МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГТУ», ВГТУ)

Факультет информационных технологий и компьютерной безопасности

Кафедра компьютерных интеллектуальных технологий проектирования

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

По дисциплине: «Системы хранения и обработки данных»

Тема: «Разработка логической структуры базы данных»

Выполнил работу студент группы змИИВТ-241: Авсянкина М.А.

подпись, дата

Принял: Короленко В.В.

подпись, дата

Воронеж 2024

Цель работы: изучить основы логического проектирования базы данных, освоить процесс разработки логической структуры базы данных и построения диаграммы «сущность-связь».

Основные задачи:

• определение сущности для проекта в соответствии с индивидуальным заданием и их атрибуты;

• выделение ключевых атрибутов;

• определение связей между сущностями и типов связей;

• построение диаграммы сущность-связь для отображения логической структуры базы данных.

Ход работы:

База данных – это совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных‍. Она представляет собой актуальное состояние некоторой предметной области и используется для удовлетворения информационных потребностей пользователей‍‍. База данных является набором перманентных данных, которые логически структурированы для обеспечения возможности их эффективного поиска и обработки в вычислительной системе‍‍.

Основные понятия теории баз данных:

1. База данных – организованная совокупность данных, хранящаяся на компьютере таким образом, чтобы она могла быть легко доступна и управляема.
2. Система управления базами данных (СУБД) – программное обеспечение, которое управляет созданием, поддержкой и использованием баз данных.
3. Таблица – структура хранения данных, состоящая из строк (записей) и столбцов (полей). Каждая таблица имеет уникальное имя и набор полей с определенными типами данных.
4. Запись (строка) – одна единица данных в таблице, представляющая собой набор значений всех полей этой строки.
5. Поле (колонка) – элемент данных внутри таблицы, который описывает определенный аспект сущности (например, имя, адрес).
6. Тип данных – определяет формат данных, которые могут содержаться в конкретном поле (например, числовой, текстовый, дата/время).
7. Первичный ключ (Primary Key) – одно или несколько полей, которые однозначно идентифицируют каждую запись в таблице. Первичные ключи уникальны и не могут содержать NULL-значений.
8. Внешний ключ (Foreign Key) – поле или группа полей, ссылающихся на первичный ключ другой таблицы. Внешние ключи обеспечивают связь между таблицами.
9. Нормализация – процесс организации данных в базе данных таким образом, чтобы минимизировать дублирование данных и избежать аномалий при вставке, удалении и обновлении данных.
10. Транзакции – последовательность операций над базой данных, которая либо выполняется полностью, либо отменяется целиком, обеспечивая целостность данных.
11. Индексы – структуры данных, используемые для ускорения поиска и сортировки данных в таблицах.

Основные модели данных:

1. Реляционная модель данных:

Самая популярная модель данных, основанная на математической теории множеств и отношений. Данные организованы в виде таблиц, каждая из которых представляет собой отношение (entity), а строки и столбцы представляют записи и поля соответственно.

Примеры СУБД: MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

1. Иерархическая модель данных:

Представляет данные в виде древовидной структуры, где каждый узел может иметь одного родителя и множество потомков.

Применялась в ранних системах управления базами данных, таких как IMS (Information Management System) от IBM.

1. Сетевая модель данных:

Развитие иерархической модели, позволяющее узлам иметь более одного родителя. Была разработана для преодоления ограничений иерархической модели, однако сейчас используется редко.

1. Объектно-ориентированная модель данных:

Основывается на принципах объектно-ориентированного программирования, где данные представлены в виде объектов с методами и свойствами.

Используется в специализированных СУБД, таких как ObjectStore и Versant.

1. Документо-ориентированная модель данных:

Хранит данные в виде документов, обычно в формате JSON или XML.

Пример: MongoDB, Couchbase.

1. Графовая модель данных:

Представление данных в виде графов, где узлы связаны ребрами. Подходит для задач, требующих анализа связей между объектами, например, социальных сетей.

Пример: Neo4j.

Draw.io – это бесплатное онлайн-приложение для создания диаграмм, блок-схем, UML-диаграмм, ER-диаграмм и других типов визуальных представлений данных. Оно предоставляет интуитивный интерфейс и широкий спектр инструментов для создания профессиональных диаграмм.

Основные возможности Draw.io:

Разнообразие шаблонов: начиная от простых блок-схем до сложных сетевых топологий и UML-диаграмм.

Интеграции с облачными сервисами: Google Drive, Dropbox, OneDrive и другими.

Экспорт в различные форматы: PNG, JPEG, SVG, PDF и другие.

Поддержка совместной работы: возможность одновременной работы над одним проектом несколькими пользователями.

Расширяемость: поддержка плагинов и расширений для дополнительных функций.

На основании базы данных, согласно индивидуальному заданию:

База данных «Платный прием в поликлинике»

• Предлагаемый набор базовых таблиц:

1. ВРАЧИ

2. ПАЦИЕНТЫ

3. ПРИЕМ ПАЦИЕНТОВ

• Минимальный набор полей базовых таблиц

1. ФИО врача

2. Специальность врача

3. Стоимость приема

4. Процент отчисления на зарплату

5. Фамилия пациента

6. Имя пациента

7. Отчество пациента

8. Дата рождения пациента

9. Адрес пациента

10. Дата приема

• Описание предметной области

Платный прием пациентов проводится врачами разных специальностей (хирург, терапевт, кардиолог, офтальмолог и т.д.). При оформлении приема должна быть сформирована квитанция об оплате приема, в которой указывается информация о пациенте, о враче, который консультирует пациента, о стоимости приема, о дате приема.

Пациент оплачивает за прием некоторую сумму, которая устанавливается персонально для каждого врача. За каждый прием врачу отчисляется фиксированный процент от стоимости приема. Процент отчисления от стоимости приема на зарплату врача также устанавливается персонально для каждого врача.

Размер начисляемой врачу заработной платы за каждый прием вычисляется по формуле: Зарплата = Стоимость приема х Процент отчисления на зарплату. Из этой суммы вычитается подоходный налог, составляющий 13% от начисленной зарплаты.

Определим основные сущности для разрабатываемой базы данных:

Врачи – оказывают медицинские услуги пациентам, поэтому они являются одной из ключевых сущностей системы.

Пациенты – обращаются за медицинскими услугами, поэтому эта сущность тоже является важной частью системы.

Приемы – представляет собой конкретную встречу врача с пациентом, где оказываются определенные медицинские услуги. Эта сущность необходима для учета всех взаимодействий между врачами и пациентами.

Платежи – фиксируют оплату пациентов за оказанные им медицинские услуги. Это важная часть системы для финансового учета.

Зарплаты врачей – необходимы для расчета заработной платы каждого врача на основании проведенных ими приемов и других факторов.

Ассоциативная сущность "Зарплата врачей - Приемы" (зарплата\_врачей\_приемов) – используется для реализации связи "многие ко многим" между сущностями "Зарплаты врачей" и "Приемы". Она помогает точно определять, какие приемы влияют на зарплату какого врача.

Атрибуты сущностей:

Сущность "Врачи":

id\_врача: уникальный идентификатор врача.

фамилия: фамилия врача.

имя: имя врача.

отчество: отчество врача.

специальность: медицинская специализация врача.

стоимость\_приема: базовая стоимость одного приема у данного врача.

процент\_отчислений: процент, который удерживается с приема врача для расчетов зарплаты.

Сущность "Пациенты":

id\_пациента: уникальный идентификатор пациента.

фамилия: фамилия пациента.

имя: имя пациента.

отчество: отчество пациента.

дата\_рождения: дата рождения пациента.

адрес: адрес проживания пациента.

Сущность "Приемы":

id\_приема: уникальный идентификатор приема.

id\_врача: ссылка на врача, проводящего прием.

id\_пациента: ссылка на пациента, проходящего прием.

дата\_приема: дата проведения приема.

стоимость\_приема: общая стоимость приема, которая может включать дополнительные услуги.

Сущность "Платежи":

id\_платежа: уникальный идентификатор платежа.

id\_приема: ссылка на прием, за который производится платеж.

сумма: сумма платежа.

дата\_оплаты: дата совершения платежа.

Сущность "Зарплаты врачей":

id\_начисления: уникальный идентификатор начисления зарплаты.

id\_врача: ссылка на врача, которому начисляется зарплата.

брутто\_зарплата: полная заработная плата до вычетов налогов и отчислений.

нетто\_зарплата: чистая заработная плата после всех вычетов.

Ассоциативная сущность "Зарплата врачей - Приемы":

id\_врача: ссылка на врача.

id\_приема: ссылка на прием.

PRIMARY KEY (id\_врача, id\_приема): составной первичный ключ, гарантирующий уникальность каждой комбинации врача и приема.

Обоснование выбора

Врачи: Необходимо отслеживать информацию о каждом враче, включая его ФИО, специализацию и стоимость приема, чтобы эффективно управлять персоналом клиники.

Пациенты: Для ведения медицинской документации необходимо собирать данные о каждом пациенте, такие как ФИО, дата рождения и адрес.

Приемы: Эта сущность нужна для фиксации конкретных встреч врачей с пациентами, что важно для учета времени работы врачей и предоставляемых услуг.

Платежи: Фиксация платежей необходима для учета финансовых операций и контроля за оплатой медицинских услуг.

Зарплаты врачей: Требуется для расчета и выплаты зарплат врачам на основании их трудовой деятельности и установленных процентов отчислений.

Ассоциативная сущность "Зарплата врачей - Приемы": Необходима для точного учета того, какие приемы участвуют в расчете зарплаты каждого врача.

Эти сущности и их атрибуты обеспечивают полную функциональность системы управления медицинским учреждением, позволяя вести учет персонала, пациентов, оказываемых услуг и финансовых операций.

Связи между сущностями

Врачи – Приемы:

Один врач может проводить множество приемов, но каждый прием проводится одним врачом. Это отношение называется "один ко многим".

В таблице приемы есть внешний ключ id\_врача, который ссылается на первичный ключ id\_врача в таблице врачи.

Пациенты – Приемы:

Один пациент может посещать множество приемов, но каждый прием посещает один пациент. Это также отношение "один ко многим".

В таблице приемы есть внешний ключ id\_пациента, который ссылается на первичный ключ id\_пациента в таблице пациенты.

Приемы – Платежи:

Каждый прием может иметь один или несколько платежей, но каждый платеж относится к одному приему. Это отношение "один ко многим".

В таблице платежи есть внешний ключ id\_приема, который ссылается на первичный ключ id\_приема в таблице приемы.

Зарплаты врачей – Врачи:

Каждая запись о зарплате относится к одному врачу, но один врач может иметь много записей о зарплате. Это отношение "один ко многим".

В таблице зарплаты\_врачей есть внешний ключ id\_врача, который ссылается на первичный ключ id\_врача в таблице врачи.

Зарплаты врачей – Приемы:

Записи о зарплате могут относиться к нескольким приемам, и каждый прием может участвовать в нескольких начислениях зарплаты. Это отношение "многие ко многим".

Для реализации этого отношения была создана промежуточная таблица зарплата\_врачей\_приемов, которая содержит внешние ключи id\_врача и id\_приема, ссылающиеся соответственно на таблицы врачи и приемы.

Таким образом, связи между сущностями можно представить следующим образом:

Врачи → Приемы (один ко многим)

Пациенты → Приемы (один ко многим)

Приемы → Платежи (один ко многим)

Врачи → Зарплаты врачей (один ко многим)

Зарплаты врачей ↔ Приемы (многие ко многим через промежуточную таблицу)

Чтобы избавиться от связи «многие ко многим» в таблице зарплаты\_врачей, мы можем упростить структуру базы данных, убрав промежуточную таблицу-связку и оставив прямую связь между таблицами врачи и приемы.

Ниже приведены измененные SQL-запросы, убраны лишние элементы.

Сущность Врачи:

* id\_врача
* фамилия
* имя
* отчество
* специальность
* стоимость\_приема
* процент\_отчислений

Сущность Пациенты:

* id\_пациента
* фамилия
* имя
* отчество
* дата\_рождения
* адрес

Сущность Приемы:

* id\_приема
* id\_врача
* id\_пациента
* дата\_приема
* стоимость\_приема

Сущность Платежи:

* id\_платежа
* id\_приема
* сумма
* дата\_оплаты

Сущность Зарплаты врачей:

* id\_начисления
* id\_врача
* id\_приема
* брутто\_зарплата
* нетто\_зарплата

Мы удалили таблицу зарплата\_врачи\_приемы, поскольку она больше не нужна для обеспечения связи «многие ко многим». В новой версии таблицы зарплаты\_врачей остались поля id\_врача и id\_приема, которые напрямую ссылаются на соответствующие таблицы. Таким образом, каждая запись в таблице зарплаты\_врачей соответствует конкретному врачу и приему.

Построим диаграмму для отображения нашей созданной логической структуры с использованием средства для разработки диаграмм draw.io.

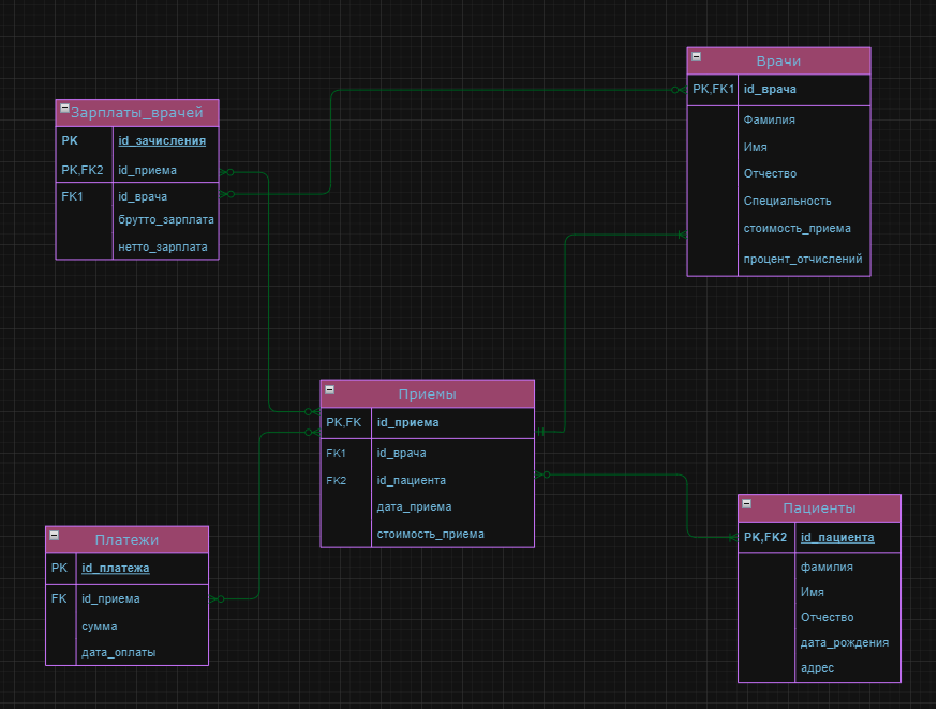


Рисунок 1 – Диаграмма отображения логической структуры базы данных

Далее выделим ключевые атрибуты ((PK, FK), и обозначим связи между нашими сущностями (рисунок 2).

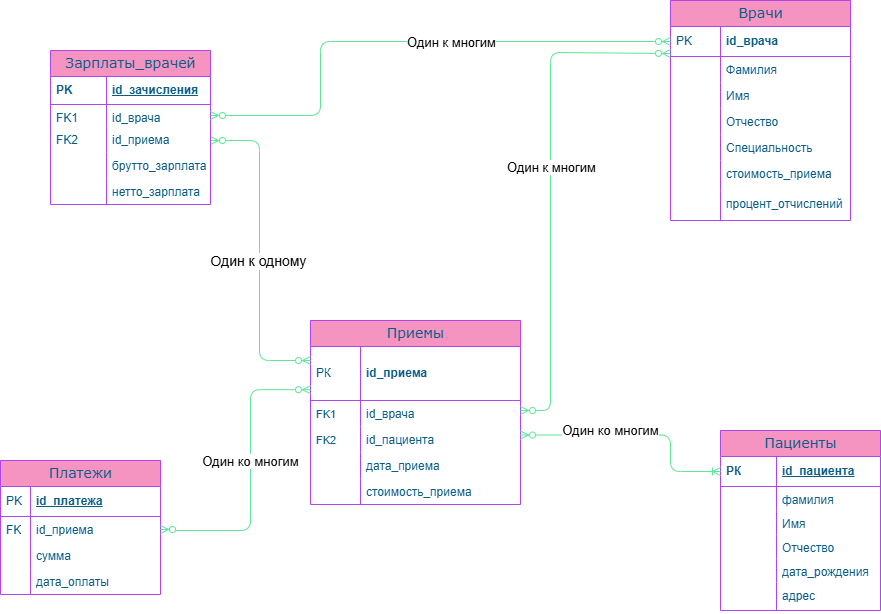


Рисунок 2 – Выделение ключевых атрибутов и обозначения связей

Диаграмма предназначена для медицинской платной клинки, учета работы врачей, пациентов, приема пациентов, оплаты за услуги и расчета зарплат врачам. Удобна и проста в использовании для неопытных пользователем, не перегружена. Она состоит из пяти сущностей:

Сущность «врачи» – содержит информацию о врачах клиники. В ней хранятся данные об их фамилии, имени, отчестве, специальности, стоимости одного приема и проценте отчислений, который начисляется с каждого приема врача.

Атрибуты:

id\_врача: уникальный идентификатор врача, первичный ключ таблицы.

фамилия: обязательный атрибут, содержащий фамилию врача.

имя: необязательный атрибут, содержащее имя врача.

отчество: необязательный атрибут, содержащее отчество врача.

специальность: обязательная строковая информация о специализации врача.

стоимость\_приема: фиксированная стоимость одного приема у данного врача.

процент\_отчислений: процент, который отчисляется непосредственно самому врачу после каждого его приема. Ограничен значениями от 0 до 100%.

Выборка сущностей и атрибутов: В этой таблице сосредоточены ключевые сведения о каждом враче, необходимые для ведения медицинской деятельности и финансового учета. Специальность позволяет классифицировать врачей по направлениям медицины, а стоимость приема помогает корректно рассчитывать оплату услуг.

Сущность «пациенты» – хранит информацию о пациентах, обращающихся к врачам клиники. Включает ФИО, дату рождения и адрес проживания.

Атрибуты:

id\_пациента: уникальный идентификатор пациента, первичный ключ таблицы.

фамилия: обязательный атрибут, содержащая фамилию пациента.

имя: необязательное имя пациента.

отчество: необязательное отчество пациента.

дата\_рождения: дата рождения пациента.

адрес: строковый атрибут, хранящий адрес проживания пациента.

Выборка сущностей и атрибутов: Эти данные необходимы для идентификации пациентов и их последующего отслеживания в системе. Дата рождения может использоваться для различных целей, таких как анализ возрастных групп пациентов, рассылке с напоминаем об проверке их организма, а также для предоставления персонализированных предложений и напоминаний.

Сущность «приемы» – фиксирует каждый прием пациента врачом. Здесь хранится информация о дате приема, пациенте, враче и стоимости этого приема.

Атрибуты:

id\_приема: уникальный идентификатор приема, первичный ключ таблицы.

id\_врача: внешний ключ, ссылающийся на таблицу врачи, указывает на конкретного врача, проводившего прием.

id\_пациента: внешний ключ, ссылающийся на таблицу пациенты, указывает на пациента, пришедшего на прием.

дата\_приема: обязательная дата проведения приема.

стоимость\_приема: стоимость проведенного приема.

Выборка сущностей и атрибутов: Этот блок данных важен для фиксации факта оказания медицинских услуг, а также для дальнейшего анализа загруженности врачей и общей статистики посещений. Стоимость приема необходима для расчетов платежей и начисления заработных плат врачам.

Сущность «платежи» – учитывает все платежи, произведенные пациентами за медицинские услуги. Записи содержат сумму платежа, дату оплаты и ссылку на конкретный прием.

Атрибуты:

id\_платежа: уникальный идентификатор платежа, первичный ключ таблицы.

id\_приема: внешний ключ, ссылающийся на таблицу приемы, указывает на прием, за который был произведен платеж.

сумма: общая сумма платежа.

дата\_оплаты: дата и время совершения платежа, автоматически заполняется текущими данными сервера (CURRENT\_TIMESTAMP), если не указана вручную.

Выборка сущностей и атрибутов: Платежи являются важной частью финансовой отчетности и помогают отслеживать поступления средств в клинику. Ссылки на конкретные приемы позволяют сопоставлять доходы с оказанными услугами.

Сущность «зарплаты\_врачей» – используется для хранения информации о начисленной зарплате каждому врачу после каждого приема. Зарплата рассчитывается исходя из стоимости приема и процента отчислений.

Атрибуты:

id\_начисления: уникальный идентификатор записи о начислении зарплаты, первичный ключ таблицы.

id\_врача: внешний ключ, ссылающийся на таблицу врачи, указывает на врача, которому начисляется зарплата.

id\_приема: внешний ключ, ссылающийся на таблицу приемы, указывает на прием, за который производится расчет зарплаты.

брутто\_зарплата: начисление за каждый прием врачу, до вычета налогов.

нетто\_зарплата: отчисления за прием врачу, после вычета подоходного налога.

Выборка сущностей и атрибутов: Расчет зарплаты врачам является важным аспектом бухгалтерского учета в клинике. Эта таблица связывает информацию о приеме и враче с финансовыми показателями, такими как брутто\_зарплата и нетто\_ зарплата после удержаний.

Связи между таблицами

Между таблицей врачи и таблицей приемы: Внешний ключ id\_врача в таблице приемы ссылается на id\_врача в таблице врачи. Это связь типа «один ко многим», так как один врач может проводить множество приемов.

Между таблицей пациенты и таблицей приемы: Внешний ключ id\_пациента в таблице приемы ссылается на id\_пациента в таблице пациенты. Это тоже связь типа «один ко многим», поскольку один пациент может посещать несколько приемов.

Между таблицей приемы и таблицей платежи: Внешний ключ id\_приема в таблице платежи ссылается на id\_приема в таблице приемы. Один прием может иметь несколько связанных платежей (например, оплата частями), поэтому эта связь также относится к типу «один ко многим».

Между таблицей врачи и таблицей зарплаты\_врачей: Внешний ключ id\_врача в таблице зарплаты\_врачей ссылается на id\_врача в таблице врачи. Аналогично предыдущему пункту, один врач может получать зарплату за несколько приемов, следовательно, это связь типа «один ко многим».

Между таблицей приемы и таблицей зарплаты\_врачей:Внешний ключ id\_приема в таблице зарплаты\_врачей ссылается на id\_приема в таблице приемы. Одна запись о приеме может генерировать одну запись о зарплате, то есть связь типа «один к одному».

Первичные ключи используются для уникальной идентификации каждой строки в таблице. Они должны быть уникальными и неизменными для каждой записи.

В данной базе данных первичными ключами являются:

id\_врача в таблице врачи – уникальный идентификатор врача. Поскольку каждый врач должен быть уникальным объектом в системе, этот ключ гарантирует отсутствие дубликатов и упрощает поиск конкретной записи о враче.

id\_пациента в таблице пациенты – идентификационный номер пациента. Каждый пациент должен иметь свой уникальный идентификатор для точного учета обращений и истории посещений.

id\_приема в таблице приемы – номер приема. Каждое посещение должно быть уникальным событием, и этот ключ позволяет однозначно идентифицировать каждое посещение.

id\_платежа в таблице платежи – идентификатор платежа. Платежи могут поступать разными суммами и в разное время, поэтому каждая транзакция должна иметь свою уникальную метку.

id\_начисления в таблице зарплаты\_врачей – идентификатор начисления зарплаты. Этот ключ нужен для того, чтобы учитывать каждую выплату отдельно, даже если они происходят по одним и тем же приемам или врачам.

Внешние ключи (FOREIGN KEY)

Внешние ключи используются для установления связи между двумя таблицами, обеспечивая ссылочную целостность данных. В данном случае внешние ключи указывают на записи в родительских таблицах:

id\_врача в таблице приемы – указывает на конкретную запись в таблице врачи, определяющую врача, который проводил прием. Это необходимо для связи конкретных приемов с определенными врачами.

id\_пациента в таблице приемы – указывает на конкретную запись в таблице пациенты, определяющую пациента, который посетил врача. Это важно для отслеживания истории посещений каждого пациента.

id\_приема в таблице платежи – указывает на конкретную запись в таблице приемы, определяющую прием, за который осуществляется платеж. Это нужно для того, чтобы точно знать, какой прием оплачивается.

id\_врача в таблице зарплаты\_врачей – указывает на конкретную запись в таблице врачи, определяющую врача, которому начислена зарплата. Это требуется для привязки начисленных выплат к конкретным врачам.

id\_приема в таблице зарплаты\_врачей – указывает на конкретную запись в таблице приемы, определяющую прием, за который начисляется зарплата. Это помогает точно определить, за какие посещения начисляются выплаты.

Диаграмма построена таким образом, чтобы обеспечить гибкость и удобство работы с информацией о врачах, пациентах, проводимых приемах, платежах и расчетах зарплат. Использование внешних ключей позволяет эффективно управлять взаимосвязями между различными объектами системы и обеспечивает целостность данных.

Вывод: таким образом, мы изучили основы логического проектирования базы данных, освоили процесс разработки логической структуры базы данных и построения диаграммы «сущность-связь». Так же научились определять сущности для нашего проекта в соответствии с индивидуальным заданием и их атрибуты. Освоили выделение основных ключевых атрибутов. Разобрались, как определять связей между сущностями и типов связей.

1. Дайте определения следующим понятиям: данные, база данных, СУБД, ведение базы данных.

Данные – это совокупность фактов, событий, объектов или других элементов, представленных в формализованном виде для последующей обработки компьютерными системами.

База данных (БД) – Организованная структура для хранения данных таким образом, чтобы они могли быть легко доступны, управляться и обновляться.

Система управления базами данных (СУБД) – Программное обеспечение, предназначенное для создания, управления и обеспечения доступа к базе данных.

Ведение базы данных – Процесс управления базой данных, включающий создание, обновление, удаление, поиск и анализ данных.

2. В чем отличие данных от информации?

Данные – это сырые факты, цифры или символы, которые сами по себе могут не иметь смысла до тех пор, пока они не будут обработаны и интерпретированы. Информация, напротив, представляет собой осмысленные и полезные выводы, полученные после анализа данных.

3. В чем отличие базы данных от банка данных и СУБД?

Банк данных – это комплексная система, состоящая из одной или нескольких баз данных, а также программного обеспечения для управления этими данными (т.е., СУБД). Банк данных может включать несколько взаимосвязанных баз данных и обеспечивать централизованное управление ими.

База данных – это конкретная структурированная коллекция данных, организованных определенным образом для облегчения поиска и извлечения.

СУБД – это программное обеспечение, которое используется для управления базами данных, включая создание, модификацию, хранение и извлечение данных.

4. Назовите основные компоненты банка данных и их назначение.

Основные компоненты банка данных включают:

Базы данных: Хранит структурированные данные.

СУБД: Управляет доступом к данным, обеспечивает целостность и безопасность данных.

Средства администрирования: Инструменты для управления базой данных, такие как резервное копирование, восстановление, мониторинг производительности.

Приложения пользователей: Интерфейсы, через которые пользователи взаимодействуют с банком данных для выполнения различных задач.

5. Классифицируйте АИС по типу хранимых данных.

Автоматизированные информационные системы (АИС) классифицируются по типу хранимых данных следующим образом:

Документальные системы: Хранение текстовых документов, изображений, аудио- и видео-файлов.

Фактографические системы: Хранение структурированных данных о фактах и событиях (например, базы данных).

Экспертные системы: Хранение знаний и правил для решения сложных задач.

6. Что понимается под трехуровневой архитектурой ANSI/SPARC?

Трехуровневая архитектура ANSI/SPARC описывает три уровня представления данных в системе управления базами данных:

Внешний уровень: Представление данных для конечных пользователей. Включает множество внешних схем, каждая из которых отражает взгляд конкретного пользователя на базу данных.

Концептуальный уровень: Общее представление всей базы данных, объединяющее все внешние схемы. Определяет логические структуры данных и связи между ними.

Внутренний уровень: Физическое представление данных, определяющее, как именно данные хранятся на физическом носителе.

7. Дайте определения внешней схеме БД, концептуальной схеме БД, внутренней схеме БД.

Внешняя схема БД: Представление данных, ориентированное на конкретные потребности отдельных групп пользователей или приложений. Внешние схемы могут различаться для разных пользователей.

Концептуальная схема БД: Логическая модель всех данных в базе данных, которая объединяет все внешние схемы и определяет структуру данных и отношения между ними без учета особенностей физического хранения.

Внутренняя схема БД: Описание того, как данные физически организованы и хранятся на носителях. Внутренняя схема включает в себя информацию о индексах, методах хранения и доступе к данным.

8. Каковы особенности иерархической модели организации данных?

Иерархическая модель данных организована в виде дерева, где каждый узел имеет одного родителя и может иметь любое количество дочерних узлов. Особенности этой модели:

Данные представлены в виде древовидной структуры.

Каждый элемент может иметь только одного предка, но может иметь много потомков.

Простота навигации по структуре данных сверху вниз.

Ограниченная гибкость при необходимости изменения структуры данных.

9. Каковы особенности сетевой модели организации данных?

Сетевая модель данных позволяет создавать сложные сети связей между элементами данных. Основные характеристики:

Элементы данных могут иметь произвольное число предков и потомков.

Гибкая структура, позволяющая моделировать сложные реальные объекты и процессы.

Сложность реализации и поддержки, особенно при изменении структуры данных.

10. Каковы особенности многомерной модели организации данных?

Многомерная модель данных предназначена для аналитических систем и хранилищ данных (OLAP-систем). Ее ключевые особенности:

Данные организованы в виде гиперкуба, который состоит из измерений и мер.

Позволяет выполнять сложные запросы и агрегаты данных по различным измерениям.

Высокая производительность при выполнении запросов, связанных с агрегацией данных.

11. Каковы особенности постреляционной модели организации данных?

Постреляционная модель является расширением реляционной модели и характеризуется следующими особенностями:

Поддерживает объектно-реляционные возможности, позволяя хранить сложные типы данных.

Объекты могут содержать методы и процедуры, что делает эту модель похожей на объектно-ориентированную.

Возможность работы с наследованием и полиморфизмом.

12. Что относится к неструктурированным данным?

Неструктурированные данные – это данные, которые не имеют заранее определенной структуры или формата. Примеры таких данных:

Текстовые документы.

Электронные письма.

Веб-страницы.

Изображения, аудио- и видео-файлы.

Эти данные трудно обрабатывать традиционными методами, такими как SQL-запросы, поэтому для их анализа используются специальные инструменты и подходы.

13. В чем преимущество использования колоночной СУБД по сравнению с реляционной?

Колончатые СУБД оптимизируют хранение данных по столбцам, что дает следующие преимущества перед реляционными СУБД:

Более высокая скорость чтения данных при выполнении операций выборки по нескольким столбцам.

Экономия места за счет сжатия данных по столбцам.

Улучшенная производительность при работе с большими объемами данных.

14. Каким образом осуществляется связь между таблицами в реляционной СУБД?

Связь между таблицами в реляционных СУБД осуществляется посредством ключей:

Первичный ключ (Primary Key) – уникальный идентификатор записи в таблице.

Внешний ключ (Foreign Key) – ссылка на первичный ключ другой таблицы, обеспечивающая связь между двумя таблицами.

Эта связь называется отношением "один ко многим" или "многие ко многим".

15. Каким образом на этапе проектирования решается проблема дублирующих записей в таблице?

Для предотвращения дублирования записей в таблице применяются следующие меры:

Использование уникальных индексов или ограничений уникальности (UNIQUE), которые гарантируют, что каждое значение в определенном столбце будет уникальным.

Применение нормальных форм, которые помогают устранить избыточность данных и минимизировать возможность появления дубликатов.

Регулярная проверка и очистка данных от возможных дублей.