждой…»

Пример 28. Для каждого клиента определить количество автомобилей, которыми он владеет.

```
SELECT clients.fio AS
ФИО,
COUNT(*) AS количество_автомобилей FROM
clients INNER JOIN cars
ON clients.id=cars.client_id
GROUP BY clients.fio
```

Столбец clients.fio не входит в итоговую функцию, но он входит в предложение SELECT, кроме всего прочего, в тексте запроса используется ключевая фраза «для каждого клиента», следовательно, этот столбец должен войти в предложение GROUP BY.

С целью сокращения длины запроса могут использоваться псевдонимы не только для имён столбцов, но и для имён таблиц (это не единственное применение механизма псевдонимов таблиц — его основное назначение в именовании динамически сформированных таблиц; этот вопрос будет рассмотрен в следующем разделе). Перепишем пример 28, используя псевдонимы для таблиц cars и clients.

Пример 29. Для каждого клиента определить количество автомобилей, которыми он владеет.

```
SELECT CL.fio AS
ФИО,
count(*) AS количество_автомобилей FROM
clients AS CL
INNER JOIN cars AS CR
ON CL.id=CR.client_id
GROUP BY CL.fio
```

Возможно, в этом запросе выгода от сокращения записи и не видна, но в действительно сложных запросах сокращения заметно упрощают чтение текста запроса.

Пример 30. Определить, на какую сумму каждому автомобилю были оказаны услуги.

```
SELECT
C.car_id AS регистрационный_номер_автомобиля,
```

```
SUM(C.s_time*S.price) AS сумма FROM
cashbook AS C INNER JOIN services AS S
ON C.service_id=S.id
GROUP BY C.car_id
```

Отметим, что запрос из примера 27, в котором рассчитывалась сумма оказанных услуг для конкретного автомобиля, и запрос из примера 30 очень похожи.

Пример 31. Определить, на какую сумму каждому автовладельцу были оказаны услуги.

```
SELECT
CL.fio AS ФИО,
SUM(CS.s_time*S.price) AS сумма FROM
cashbook AS CS
INNER JOIN services AS S ON CS.service_id=S.id
INNER JOIN cars AS CR ON CS.car_id=CR.id
INNER JOIN clients AS CL ON CL.id=CR.client_id
GROUP BY CL.fio
```

Несмотря на возрастающую сложность запроса и количество объединяемых таблиц, общий принцип построения остаèтся прежним: всè, что не вошло в итоговую функцию, должно войти в предложение **GROUP BY**.

Пример 32. Определить выручку автомастерской за каждый месяц.

```
SELECT MONTH(CS.s_date) AS месяц,
SUM(CS.s_time*S.price) AS сумма FROM
cashbook AS CS
INNER JOIN services AS S ON CS.service_id=S.id
GROUP BY MONTH(CS.s_date)
```

Для получения номера месяца используется функция **MONTH()**, которая возвращает целое число в диапазоне от **1** до **12**. Аналогично могут использоваться функции **YEAR()** и **DAY()**, которые возвращают соответственно номер года и номер дня.
Несмотря на то, что столбец **MONTH(CS.s_date)** получил псевдоним **месяц**, им нельзя пользоваться внутри этого запроса; и в предложении **GROUP BY** приходится записать то же самое выражение, которое использовалось для построения вычисляемого столбца.
При помощи **HAVING** отбираются все предварительно сгруппированные посредством **GROUP BY** блоки данных, удовлетворяющие заданным в **HAVING** условиям.
Условия в **HAVING** отличаются от условий в **WHERE**:

- **HAVING** исключает из результирующего набора данных группы с результатами агрегированных значений;
- **WHERE** исключает из расчета агрегатных значений по группировке записи, не удовлетворяющие условию;
- в условии поиска **WHERE** нельзя задавать агрегатные функции.

Пример 33. Определить клиентов, которые владеют более чем одним автомобилем. SELECT clients.fio AS ФИО,

COUNT(*) AS количество_автомобилей FROM

clients INNER JOIN cars

ON clients.id=cars.client_id

GROUPBYclients.fio

HAVINGCOUNT(*)>1

В этом примере мы вновь сталкиваемся с той же проблемой, что и в примере 32: несмотря на то, что для вновь сформированного столбца был обозначен псевдоним, мы не можем использовать этот псевдоним в предложении **HAVING** и вынуждены повторить запись выражения, которое лежит в основе создания столбца.

Пример 34. Определить месяцы, в которые выручка автомастерской не превышала 10000.

SELECT

MONTH(CS.s_date) AS месяц,

SUM(CS.s_time*S.price) AS сумма FROM

cashbook AS CS

INNER JOIN services AS S ON CS.service_id=S.id

GROUP BY MONTH(CS.s_date)

HAVING SUM(CS.s_time*S.price)<=10000

Подзапросы

Очень часто получить необходимые данные при помощи одного запроса не представляется возможным. В этих случаях используется такой механизм, как вложенные запросы, или подзапросы.

Внутренний подзапрос представляет собой такой же оператор **SELECT**, как и внешний. Внешний оператор **SELECT** имеет доступ к результатам, полученным во внутреннем запросе, а внутренний запрос может обращаться не только к данным своих таблиц, но и к таблицам, которые используются во внешнем запросе.

Подзапрос — это инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором.

Текст подзапроса должен быть заключен в скобки. К подзапросам применяются следующие правила и ограничения:

- Внутренние запросы обязательно должны быть помещены после оператора сравнения. Ключевое слово **ORDER BY** не используется, хотя и может присутствовать во внешнем подзапросе.
- Список в предложении **SELECT** внутреннего запроса должен состоять из имен отдельных столбцов или составленных из них выражений — за исключением случая, когда в подзапросе присутствует ключевое слово **EXISTS**.
- По умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в предложении **FROM** внутреннего запроса. Однако, допускается ссылка и на столбцы таблицы, указанной во фразе **FROM** внешнего запроса.

Подзапрос может встречаться в одной из следующих секций:

- в предложении **SELECT**, в предложении **WHERE**,

- в предложении **HAVING**, в предложении **FROM**.

Подзапросы, возвращающие единичные значения

Выделяют скалярные подзапросы, возвращающие единичные значения, и табличные подзапросы, возвращающие множественные значения. Рассмотрим скалярные подзапросы.

Пример 35. Определить самого молодого владельца автомобиля.

```
SELECT fio
FROM clients
WHERE birth=
(SELECT MIN(birth) FROM clients)
```

В этом запросе сначала будет выполнен внутренний подзапрос, который вычисляет минимальный возраст клиента. Но нам нужно не само значение минимального возраста, а фамилия клиента, который обладает минимальным возрастом. Фамилия определяется уже на уровне внешнего запроса.

Наиболее часто встречающаяся ошибка при написании подобных запросов состоит в том, что студенты сортируют клиентов по возрасту и выбирают самую верхнюю строчку из отсортированной таблицы. На самом деле они совершают две грубые ошибки. Первая состоит в том, что операция поиска минимума или максимума имеет гораздо меньшую сложность по сравнению с операцией сортировки. Вторая состоит в том, что может существовать несколько строк таблицы, удовлетворяющих условию минимума. Если произвести сортировку и взять в ответ только верхнюю строку, то остальные строки будут утеряны.

Пример 36. Определить услуги, стоимость которых превосходит среднее, также вывести превышение над средним.

```
SELECT title
AS название_услуги, price AS цена,
price-(SELECT AVG(price) FROM services) AS превышение
FROM services
WHERE price>
(SELECT AVG(price) FROM services)
```

В этом примере подзапрос на вычисление среднего арифметического встречается два раза: один раз в предложении **SELECT** как часть арифметического выражения для вычисления превышения над средним арифметическим, а другой раз в предложении **WHERE** как часть условия для отбора только тех строк, которые отвечают поставленному условию.

Пример 37. Определить даты, когда оказывалась самая дорогая услуга.

```
SELECT DISTINCT s_date
FROM cashbook
WHERE service_id=
(
 SELECT id FROM services
 WHERE price=
 (SELECT MAX(price) FROM services) )
```

Пример 37 демонстрирует, что уровень вложенности запросов может быть больше, чем один. Подзапрос второго уровня определяет стоимость самой дорогой услуги. Подзапрос первого уровня определяет **id** самой дорогой услуги. И, наконец, запрос самого верхнего уровня определяет те дни, когда оказывалась услуга с выбранным **id**. Важным нюансом в написании этого запроса является использование ключевого слова **DISTINCT**, т.к. в один и тот же день самая дорогая услуга может оказываться более одного раза.

Пример 38. Определить клиентов, которым оказывалась самая дорогая услуга.

```
SELECT DISTINCT
CL.fio FROM
clients AS CL INNER JOIN cars AS CR
ON CL.id=CR.client_id
INNER JOIN cashbook AS CS
ON CS.car_id=CR.id
INNER JOIN services AS S
ON S.id=CS.service_id
WHERE service_id=
(
 SELECT id FROM services
 WHERE price=
 (SELECT MAX(price) FROM services)
)
```

Как и в примере 37, в примере 38 необходимо использовать ключевое слово **DISTINCT**, т.к. одному и тому же клиенту самая дорогая услуга может оказываться более одного раза. Наконец, рассмотрим пример 39.

Пример 39. Определить клиентов, для которых средняя цена услуг (еѐ необходимо вывести на экран), которыми они пользуются, выше аналогичного показателя для полного перечня доступных услуг.

```
SELECT CL.fio AS ФИО, AVG(S.price) AS
средняя_цена_услуги FROM
clients AS CL INNER JOIN cars AS CR ON
CL.id=CR.client_id
INNER JOIN cashbook AS CS
ON CS.car_id=CR.id
INNER JOIN services AS S
ON S.id=CS.service_id
GROUP BY CL.fio
HAVING AVG(S.price)>
(SELECT AVG(price) FROM services)
```

Подзапросы, возвращающие множественные значения

Во многих случаях значение, подлежащее сравнению в предложениях **WHERE** или **HAVING**, представляет собой не одно, а несколько значений. Вложенные подзапросы генерируют непоименованное промежуточное отношение — временную таблицу. Оно может использоваться только в том месте, где появляется в подзапросе.

Применяемые к подзапросу операции основаны на тех операциях, которые, в свою очередь, применяются к множеству, а именно:

- **{WHERE | HAVING} выражение [NOT] IN (подзапрос)**;
- **{WHERE | HAVING} выражение оператор_сравнения {ALL | ANY} (подзапрос)**;
- **{WHERE | HAVING } [NOT] EXISTS (подзапрос)**;

Несмотря на то, что временный запрос является неименованной таблицей, мы можем дать ей имя, которое будет доступно в рамках запроса более высокого уровня.

Пример 40. Определить клиентов, которым были оказаны услуги на сумму, превышающую 10000.

```
SELECT T.ФИО
FROM
(SELECT
CL.fio AS ФИО,
SUM(CS.s_time*S.price) AS сумма FROM
cashbook AS CS
INNER JOIN services AS S ON CS.service_id=S.id
INNER JOIN cars AS CR ON CS.car_id=CR.id
INNER JOIN clients AS CL ON CL.id=CR.client_id
GROUP BY CL.fio) AS T
WHERE T.сумма>10000
```

В примере 40 результат выполнения подзапроса содержит более одной строки и более одного столбца, т.е. этот результат можно интерпретировать как таблицу. Для того, чтобы с этой таблицей можно было работать, дадим ей имя. Это может быть сделано при помощи псевдонима. В этом примере временная таблица получает псевдоним **T**, и в рамках запроса верхнего уровня можно обращаться к её столбцам на основании тех псевдонимов, которые им были даны на уровне внутреннего запроса.

Напомним, что оператор **IN** используется для сравнения некоторого значения со списком значений, при этом проверяется, входит ли значение в предоставленный список или сравниваемое значение не является элементом представленного списка. Этот оператор уже рассматривался ранее, но список потенциальных значений был статичный. Рассмотрим случаи, когда этот список может генерироваться динамически при помощи вложенных подзапросов.

Пример 41. Определить клиентов, которым оказывалась самая дорогая услуга.

```
SELECT fio FROM clients
WHERE clients.id IN
```

```
(
  SELECT client_id FROM cashbook
  INNER JOIN services   ON
cashbook.service_id=services.id
  INNER JOIN cars
  ON cashbook.car_id=cars.id
  WHERE price=
  (SELECT MAX(price) FROM services)
)
```

Запрос из примера 41 уже рассматривался в примере 38, но там он был реализован более сложным способом. А вот пример 42 без конструкции IN решить уже будет затруднительно.

Пример 42. Определить клиентов, которым оказывалась самая дорогая услуга и которым никогда не оказывалась самая дешёвая услуга.

```
SELECT fio FROM clients
WHERE clients.id IN
(
  SELECT client_id
  FROM cashbook
  INNER JOIN services   ON
cashbook.service_id=services.id
  INNER JOIN cars
  ON cashbook.car_id=cars.id
  WHERE price=
  (SELECT MAX(price) FROM services)
) AND
clients.id NOT IN
(
  SELECT client_id
  FROM cashbook
  INNER JOIN services   ON
cashbook.service_id=services.id
  INNER JOIN cars
  ON cashbook.car_id=cars.id
  WHERE price=
  (SELECT MIN(price) FROM services)
)
```

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 тогда, когда оно выполняется для всех значений в результирующем столбце подзапроса.

Если записи подзапроса предшествует ключевое слово **ANY**, то условие сравнения считается выполненным, если оно выполняется хотя бы для одного из значений в результирующем столбце подзапроса.

Если в результате выполнения подзапроса получено пустое значение, то для ключевого слова ALL условие сравнения будет считаться выполненным, а для ключевого слова ANY — невыполненным.

Пример 43. Определить автовладельцев, которым были оказаны услуги на самую большую сумму.

```
SELECT
CL.fio AS ФИО
FROM cashbook
AS CS
INNER JOIN services AS S
ON CS.service_id=S.id
INNER JOIN cars AS CR
ON CS.car_id=CR.id
INNER JOIN clients AS CL
ON CL.id=CR.client_id
GROUP BY CL.fio
HAVING SUM(CS.s_time*S.price)>=ALL
(
 SELECT
SUM(CS.s_time*S.price) AS сумма   FROM
cashbook AS CS
 INNER JOIN services AS S
 ON CS.service_id=S.id
 INNER JOIN cars AS CR
 ON CS.car_id=CR.id
 INNER JOIN clients AS CL
 ON CL.id=CR.client_id
GROUP BY CL.fio
)
```

В результате выполнения подзапроса формируется столбец сумм, которые были уплачены каждым из автовладельцев. Следует обратить внимание на то, что сама фамилия не участвует в предложение **SELECT**, но участвует в предложении GROUP BY. Отметим также,

что в запросе верхнего уровня итоговая функция не присутствует в предложении **SELECT**, но присутствует в предложении **HAVING**.

Важен также тот факт, что для сравнения используется операция «больше или равно», а не операция «больше». Если применить операцию «больше», то запрос всегда будет возвращать пустое множество, т.к. не может быть клиентов, которые потратили больше себя.

Подобный принцип построения запроса может быть не самым эффективным, т.к. при его выполнении растёт и группировка итоговых сумм будет производиться дважды: первый раз во внутреннем подзапросе, а второй раз в запросе верхнего уровня. Кроме того, сам текст запроса содержит достаточно большие повторяющиеся куски. В целом эта задача могла бы быть более элегантно решена при помощи пользовательских просмотров.
Рассмотрим еще один пример.

Пример 44. Определить автовладельцев, которым были оказаны услуги на сумму, большую, чем сумма, уплаченная хотя бы одним жителем Ульяновска.

```
SELECT
CL.fio AS ФИО
FROM cashbook
AS CS
INNER JOIN services AS S
ON CS.service_id=S.id
INNER JOIN cars AS CR
ON CS.car_id=CR.id
INNER JOIN clients AS CL
ON CL.id=CR.client_id
GROUP BY CL.fio
HAVING
SUM(CS.s_time*S.price)>ANY
(
 SELECT
SUM(CS.s_time*S.price) AS сумма   FROM
cashbook AS CS
 INNER JOIN services AS S
 ON CS.service_id=S.id
 INNER JOIN cars AS CR
 ON CS.car_id=CR.id
 INNER JOIN clients AS CL
 ON CL.id=CR.client_id
 WHERE CL.address='Ульяновск'
 GROUP BY CL.fio
```

)

В этом примере во внутреннем запросе формируется столбец сумм, которые были уплачены жителями Ульяновска, а во внешнем запросе выбираются те, для которых этот показатель является более высоким хотя бы в одной позиции.

Ключевые слова **EXISTS** и **NOT EXISTS** предназначены для использования только совместно с подзапросами. 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 операции **EXISTS** будет значение **FALSE**.

Для ключевого слова **NOT EXISTS** используются правила обработки, обратные по отношению к ключевому слову **EXISTS**. Поскольку, по ключевым словам,**EXISTS** и **NOT EXISTS** проверяется лишь наличие строк в результирующей таблице подзапроса, то эта таблица может содержать произвольное количество столбцов.

Пример 45. Определить автовладельцев, которым была оказана хотя бы одна услуга.

```
SELECT fio
FROM clients
WHERE EXISTS
(SELECT * FROM cashbook
INNER JOIN cars
ON cars.id=cashbook.car_id
WHERE client_id=clients.id)
```

Пример 46. Определить автовладельцев, которым не было оказано ни одной услуги.

```
SELECT fio
FROM clients
WHERE NOT EXISTS
(SELECT * FROM cashbook
INNER JOIN cars
ON cars.id=cashbook.car_id
WHERE client_id=clients.id)
```

В этих двух примерах используется приём, при котором из внутреннего запроса осуществляется обращение к столбцам запроса верхнего уровня. Во внутреннем подзапросе происходит обращение к таблице **cashbook** и делается попытка выбрать из неё все строчки, относящиеся к некоторому фиксированному текущему клиенту. Затем в запросе верхнего уровня фиксируется следующий клиент, и для него повторяется процесс выполнения внутреннего запроса.

3. ВАРИАНТЫ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1 Вариант: Деятельность торговой фирмы

В базе данных учесть следующие признаки: дату, количество, наименование, тип, город где был изготовлен товар, цену проданного товара, покупателя, его фирму, город, телефон.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;
    - ограничений NOT NULL;
    - ограничений первичного ключа PRIMARY KEY;
    - ограничений внешнего ключа FOREIGN KEY;
    - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить покупателя, который купил максимальное количество товаров”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждой покупки рассчитать общую стоимость”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить сумму продажи для каждого месяца”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить покупателей, купивших товаров на сумму, превышающую среднюю сумму покупок всех покупателей”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить тип товаров, которого куплено больше всего”*.

2 Вариант: Деятельность предприятия по сборке изделий

В базе данных учесть следующие признаки: наименование, тип, цену продажи некоторого изделия, количество дней на его сборку, количество компонент в изделии, описание, изготовителя компонент, страну изготовителя компонент, тип и стоимость каждого компонента.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;

- ограничений NOT NULL;
- ограничений первичного ключа PRIMARY KEY;
- ограничений внешнего ключа FOREIGN KEY;
- правил RULE.

3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждого изделия рассчитать его стоимость”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти изделия, в состав которых входит больше всего компонентов”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить компоненты, которые входят в большее число изделий”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Вычислить наценку для каждого изделия”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти изделия, на сборку которых уходит дней больше, чем в среднем на сборку изделий”*.

3 Вариант: Деятельность стола заказов

В базе данных учесть следующие признаки: дату получения и исполнения заказа, скидку на заказ, количество и цену товара, вошедшего в заказ, имя клиента, его расчетный счет и величину кредита.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить заказ, на выполнение которого ушло больше всего дней”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить клиентов, стоимость заказов которых превысила их кредит”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Рассчитать стоимость каждого заказа с учетом скидки”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить клиента, который купил больше всего товаров”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить город, где живет клиент, чаще других оформляющий заказы”*.

4 Вариант: Оплата коммунальных услуг

В базе данных учесть следующие признаки: фамилию квартиросъемщика, его адрес, жилую площадь, число людей проживающий в квартире, дату (предельную и фактическую) оплаты коммунальных услуг, стоимость одного квадратного метра жилплощади, стоимость потребления холодной воды на одного проживающего.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Рассчитать для каждого квартиросъемщика квартплату за каждый месяц”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить задолжников по квартплате за каждый месяц”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить дом с максимальной жилой площадью”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить дом с максимальной плотностью населения”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить дома, средняя площадь квартир в которых больше средней площади квартир других домов”*.

5 Вариант: Работа фирмы с поставщиками

В базе данных учесть следующие признаки: дату продажи некоторого товара, количество, цену, скидку при продаже и налог на продажу, транспортные расходы, а также поставщиков товара, страну и наличие лицензии на продажу.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;

- ограничений по умолчанию DEFAULT;
- ограничений NOT NULL;
- ограничений первичного ключа PRIMARY KEY;
- ограничений внешнего ключа FOREIGN KEY;
- правил RULE.

3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Рассчитать общую стоимость поставленного товара с учетом транспортных расходов, скидки и налога”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить сумму налога за каждый месяц”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить страну, в которой изготовлен товар, пользующийся наибольшей популярностью”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить самый дешевый товар, поступающий без лицензии”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить страну, товар из которой приходит с максимальными транспортными расходами”*.

6 Вариант: Начисление зарплаты

В базе данных учесть следующие признаки: фамилию, адрес, телефон сотрудника, дату его рождения и дату устройства на работу, дату, вид и количество в часах выполненной работы, описание выполненной работы, тип освобождения от налога, нижнюю и верхнюю границы оплаты одного часа.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить сумму выплат по каждому виду работ за каждый месяц (как нижняя грань оплаты, если сотрудник работает меньше года, средняя оплата, если сотрудник работает от года до пяти лет, и верхняя грань — более пяти лет)”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждого сотрудника рассчитать его ежемесячный заработок”*.

5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти сотрудника, который работает дольше других”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Вычислить сумму налога, которую фирма платит каждый месяц”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить сотрудников, ежемесячная оплата которых оказалась больше средней”*.

7 Вариант: Деятельность бюро добрых услуг

В базе данных учесть следующие признаки: вид услуги, ее описание и стоимость, дату оказания этой услуги, скидку при оплате в зависимости от социального положения клиента, имя и место проживания клиента (город, село).

1. Разработать структуру базы данных согласно варианту.

2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:

   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.

3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить услугу, пользующуюся наибольшей популярностью”*.

4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждого клиента рассчитать стоимость услуг с учетом социального положения и скидок”*.

5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить доход фирмы от предоставленных услуг за каждый месяц”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить, жители города или села чаще всего обращаются в фирму”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Рассчитать количество и сумму предоставленных населению услуг по категориям, определенным социальным происхождением клиентов”*.

8 Вариант: Оплата междугородних телефонных разговоров

В базе данных учесть следующие признаки: дату и время, продолжительность телефонного разговора, город, с которым состоялся разговор, фамилию, адрес, номер телефона клиента, тарифы городов и скидки на время разговора в течение суток.

1. Разработать структуру базы данных согласно варианту.

2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;
    - ограничений NOT NULL;
    - ограничений первичного ключа PRIMARY KEY;
    - ограничений внешнего ключа FOREIGN KEY;
    - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждого клиента вычислить сумму оплаты междугородних разговоров”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить город, с которым чаще всего разговаривают клиенты”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить клиента, который говорит по телефону чаще и дольше других”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить время суток, на которое приходится больше всего разговоров”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить день, в который телефонная линия была занята меньше всего”*.

9 Вариант: Поваренная книга

В базе данных учесть следующие признаки: названия и типы блюд, описание компонент блюда с указанием количества в граммах, калорийности и стоимости 1 грамма, количества жиров, углеводов и белков в 1 грамме компонента.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;
    - ограничений NOT NULL;
    - ограничений первичного ключа PRIMARY KEY;
    - ограничений внешнего ключа FOREIGN KEY;
    - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Вычислить стоимость и калорийность каждого блюда”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить блюдо из супов с наименьшим содержанием жиров”*.

5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить компоненты самого дорогого блюда”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти компонент, который входит в большинство блюд”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить содержание жира, белков и углеводов в самом дорогом блюде самого дешевого в среднем типа блюд”*.

10 Вариант: Книжная палата

В базе данных учесть следующие признаки: дату и количество проданных книг, название, автора, издательство, тематику, цену проданной книги, сведения об авторе: фамилию, пол, дату рождения.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить тематику, по которой продается больше всего книг”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“По каждому месяцу вычислить сумму продаж”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить, книги каких авторов пользуются наибольшей популярностью, авторов-мужчин или авторов-женщин”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить дни, когда было продано книг больше, чем обычно (т.е. больше среднего)”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Какие по тематике книги пишут молодые авторы”*.

11 Вариант: Музыкальная коллекция

В базе данных учесть следующие признаки: дату, количество и стоимость проданного альбома, страну, авторов слов и музыки, исполнителя, длительность каждой композиции в альбоме, название альбома и год выхода.

1. Разработать структуру базы данных согласно варианту.

2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;
    - ограничений NOT NULL;
    - ограничений первичного ключа PRIMARY KEY;
    - ограничений внешнего ключа FOREIGN KEY;
    - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Вычислить сумму продаж по каждому месяцу”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить страну, выпустившую самый долгозвучащий диск”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить песню, пользующуюся наибольшей популярностью”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Составить рейтинг исполнителей по каждому месяцу”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить автора слов, написавшего больше всех песен”*.

12 Вариант: Видеотека

В базе данных учесть следующие признаки: дату продажи видеокассеты, название фильма, страну, режиссера, тематику фильма, наличие Оскаров, дату выпуска фильма, стоимость кассеты, информацию о покупателе: возраст, пол, социальное положение.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;
    - ограничений NOT NULL;
    - ограничений первичного ключа PRIMARY KEY;
    - ограничений внешнего ключа FOREIGN KEY;
    - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить сумму продаж по каждому месяцу”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить, какой тип покупателей чаще других покупает видеокассеты”*.

5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Какой самый старый фильм был продан за последний месяц”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить страну, завоевавшую своими фильмами больше всего Оскаров”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Какие по тематике фильмы смотрит молодежь”*.

13 Вариант: Олимпийские игры

В базе данных учесть следующие признаки: номер, символ олимпиады, город проведения, наличие в городе гор или моря, дату открытия и закрытия, число видов спорта, по которым проводятся соревнования, команды-участницы, количество спортсменов в команде, число завоеванных золотых, серебряных и бронзовых медалей каждой командой.

1. Разработать структуру базы данных согласно варианту.

2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:

   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.

3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждой олимпиады рассчитать отношения числа завоеванных медалей к числу участников”*.

4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить команду, для которой отношение числа завоеванных золотых медалей к числу участников больше, чем аналогичный показатель олимпиады”*.

5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить команды, которые чаще других участвовали в олимпиадах”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить команды, которые по числу завоеванных медалей на протяжении всех олимпиад являются лидерами”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти олимпиады, символы которых совпадали”*.

14 Вариант: Учебный процесс

В базе данных учесть следующие признаки: фамилии студентов, дату рождения, курс, дату сдачи, оценку и название предмета для каждого студента, для каждого предмета указать число часов на изучение, код предмета: гуманитарный блок, математический или профессиональный и кафедру, которая ведет данный предмет.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;
    - ограничений NOT NULL;
    - ограничений первичного ключа PRIMARY KEY;
    - ограничений внешнего ключа FOREIGN KEY;
    - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить предмет, по которому нет двоек”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить студентов, сдавших успешно экзамены и набравших в сумме часов больше, чем среднее число часов по всем студентам”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить блок дисциплин, средняя оценка по которым самая высокая”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить кафедру, по предметам которой получено больше всего двоек студентами младших курсов”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти студентов, сдавших все экзамены успешно, если их день рождения пришелся на период сдачи экзаменов”*.

15 Вариант: Учебная нагрузка преподавателя

В базе данных учесть следующие признаки: фамилию, должность, звание преподавателя, кафедру, на которой он работает, название предмета, который он ведет, для предмета указать название, длительность в часах, код предмета: гуманитарный блок, математический или профессиональный, для каждой должности указать стоимость часа.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
    - проверочных ограничений CHECK;
    - ограничений по умолчанию DEFAULT;
    - ограничений NOT NULL;
    - ограничений первичного ключа PRIMARY KEY;
    - ограничений внешнего ключа FOREIGN KEY;
    - правил RULE.

3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Вычислить зарплату каждого преподавателя”*.

4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить блок дисциплин, которые читают самые квалифицированные преподаватели”*.

5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить кафедру, для которой отношение числа предметов к числу преподавателей самое большое”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить кафедры, на которые приходится учебная нагрузка в часах, больше средней по кафедрам”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить предмет, по которому предусмотрено самое большое количество часов на изучение”*.

16 Вариант: Продажа билетов на самолеты

В базе данных учесть следующие признаки: дату продажи билета, номер рейса, дату вылета рейса, номер места, фамилию пассажира, его социальное положение, данные по типу самолета, обслуживающего рейс: тип самолета, стоимость билета, конечный пункт, продолжительность маршрута, экипаж, квалификация командира, его возраст и стаж полетов.

1. Разработать структуру базы данных согласно варианту.

2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:

   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.

3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Вычислить прибыль от каждого рейса”*.

4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить рейсы до заданного пункта, на которые остались свободные места”*.

5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти отношение количества рейсов дальнего следования, которые выполняют квалифицированные командиры экипажей, к общему числу рейсов дальнего следования”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Какой экипаж имеет больше всего налетов, по количеству и по продолжительности”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Какие пассажиры по своему социальному положению летают чаще других”*.

17 Вариант: Автобусный парк

В базе данных учесть следующие признаки: дату подачи и дату исполнения заявки, продолжительность и вид поездки, число участников, организация сделавшая заказ, сумма аванса, выплаченного за поездку, марка выделенного автобуса, его техническое состояние, количество мест, стоимость билета, налоги и скидки в зависимости от вида поездки.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить водителей с плохим техническим состоянием автобуса, чаще других отправляющихся в рейс”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Вычислить прибыль от поездок за каждый месяц”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Какие поездки пользуются наибольшей популярностью”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждой организации рассчитать долг с учетом стоимости поездок и выплаченного аванса”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить водителя, совершившего больше всего поездок”*.

18 Вариант: Финансовое состояние вузов

В базе данных учесть следующие признаки: число госбюджетных и хоздоговорных студентов в каждой студенческой группе, число преподавателей на факультете, среднюю стоимость обучения одного студента на факультете, среднюю зарплату преподавателя по университету, название университета, город, фамилию ректора, объем госбюджетных поступлений и дотаций из местного бюджета для каждого университета.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:

- проверочных ограничений CHECK;
- ограничений по умолчанию DEFAULT;
- ограничений NOT NULL;
- ограничений первичного ключа PRIMARY KEY;
- ограничений внешнего ключа FOREIGN KEY;
- правил RULE.

3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Для каждого вуза рассчитать объем свободных наличных средств”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить факультеты с самым большим отношением числа студентов к числу преподавателей”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить вуз с самой низкой средней стоимостью обучения одного студента”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Найти вуз с самым большим числом хоздоговорных студентов”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить, какая сумма приходится на каждого студента”*.

19 Вариант: Областное УВД

В базе данных учесть следующие признаки: город, дату совершения и дату раскрытия преступления, вид и тяжесть преступления, описать участников: фамилию, дату рождения, вид участия, описать примененное оружие: марку, страну изготовления, за кем числится.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить, преступления какого вида раскрываются быстрее других”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Оружие какой страны наиболее часто используется в преступлениях”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить сколько преступлений и какого вида приходится на каждую возрастную группу”*.

6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“В каком городе преступления раскрываются быстрее, чем в других городах”*.

7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“В какой месяц было совершено больше всего преступлений”*.

20 Вариант: Фирма по продаже подержанных автомобилей

В базе данных учесть следующие признаки: дату продажи, продавца, вид оплаты, данные о покупателе: фамилию, пол, возраст, социальное положение, информацию об автомобиле: марка, цвет, изготовитель, дата изготовления, техническое состояние, мощность двигателя.

1. Разработать структуру базы данных согласно варианту.
2. Создать и заполнить таблицы в среде MS SQL Server. При необходимости определитьпользовательские типы данных. Обеспечить целостность данных путем реализации на сервере бизнес-правила в виде:
   - проверочных ограничений CHECK;
   - ограничений по умолчанию DEFAULT;
   - ограничений NOT NULL;
   - ограничений первичного ключа PRIMARY KEY;
   - ограничений внешнего ключа FOREIGN KEY;
   - правил RULE.
3. Сформировать и выполнить запрос, оформив его в виде просмотра: *“В каком месяце была продана самая дорогая из старых машин”*.
4. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Машины какой страны пользуются популярностью у молодежи”*.
5. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить сумму продаж за каждый месяц”*.
6. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Определить, какая возрастная группа покупает в среднем самые дорогие автомобили”*.
7. Сформировать и выполнить запрос, оформив его в виде просмотра: *“Какая группа по социальному положению предпочитает при расчете кредитные карточки”*.