**1. Необходимость проектирования баз данных, цели проектирования, этапы проектирования**

Проектирование базы данных необходимо для того, чтобы информация в ней хранилась структурированно, последовательно и без избыточности, а также чтобы обеспечивалась целостность, удобство поиска и обработки данных.

Цели проектирования — организовать данные так, чтобы минимизировать избыточность, исключить противоречия и обеспечить логические связи между данными.

 Этапы проектирования:

1. анализ предметной области;
2. определение сущностей и связей;
3. создание концептуальной модели;
4. преобразование её в логическую (обычно реляционную) модель;
5. реализация физической структуры (таблицы, индексы);
6. тестирование и оптимизация.

**2. Основные понятия реляционной базы данных: сущность, атрибут, ключ, запись, связь**

Сущность — это объект реального мира, информацию о котором требуется хранить (например, «студент» или «книга»).  
 Атрибут — это характеристика или свойство сущности (например, имя, возраст, цена).  
 Ключ — это один или несколько атрибутов, однозначно определяющих запись в таблице.  
 Запись (строка) — это конкретный набор значений атрибутов, описывающих одну сущность.  
 Связь — это логическая ассоциация между сущностями, отражающая их взаимодействие (например, «студент сдаёт экзамен»).

**3. Виды моделей данных. Иерархическая и сетевая модели. Достоинства и недостатки**

Иерархическая модель представляет данные в виде древовидной структуры, где каждая запись имеет одну родительскую и множество дочерних.

 Плюсы: простота структуры, высокая скорость доступа.  
 Минусы: неудобство при сложных связях и ограниченность в гибкости.

Сетевая модель устроена как граф, где одна запись может иметь несколько родителей.

 Плюсы: поддержка сложных связей, высокая производительность.  
 Минусы: сложность структуры и работы с ней, сложность программирования.

**4. Виды моделей данных. Реляционная модель данных, основные понятия и элементы**

Реляционная модель представляет данные в виде таблиц (отношений), где строки — это записи, а столбцы — атрибуты.

 Основные элементы:

 — отношения (таблицы);  
 — атрибуты (столбцы);  
 — кортежи (строки);  
 — ключи (уникальные идентификаторы записей);  
 — связи между таблицами.

 Эта модель проста в использовании, легко масштабируется и обеспечивает логическую целостность данных.

**5. Операции реляционной алгебры: проекция, выборка, соединение, объединение, пересечение, вычитание, умножение**

Проекция — получение подмножества столбцов таблицы.

 Выборка — получение строк, удовлетворяющих заданному условию.

 Соединение — объединение таблиц по общим атрибутам.

 Объединение — объединение всех записей двух таблиц, без повторов.

 Пересечение — получение записей, присутствующих в обеих таблицах.

 Вычитание — получение записей, которые есть в одной таблице, но отсутствуют в другой.

 Умножение (декартово произведение) — получение всех возможных пар записей из двух таблиц.

**6. Понятие ключа отношения. Виды ключей. Правила выбора ключа**

В **реляционной модели данных** *ключ* — это такой атрибут (или комбинация атрибутов), который **однозначно идентифицирует каждую строку (кортеж)** в таблице (отношении). Ключ предотвращает дублирование данных и служит основой для организации связей между таблицами.

**Виды ключей:**

* **Первичный ключ (Primary Key)** — основной уникальный идентификатор записи. В таблице может быть только один первичный ключ, и его значение не должно быть NULL. Пример: passport\_number в таблице citizens.
* **Альтернативный ключ (Candidate Key)** — любой ключ, который может быть использован как первичный, но не выбран в этой роли. Все альтернативные ключи также уникальны.
* **Внешний ключ (Foreign Key)** — атрибут, который **ссылается** на первичный ключ **другой таблицы**. Он обеспечивает **связь между таблицами** и поддерживает ссылочную целостность. Пример: department\_id в таблице employees, ссылающийся на departments.id.
* **Составной ключ (Composite Key)** — ключ, состоящий из **нескольких атрибутов**, вместе образующих уникальное значение. Часто используется в таблицах связей (например, student\_id + course\_id в enrollments).

**Правила выбора ключа:**

* Ключ должен быть **уникальным** для каждой строки.
* Должен быть **минимальным** (не содержать избыточных атрибутов).
* Должен быть **неизменяемым** (не должен часто обновляться).
* Желательно использовать **искусственный (суррогатный) ключ** (например, ID), если нет подходящего естественного уникального атрибута.

**7. Понятие функциональной зависимости атрибутов. Виды зависимостей**

**Функциональная зависимость** — это логическая связь между атрибутами в таблице: если значению атрибута A однозначно соответствует значение атрибута B, говорят, что **B функционально зависит от A**, и записывают: **A → B**

Это означает, что если два кортежа имеют одинаковое значение атрибута A, то у них обязательно одинаковое значение B.

**Виды функциональных зависимостей:**

* **Полная функциональная зависимость** — атрибут зависит **от всего составного ключа**, а не от его части. Это важное условие для второй нормальной формы.  
   Пример: student\_id + course\_id → grade
* **Частичная зависимость** — атрибут зависит **только от части составного ключа**, а не от всего ключа целиком.  
   Пример: если student\_id + course\_id — ключ, но student\_name зависит только от student\_id, значит зависимость частичная. Такие зависимости устраняются при переходе к 2НФ.
* **Транзитивная зависимость** — атрибут зависит от **другого атрибута, который сам зависит от ключа**.  
   Пример: student\_id → group\_id и group\_id → group\_name ⇒ student\_id → group\_name. Это признак нарушения 3НФ.

**8. Понятие нормализации базы данных. Нормальные формы. Требования 1НФ, 2НФ и 3НФ**

**Нормализация** — это процесс **структурирования данных** в реляционной базе, с целью устранения избыточности и обеспечения целостности. Данные преобразуются в таблицы, удовлетворяющие определённым **нормальным формам**.

**Нормальные формы (НФ):**

* Каждая следующая форма основывается на предыдущей и добавляет новые требования.

Первая нормальная форма (1НФ):

* Все поля содержат только **атомарные значения** (неделимые).
* Нет множественных или повторяющихся групп данных.

Пример нарушения: поле phone\_numbers = "123, 456, 789" (несколько значений в одном поле). Нужно вынести телефоны в отдельную таблицу.

Вторая нормальная форма (2НФ):

* Таблица уже в 1НФ.
* Нет **частичных зависимостей** неключевых атрибутов от **составного** ключа.

То есть все неключевые поля должны зависеть от **всего** ключа, а не от его части. Частичные зависимости выносятся в отдельные таблицы.

Третья нормальная форма (3НФ):

* Таблица уже в 2НФ.
* Нет **транзитивных зависимостей** между неключевыми атрибутами.

Все неключевые атрибуты должны зависеть **только от ключа**, а не друг от друга. Транзитивные зависимости устраняются путём создания новых таблиц.

**9. Понятие связи между сущностями. Характеристики связи**

Связь — это логическая ассоциация между двумя или более сущностями, отражающая их взаимодействие или зависимость.

 Характеристики связи:

 — **степень** (число участвующих сущностей, чаще всего две);

 — **тип связи**:

 • один-к-одному (1:1);

 • один-ко-многим (1:N);

 • многие-ко-многим (M:N);

 — **обязательность** (обязательно ли сущности должны участвовать в связи);

 — **роль** (показывает назначение сущности в связи).

**10. Приведение связи типа «многие-ко-многим» к типу «один-ко-многим»**

Связь **"многие-ко-многим" (many-to-many)** между двумя сущностями в реляционной базе данных невозможно реализовать напрямую в виде двух таблиц, поскольку каждая строка одной таблицы может соответствовать нескольким строкам другой, и наоборот. Реляционные СУБД не поддерживают такую связь напрямую, поэтому её **необходимо преобразовать в две связи "один-ко-многим"** через **промежуточную таблицу (таблицу-связку)**.

Суть преобразования:

Допустим, есть две сущности: Students (Студенты) и Courses (Курсы). Один студент может записаться на несколько курсов, и один курс может посещаться несколькими студентами. Это и есть классическая связь "многие-ко-многим".

Чтобы реализовать это в базе данных:

1. Создаётся **третья таблица** — например, Enrollments (Записи на курсы).
2. Она содержит **внешние ключи** на Students и Courses.
3. Теперь каждая запись в Enrollments означает, что конкретный студент записан на конкретный курс.

Теперь каждая связь "многие-ко-многим" заменяется двумя связями "один-ко-многим":

* Один студент может иметь **много записей** в Enrollments.
* Один курс может иметь **много записей** в Enrollments.

Таким образом, реализуется связь через третичную таблицу, и это обеспечивает корректную нормализацию данных.

**11. СУБД Access. Характеристики, применение. Состав базы данных в СУБД Access**

Microsoft Access — это настольная система управления базами данных (СУБД), предназначенная для создания небольших и средних по объёму баз данных.

 Характеристики: простота, наличие графического интерфейса, поддержка SQL, связь с другими продуктами Microsoft.

 Применяется в малом бизнесе, для учёта, каталогов, реестров.

 Состав базы данных в Access:

 — таблицы;  
 — запросы;  
 — формы для ввода и отображения данных;  
 — отчёты для печати;  
 — макросы;  
 — модули.

**12. Предложение SELECT языка SQL и его элементы**

Команда **SELECT** предназначена для выборки данных из таблиц.

 Основные элементы:  
 — **SELECT** — указание нужных столбцов;  
 — **FROM** — источник данных (таблица);  
 — **WHERE** — условие отбора строк;  
 — **ORDER BY** — сортировка результата;  
 — **GROUP BY** — группировка строк;  
 — **HAVING** — условие для групп.

**13. Виды запросов к базе данных и их реализация на языке SQL**

Запросы бывают:

 — **выборочные** (SELECT) — получение данных;  
 — **добавления** (INSERT) — добавление новых записей;  
 — **обновления** (UPDATE) — изменение существующих данных;  
 — **удаления** (DELETE) — удаление записей.  
 В SQL для этого используются одноимённые команды: **SELECT**, **INSERT**, **UPDATE**, **DELETE**.

**14. Подзапросы в языке SQL. Назначение, виды, порядок выполнения**

**Подзапрос (вложенный запрос)** — это SQL-запрос, помещённый внутрь другого запроса и обрабатываемый перед выполнением внешнего. Он позволяет получить промежуточные данные, необходимые для условия или значения основного запроса.

Назначение:

* Использование результата одного запроса как входных данных для другого.
* Выбор значений на основе более сложных условий.
* Формирование условий фильтрации, агрегации, обновления и т. д.

Виды подзапросов:

**Одиночные** — возвращают одно значение (например, одно число или одно поле). Используются в выражениях вроде =, <, >, !=. SELECT name FROM students WHERE age = (SELECT MAX(age) FROM students);

**Множественные** — возвращают набор значений. Используются с IN, ANY, ALL, EXISTS.

SELECT name FROM students WHERE group\_id IN (SELECT id FROM groups WHERE faculty = 'ФКН');

**Коррелированные** — выполняются **для каждой строки основного запроса**, то есть зависят от неё.

SELECT name FROM students s

WHERE EXISTS (

  SELECT \* FROM enrollments e WHERE e.student\_id = s.id AND e.course\_id = 101

);

Порядок выполнения:

Сначала выполняются **вложенные подзапросы**, затем их результат используется в основном запросе. В случае коррелированных подзапросов выполнение происходит построчно — для каждой строки основной таблицы.

**15. Запросы с параметром. Достоинства и недостатки**

**Параметрические запросы** — это шаблонные SQL-запросы, где значения передаются **в виде параметров** из программы, а не вставляются напрямую в строку запроса.

Достоинства:

* **Безопасность** — параметры обрабатываются отдельно от текста SQL, что предотвращает **SQL-инъекции** (одна из главных угроз в веб-приложениях).
* **Повторное использование** — один и тот же запрос можно выполнять многократно с разными значениями.
* **Удобство автоматизации** — особенно в приложениях с пользовательским вводом или запросами в цикле.

PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM users WHERE login = ?");

stmt.setString(1, userInput);

ResultSet rs = stmt.executeQuery();

Недостатки:

* Требуют **дополнительного программного кода** для подготовки запроса и передачи параметров.
* Иногда сложнее отлаживать, особенно при большом количестве параметров.
* Нельзя динамически менять структуру самого SQL — только значения.

**16. Целостность и сохранность баз данных. Виды целостности**

**Целостность данных**

Целостность — это способность базы данных **сохранять корректность, непротиворечивость и согласованность данных** в течение всего времени её использования. Это достигается за счёт соблюдения определённых правил и ограничений, которые накладываются на структуру и содержимое таблиц.

Виды целостности:

1. **Целостность сущности (Entity Integrity)** Каждая строка в таблице должна иметь уникальный **первичный ключ**.  
    Это означает, что:
   * Значение первичного ключа не может быть NULL;
   * Значения первичных ключей не должны повторяться. Это обеспечивает уникальность записей и возможность их однозначной идентификации.
2. **Ссылочная целостность (Referential Integrity)** Все **внешние ключи (foreign keys)** должны ссылаться на существующие записи в другой таблице. Например, если в таблице заказов указывается идентификатор клиента, то такой клиент **должен существовать** в таблице клиентов. Это предотвращает появление "висячих" ссылок на несуществующие объекты.
3. **Логическая (или доменная) целостность (Domain Integrity)** Атрибуты должны содержать **только допустимые значения**, соответствующие своему назначению. Это обеспечивается:
   * типами данных (INTEGER, DATE, VARCHAR, и т.д.),
   * ограничениями (CHECK, NOT NULL, DEFAULT, ENUM),
   * правилами и бизнес-логикой. Например, возраст не может быть отрицательным, дата окончания не может быть раньше даты начала.

**Сохранность данных**

**Сохранность (data durability, data protection)** — это способность базы данных **противостоять потерям данных** из-за внешних факторов: сбоев питания, ошибок программного обеспечения, сбоев оборудования, человеческого фактора и т. д.

Способы обеспечения сохранности:

* **Резервное копирование (бэкапы)** — регулярное создание копий данных, которые можно восстановить при сбоях;
* **Журнал транзакций** (transaction log) — записи всех операций, позволяющие откатить или восстановить изменения;
* **Системы репликации и кластеризации** — копирование данных на другие сервера;
* **UPS и RAID** — аппаратные методы защиты данных;
* **Транзакционность (ACID-свойства)** — гарантирует, что данные либо полностью записаны, либо не изменены вообще, если произошёл сбой.

Таким образом, целостность обеспечивает **логическую правильность** и непротиворечивость данных, а сохранность — **физическую надёжность** их хранения и защиту от потерь. Оба аспекта являются обязательными требованиями к надёжной и устойчивой системе баз данных.