**1. Назначение и возможности JDBC?**

1. JDBC (Java Database Connectivity) - это API (Application Programming Interface), которое обеспечивает доступ к различным реляционным базам данных с помощью языка Java. JDBC позволяет Java-приложению взаимодействовать с базой данных для выполнения запросов, изменения данных и получения результатов.

**2. Что такое JDBC драйвер? Какие бывают типы?**

1. JDBC драйвер - это компонент, который позволяет Java-приложению взаимодействовать с конкретной базой данных. В зависимости от типа базы данных, существует несколько типов JDBC драйверов:

* JDBC-ODBC Bridge Driver: используется для подключения к базе данных, доступной через ODBC-драйвер;
* Native API Driver: используется для подключения к базе данных, используя специфические API для данной базы данных;
* Network Protocol Driver: используется для подключения к базе данных посредством сетевого протокола, такого как TCP/IP;
* JDBC Thin Driver: используется для подключения к базе данных напрямую через сокеты.

**3. Какие шаги (перечислите) нужно выполнить для извлечения данных?**

* Загрузить драйвер JDBC;
* Установить соединение с базой данных, используя URL-адрес, имя пользователя и пароль;
* Создать запрос SQL, который будет выполнен на базе данных;
* Выполнить запрос, используя соответствующий метод;
* Получить результаты запроса и обработать их.

**4. Как и где задать параметры соединения?**

1. Параметры соединения с базой данных могут быть заданы в URL-адресе при установке соединения или в коде приложения, используя объект Connection. Примеры параметров соединения: имя пользователя, пароль, адрес сервера, порт, имя базы данных и т.д.

**5. Перечислите основные JDBC API компоненты.**

* DriverManager: класс, который управляет драйверами JDBC и устанавливает соединения с базами данных;
* Connection: интерфейс, который представляет соединение с базой данных;
* Statement: интерфейс, который используется для выполнения SQL-запросов на базе данных;
* PreparedStatement: интерфейс, который используется для выполнения предварительно скомпилированных SQL-запросов на базе данных;
* ResultSet: интерфейс, который представляет результаты выполнения SQL-запроса на базе данных.

**6. Какая разница между еxecute, executeQuery, executeUpdate?**

1. Метод execute используется для выполнения любых SQL-запросов, в том числе SELECT, INSERT, UPDATE и DELETE. Метод executeQuery используется только для выполнения SQL-запросов SELECT, которые возвращают наборы результатов. Метод executeUpdate используется для выполнения SQL-запросов INSERT, UPDATE и DELETE, которые не возвращают наборы результатов.

**7. Расскажите об интерфейсе ResultSet.**

1. ResultSet - это интерфейс, который представляет результаты выполнения SQL-запроса на базе данных. ResultSet содержит текущую позицию указателя и предоставляет методы для перемещения указателя по результатам запроса. Результаты запроса хранятся в таблице с колонками, и ResultSet позволяет получить доступ к каждой колонке и каждой строке таблицы.

**8. Как и для чего используют PreparedStatement?**

1. PreparedStatement - это интерфейс, который используется для выполнения предварительно скомпилированных SQL-запросов на базе данных. PreparedStatement предоставляет способ ускорить выполнение повторяющихся запросов, так как запрос компилируется только один раз, а затем может быть многократно выполнен с различными параметрами. PreparedStatement также позволяет избежать SQL-инъекций, так как параметры запроса могут быть безопасно переданы в запрос.

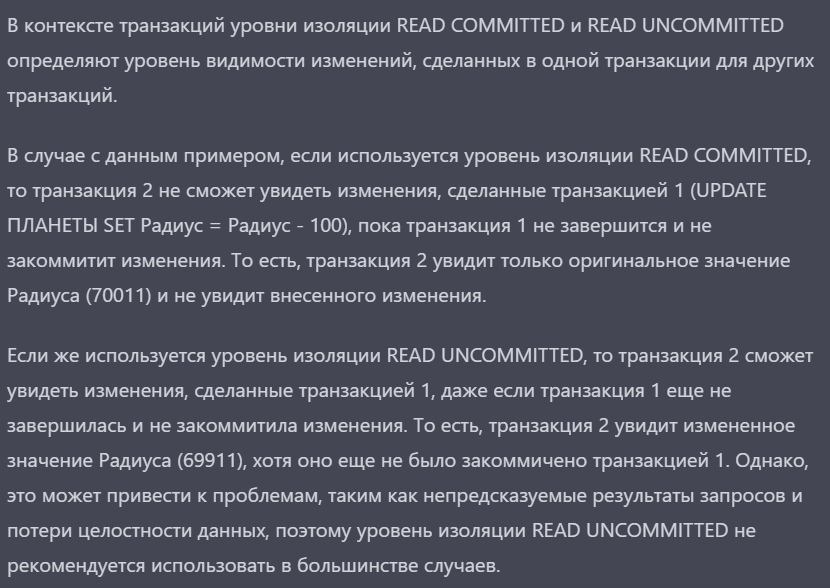
**9. Как и для чего используют транзакции?**

1. Транзакции используются для группировки одного или нескольких SQL-запросов в логически связанные операции, которые должны быть выполнены вместе как единое целое. Если один из запросов не выполняется или происходит ошибка, то все остальные запросы в транзакции будут отменены, чтобы база данных не осталась в неконсистентном состоянии.

**10.Перечислите и поясните типы чтения данных в транзакциях. Что**

**такое уровни изоляции?**

* Read Uncommitted: этот уровень изоляции не предоставляет гарантии того, что данные, которые читаются из таблицы, будут согласованы с другими операциями на базе данных. Это означает, что другие транзакции могут изменять данные, которые читаются, в то время как эта транзакция еще не завершилась;
* Read Committed: этот уровень изоляции гарантирует, что данные, которые читаются из таблицы, будут согласованы только с другими операциями на базе данных, которые уже завершены. Это означает, что другие транзакции могут изменять данные, которые читаются, в то время как эта транзакция еще не завершилась;
* Repeatable Read (повторяемое чтение): транзакция видит только те данные, которые были прочитаны в начале транзакции. Другие транзакции не могут изменять или удалять эти данные.
* Serializable (сериализуемая): данные в транзакции не могут быть изменены или удалены другими транзакциями, пока транзакция не будет завершена.

**11.Что такое DatabaseMetaData интерфейс?**

1. DatabaseMetaData - это интерфейс, предоставляемый JDBC, который позволяет получить метаданные о базе данных, с которой установлена соединение. Этот интерфейс предоставляет информацию о таблицах, столбцах, индексах, ограничениях, процедурах и многом другом.

С помощью DatabaseMetaData можно получить информацию о структуре базы данных, что может быть полезным при написании общих абстракций для работы с различными базами данных, а также при создании универсальных инструментов управления базами данных.

**12.Зачем используют шаблон DAO?**

Шаблон DAO (Data Access Object) является одним из наиболее распространенных шаблонов проектирования для работы с базами данных в приложениях, написанных на языке Java. Его основное назначение - абстрагирование бизнес-логики от конкретных операций с базой данных.

DAO-шаблон позволяет разделить работу с базой данных на две части: слой доступа к данным (Data Access Layer) и слой бизнес-логики (Business Logic Layer). С помощью этого шаблона можно достичь следующих преимуществ:

1. Уменьшение связанности (coupling) между слоями приложения. DAO предоставляет абстрактный интерфейс для работы с данными, который скрывает детали реализации и способ хранения данных. Таким образом, при изменении источника данных (например, переход с одной СУБД на другую) не требуется изменять код бизнес-логики.
2. Упрощение тестирования. DAO-объекты могут быть заменены на мок-объекты во время тестирования бизнес-логики. Это позволяет тестировать бизнес-логику независимо от способа хранения данных.
3. Увеличение переиспользуемости кода. DAO-объекты могут быть использованы в разных частях приложения для выполнения одних и тех же операций с базой данных.
4. Улучшение безопасности. DAO-шаблон помогает предотвратить атаки SQL-инъекций, так как запросы к базе данных формируются с использованием параметризованных запросов, а не конкатенации строк.

Использование шаблона DAO позволяет улучшить архитектуру приложения, сделать его более модульным и поддерживаемым.