**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**

**Модель данных — это способ организации, хранения и управления данными в базе. Иерархическая и сетевая модели относятся к традиционным (довереляционным) моделям, которые использовались до появления реляционной модели.**

**Иерархическая модель данных организует данные в виде дерева. Каждая запись (узел) связана только с одной родительской записью, но может иметь несколько дочерних. Верхняя запись называется корневой, а все остальные образуют иерархические уровни.**

**Достоинства:**

* **Простая и логичная структура для представления данных с естественной иерархией (например, организация, файл-система).**
* **Высокая производительность при доступе к данным по иерархическим путям.**
* **Чёткая структура упрощает навигацию и оптимизацию при чтении.**

**Недостатки:**

* **Невозможно выразить сложные связи: одна запись не может быть связана с двумя родителями.**
* **Сложность при изменении структуры (например, при необходимости переноса дочернего элемента).**
* **Жёсткая зависимость от структуры, трудности в моделировании реальных систем с перекрёстными связями.**

**Сетевая модель данных является обобщением иерархической: структура организована в виде графа, где запись может иметь несколько родителей и участвовать в различных связях. Записи соединяются через наборы (sets), определяющие отношения между ними.**

**Достоинства:**

* **Поддержка сложных связей между записями (например, многие-ко-многим).**
* **Высокая эффективность при выполнении операций с большими объёмами данных.**
* **Гибкость в моделировании разнообразных предметных областей.**

**Недостатки:**

* **Структура более сложная, чем у иерархической модели.**
* **Требуется глубокое понимание связей и их программная реализация.**
* **Трудности в сопровождении и изменении структуры, особенно в больших системах.**

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

**Реляционная модель данных была предложена Эдгаром Коддом в 1970 году и стала самой распространённой моделью для организации и управления базами данных. Она основана на представлении данных в виде отношений (таблиц), каждая из которых содержит строки и столбцы.**

**Основные элементы реляционной модели:**

* **Отношение (таблица) — двухмерная структура, представляющая один тип сущности (например, "студенты", "курсы").**
* **Атрибут (столбец) — характеристика сущности, например, имя или возраст.**
* **Строка (кортеж ) — одна запись, содержащая значения всех атрибутов (один объект).**
* **Ключи:**
  + **Первичный ключ (Primary Key) — уникальный идентификатор записи (например, ID).**
  + **Внешний ключ (Foreign Key) — ссылка на первичный ключ другой таблицы, используется для установления связей между таблицами.**
* **Связи между таблицами — реализуются через ключи и выражают зависимости между сущностями (например, каждый заказ связан с клиентом).**

**Преимущества реляционной модели:**

* **Простота концепции и широкая поддержка.**
* **Гибкость в построении и изменении структуры базы.**
* **Поддержка языка SQL для работы с данными.**
* **Логическая независимость от физической реализации.**
* **Возможность нормализации для устранения избыточности и обеспечения целостности.**

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

**Реляционная алгебра — это формальный язык, описывающий операции над таблицами (отношениями) в реляционной модели. Эти операции позволяют получать новые таблицы из существующих и формируют основу языка SQL.**

**Основные операции:**

* **Проекция (π) — извлечение определённых столбцов из таблицы, то есть создание новой таблицы, содержащей только нужные атрибуты. Пример: π(name, age)(Students) извлекает только имена и возраст.**
* **Выборка (σ) — отбор строк по заданному условию. Пример: σ(age > 18)(Students) выберет студентов старше 18 лет.**
* **Соединение (⋈) — объединение строк из двух таблиц по какому-либо условию (чаще всего по совпадению ключей). Пример: Students ⋈ Courses — соединяет студентов и их курсы.**
* **Объединение (∪) — объединение всех записей двух таблиц с одинаковой структурой, при этом дубликаты удаляются. Пример: Table1 ∪ Table2.**
* **Пересечение (∩) — оставляет только те строки, которые есть одновременно в обеих таблицах. Пример: Table1 ∩ Table2.**
* **Вычитание (−) — возвращает строки из первой таблицы, которых нет во второй. Пример: Table1 − Table2.**
* **Декартово произведение (×) — объединяет каждую строку первой таблицы с каждой строкой второй таблицы, создавая все возможные пары. Пример: Students × Courses — все возможные сочетания студентов и курсов (без учёта связей).**

**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.**
* **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.**

**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 методика проектирования структуры базы данных, направленная на устранение избыточности, дублирования данных и логических аномалий при обновлении. Она предполагает разбиение больших таблиц на более мелкие и установление между ними логических связей. Основной целью нормализации является создание структур, где каждая единица информации хранится в одном месте, и любые изменения не приводят к несогласованности данных.**

**Процесс нормализации формализован через систему нормальных форм (НФ) — наборов требований, которым должна соответствовать таблица. Каждая следующая форма основывается на предыдущей и добавляет новые ограничения.**

**Первая нормальная форма (1НФ) требует, чтобы все значения в таблице были *атомарными* — то есть, в каждой ячейке должно содержаться одно и только одно значение. Это исключает наличие списков, массивов или вложенных структур в отдельных полях. Также таблица должна иметь первичный ключ, уникально идентифицирующий каждую запись.**

***Пример нарушения 1НФ*:  
 Поле phones = "123456, 789012" — нарушает атомарность, так как содержит несколько номеров.  
 *Решение*: создать отдельную таблицу user\_phones(user\_id, phone).**

**Вторая нормальная форма (2НФ) требует, чтобы таблица уже находилась в 1НФ и чтобы все *неключевые атрибуты* зависели *от всего первичного ключа*, а не от его части. Эта форма актуальна только для таблиц с составными ключами. Если есть зависимость только от части ключа, такие атрибуты выносятся в отдельную таблицу.**

***Пример нарушения 2НФ*:  
 Таблица student\_course(student\_id, course\_id, student\_name) — имя студента зависит только от student\_id, а не от всего ключа (student\_id, course\_id).  
 *Решение*: выделить таблицу students(student\_id, student\_name).**

**Третья нормальная форма (3НФ) требует, чтобы таблица уже была во 2НФ и чтобы в ней *не было транзитивных зависимостей*, то есть чтобы никакой неключевой атрибут не зависел от другого неключевого атрибута. Все неключевые поля должны зависеть *только от первичного ключа*.**

***Пример нарушения 3НФ*:  
 Таблица employee(emp\_id, dept\_id, dept\_name) — dept\_name зависит от dept\_id, а не напрямую от emp\_id.  
 *Решение*: вынести информацию о департаментах в отдельную таблицу departments(dept\_id, dept\_name).**

**Нормализация повышает целостность и удобство сопровождения базы данных, хотя в некоторых случаях (например, при высоких требованиях к скорости чтения) может применяться частичная денормализация для ускорения работы.**

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

**Связь между сущностями — это логическая ассоциация, определяющая, каким образом объекты (сущности) в базе данных связаны между собой. Сущность в реляционной модели представляется таблицей, а связь между сущностями — отношением между строками этих таблиц. Связи помогают моделировать реальную структуру данных и позволяют обеспечить корректную интеграцию между объектами.**

**Основные характеристики связи:**

* **Степень связи — определяет количество сущностей, участвующих в связи. Обычно рассматриваются бинарные связи (между двумя сущностями), но могут быть и более сложные (например, тернарные — между тремя).**
* **Тип (кардинальность) связи:**
  + **Один-к-одному (1:1) — каждой записи одной сущности соответствует максимум одна запись другой. Пример: каждый человек — одна биография.**
  + **Один-ко-многим (1:N) — одной записи первой сущности может соответствовать много записей второй, но обратное не верно. Пример: один отдел — много сотрудников.**
  + **Многие-ко-многим (M:N) — каждой записи одной сущности может соответствовать множество записей другой и наоборот. Пример: студенты и курсы — один студент может посещать несколько курсов, и один курс может быть у многих студентов. *Для реализации связи M:N создаётся отдельная промежуточная таблица.***
* **Обязательность участия (опциональность) — определяет, обязана ли сущность участвовать в связи. Например, каждый заказ должен быть привязан к клиенту (обязательное участие), но не каждый клиент обязан иметь заказы (необязательное участие).**
* **Роли в связи — уточняют, как именно участвует каждая сущность в связи. Одна и та же сущность может играть разные роли в разных связях. Например, в связи «сотрудник — руководитель» обе сущности могут быть из одной таблицы employees, но с разными ролями: подчинённый и начальник.**

**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 система управления базами данных (СУБД), входящая в пакет Microsoft Office. Она ориентирована на настольные решения и предназначена для создания, хранения и управления небольшими и средними базами данных. Access позволяет пользователям без глубоких знаний программирования создавать рабочие приложения с графическим интерфейсом и встроенной логикой обработки данных.**

**Основные характеристики Access:**

**Access обладает графическим интерфейсом, который упрощает проектирование базы данных, форм и отчётов. Она поддерживает язык SQL для работы с данными, что позволяет использовать стандартные запросы к базе. Одним из ключевых преимуществ Access является тесная интеграция с другими продуктами Microsoft (Excel, Word, Outlook), что облегчает обмен данными между программами. Также Access предоставляет инструменты для быстрого создания форм и отчётов, а макросы и модули (на языке VBA) позволяют автоматизировать действия и расширять функциональность.**

**Применение Microsoft Access:**

**СУБД Access активно используется в организациях малого и среднего бизнеса. Чаще всего она применяется для создания систем учёта (например, учета товаров, клиентов, сотрудников), ведения каталогов, журналов, регистрационных форм, реестров, хранения контактной информации и создания отчётов по данным. Благодаря низкому порогу входа, Access также подходит для обучения базам данных и быстрого создания прототипов информационных систем.**

**Состав базы данных в Microsoft Access:**

**База данных в Access представляет собой единый файл с расширением .accdb, внутри которого хранятся все объекты базы:**

* **Таблицы — основной элемент базы данных, в которых непосредственно хранятся данные. Каждая таблица содержит строки (записи) и столбцы (поля).**
* **Запросы (Queries) — позволяют извлекать, фильтровать, объединять и изменять данные, используя язык SQL или конструктор запросов.**
* **Формы (Forms) — визуальные элементы интерфейса для удобного ввода и отображения данных пользователем. Часто используются как "окна" приложения.**
* **Отчёты (Reports) — используются для форматированной печати данных. Отчёты позволяют создать шаблоны документов, ведомостей, сводок и т. д.**
* **Макросы (Macros) — простая система автоматизации, позволяющая создавать сценарии из заранее определённых действий (например, открыть форму, запустить запрос).**
* **Модули (Modules) — более гибкая и мощная система программирования на VBA (Visual Basic for Applications), дающая доступ к программной логике и обработке событий.**

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

**Команда SELECT — основная команда языка SQL, предназначенная для выборки данных из таблиц базы данных. Она позволяет извлекать определенные поля, фильтровать строки, сортировать результаты и выполнять агрегатные вычисления.**

**Структура предложения SELECT и его элементы:**

* **SELECT — указывает, какие именно поля следует извлечь (можно указать \* для всех полей). Пример: SELECT name, age**
* **FROM — указывает имя таблицы (или нескольких таблиц), из которой берутся данные. Пример: FROM students**
* **WHERE — задает условия для отбора строк, которые должны попасть в результат. Пример: WHERE age > 18**
* **ORDER BY — сортирует результат по указанному полю (или полям), по возрастанию (ASC) или убыванию (DESC). Пример: ORDER BY name ASC**
* **GROUP BY — группирует строки по значениям указанного поля (или полей), используется при агрегатных функциях (COUNT, SUM, AVG и т. д.). Пример: GROUP BY department**
* **HAVING — фильтрует группы после применения GROUP BY, в отличие от WHERE, который работает до группировки. Пример: HAVING COUNT(\*) > 5**

**Общий пример:**

**SELECT department, COUNT(\*)**

**FROM employees**

**WHERE salary > 30000**

**GROUP BY department**

**HAVING COUNT(\*) > 2**

**ORDER BY COUNT(\*) DESC;**

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

**Существует четыре основных типа запросов к базе данных, каждый из которых выполняет свою задачу:**

* **Выборочные запросы (SELECT) — используются для извлечения данных. Это наиболее часто используемый тип запроса. Пример: SELECT \* FROM products WHERE price < 1000;**
* **Запросы на добавление (INSERT) — позволяют вставлять новые записи в таблицу. Пример: INSERT INTO students (name, age) VALUES ('Ivan', 20);**
* **Запросы на обновление (UPDATE) — применяются для изменения уже существующих записей. Пример: UPDATE students SET age = 21 WHERE name = 'Ivan';**
* **Запросы на удаление (DELETE) — предназначены для удаления строк из таблицы по определённому условию. Пример: DELETE FROM students WHERE age < 18;**

**Каждый из этих типов запросов реализуется через соответствующую SQL-команду. Они позволяют выполнять полный спектр операций с данными: от извлечения и анализа до редактирования и очистки базы.**

**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.**