ЗМІСТ

1. Що таке dependency injection?
2. Що таке Spring IoK контейнер?
3. Які типи IoK?
4. Які загальні реалізації ApplicationContext. В чому різниця між BeanFactory та ApplicationContext?
5. Які scope підтримує Spring. Приклади використання
6. Які етапи життєвого циклу компонента. Основні інтерфейси. приклади.
7. Аспектно-орієнтоване програмування у Spring
8. Які основні компоненти фреймворку Spring MVC
9. Що таке starter у Spring – створення, використання.
10. Робота з кількома різними БД в одному додатку Spring
11. Конфігурування Spring додатків: @Configuration, @ComponentScan, @Value, properties, використання змінних оточення є
12. Опишіть Spring Security. Архитектура.
13. Що таке CORS і як з ним працювати у Spring
14. Розкажіть про проксі та про @Transactional. Як працює та навіщо? Які можуть бути проблеми? Чи можна навісити @Transactional на приватний метод? А якщо викликати метод з @Transactional всередині іншого методу з @Transactional одного класу – буде працювати?
15. Як обробляється http-запит у Spring?
16. Обробка exceptions у Spring
17. Які є кеші Hibernate і які працюють за замовчуванням?
18. Чим відрізняється Lazy від Eager у Hibernate?
19. Що таке 'проблема N+1 запиту' під час використання Hibernate? Коли з'являється? Як вирішити? Як виявити?
20. Як описати complex primary key під час використання Hibernate?
21. Як можна відобразити наслідування на БД за допомогою JPA (Hibernate)?
22. Рівні ізольованості транзакції. Яку проблему вирішує кожний рівень? Який за замовчуванням у більшості реляційних БД?
23. Яка роль інтерфейсу Session у Hibernate
24. Яка роль інтерфейсу SessionFactory у Hibernate
25. Що таке Lazy fetching у Hibernate
26. Кешування в Hibernate
27. Що таке Named SQL Query
28. Як керувати транзакціями за допомогою Hibernate
29. Чому ми не повинні робити Entity class як final?
30. У чому різниця між Hibernate save(), saveOrUpdate() та persist()?

- 31. Основні компоненти Kafka

32. Що таке топіки та розділи (partitions) у Kafka?
33. Dead Letter Queue у Kafka
34. Що таке фіксації зсувів і чому вони такі важливі для Kafka

1. Що таке dependency injection?

https://docs.spring.io/spring-framework/reference/core/beans/dependencies/factory-collaborators.html

Dependency Injection – це методи для реалізації DIP. DIP – Dependency Inversion Principle – є одним з принципів SOLID. Він зазначає, що високорівневі модулі не мають залежати від низькорівневих модулів. Обидва мають залежати від абстракцій. Абстракції не мають залежати від деталей реалізації. Деталі реалізації залежать від абстракцій.

Dependency Injection (в Spring) – це процес, за допомогою якого об’єкти визначають свої залежності. Залежності можуть бути визначені через аргументи конструктора або аргументи фабричного методу. За допомогою IoC контейнеру в Spring відбувається побудова бінів. Таким чином, біну не потрібно знати, які саме реалізації інтерфейсів використовуються та як вони створюються. Контроль над цим передається IoC контейнеру (звідки й назва Inversion of Control). Код завдяки цьому стає чистішим, легшим для тестування та легше розділяється на компоненти.

Для Dependency Injection використовуються Constructor Injection, Setter Injection (основні), Field Injection (не рекомендовано у Spring) та Interface Injection (окремий, не реалізований у Spring; полягає у створенні інтерфейсів з сетер-методами;).

2. Що таке Spring IoK контейнер?

https://docs.spring.io/spring-framework/reference/core/beans/introduction.html

Типи: Bean Factory, Application Context;

Spring IoC контейнер представлений у вигляді інтерфейсу context.ApplicationContext. Він відповідальний за створення, конфігурацію та збірку бінів. Контейнер отримує відповідні інструкції. Інструкції можуть бути подані у вигляді анотацій всередині класів бінів, класі-конфігурації з фабричними методами або XML конфігурації. Відповідно вирізняють Annotation-based configuration, Java-based configuration, XML-based configuration, Groovy based configuration.

Після створення Application Context отримується повністю готова до використання система.

Використання контейнеру. Викликаються методи getBean(name, type), getBean(type), тощо.

https://www.baeldung.com/spring-beanfactory-vs-applicationcontext

3. Які типи IoK?

Типи: Bean Factory, Application Context;
Bean Factory - простий інтерфейс
Application Context - більш складний інтерфейс. Легше інтегрується з АОП, ініціалізує все на початку, більш гнучка конфігурація, messaging, ширша підтримка Bean Scopes.
https://www.baeldung.com/spring-beanfactory-vs-applicationcontext

Збереження різних scope бінів:
Singleton - зберігається в Application Context
Prototype - зберігається у клієнтському коді (не зберігається в контейнері, лише створюється ним)
Request - зберігається в атрибуті http-реквесту
Session - (The session-scoped beans are stored inside the HttpSession object itself.)
Application - зберігається в атрибуті Servlet Context
Web Socket - WebSocket session attributes

4. Які загальні реалізації ApplicationContext. В чому різниця між BeanFactory та ApplicationContext?

https://docs.spring.io/spring-framework/reference/core/beans/introduction.html

Bean Factory – інтерфейс, що надає можливості конфігурації для будь-якого типу об’єктів. Application Context – розширення інтерфейсу Bean Factory, що додає:

- Легшу інтеграцію зі Spring AOP
- Message Resource Handling
- Event publication
- Web application Context

Найчастіше використовуються такі реалізації Application Context як AnnotationConfigApplicationContext та ClassPathXmlApplicationContext

5. Які scope підтримує Spring. Приклади використання

https://docs.spring.io/spring-framework/reference/core/beans/factory-scopes.html

Scope – це особливість взаємодії біна з контейнером, що визначає скільки разів та як часто має створюватися бін.

Singleton – за замовчуванням, один об'єкт в IoC контейнері. Для бінів без стану.

Prototype – можна створювати багато об'єктів, але вони не зберігаються в контейнері (далі їх життєвим циклом керує програміст). Корисно для бінів зі станом.

Інші використовуються в реалізаціях Application Context, що взаємодіють з web (наприклад, XmlWebApplicationContext):

Request – на кожен HTTP запит.

Session – на кожну HTTP сесію.

Application – на кожен Servlet Context (коли запускається Tomcat

Websocket – на кожен Web-сокет.

Можна створювати власні Scope за допомогою інтерфейсу Scope та методу void registerScope(String scopeName, Scope scope) в Configurable Bean Factory.

6. Які етапи життєвого циклу компонента. Основні інтерфейси. Приклади.

https://www.baeldung.com/spring-component-annotation
https://www.geeksforgeeks.org/spring-component-annotation-with-example/
Анотація @Component дозволяє Spring виявляти біни при запуску програми. Таким чином, компонент створюється разом із запуском програми і існує до завершення її роботи.
Основні інтерфейси: @Service, @Repository, @Controller.
@Service використовується для позначення класів, що містять сервіси або бізнес-логіку.
@Repository - для класів, у яких реалізовані CRUD-операції, наприклад DAO або репозиторії, які працюють із базами даних.
@Controller - класи, які обробляють запити користувачів та повертають відповіді на них, наприклад REST-контролери.

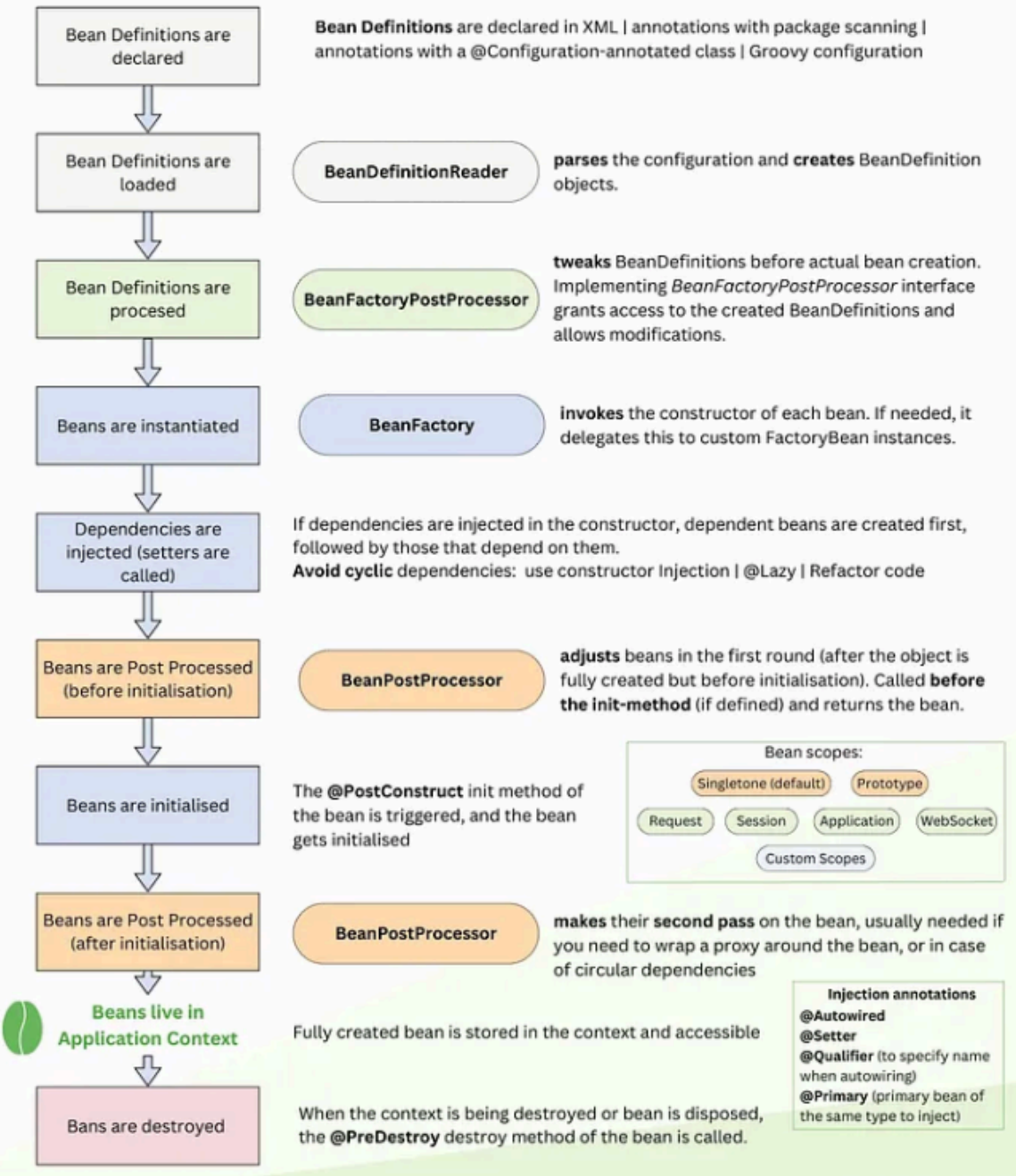
https://www.geeksforgeeks.org/bean-life-cycle-in-java-spring/

https://medium.com/@TheTechDude/spring-bean-lifecycle-full-guide-f865966e89ce

Spring Bean Lifecycle CheatSheet

Spring Bean - POJO (Plain Old Java Object) managed by the IoC container (Spring)

Bean annotations:
@Bean **@Component** **@Service**
@Controller **@RestController** **@Repository**



7. Аспектно-орієнтоване програмування у Spring

AOP is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns

АОП – це інший погляд на програмування. Якщо в ООП одиницею є клас, то в АОП це аспект. Аспекти дозволяють модуляризувати деякі відповідальності, що зустрічаються вздовж декількох об'єктів (наприклад, транзакції). Такі відповідальності називаються наскрізними.

АОП та IoC контейнер в Spring є незалежними, тобто контейнер може існувати і без АОП

АОП в Spring реалізується або за допомогою schema-based approach, або за «у стилі» анотації @AspectJ.

АОП використовується в Spring для того щоб:

- Надавати декларативні сервіси, зокрема декларативний менеджмент транзакцій
- Надавати можливість користувачам створювати власні аспекти

https://docs.spring.io/spring-framework/reference/core/aop/introduction-defn.html

Основні поняття:

Aspect – Join Point – Advice – Pointcut – Introduction – Target Object – AOP Proxy – Weaving

Weaving = linking (Spring робить це в runtime)

Using JDK dynamic proxies; AOP proxy

Типу advice в Spring:

- Before advice (перед join point / методом, не може зупинити виконання методу, не викинувши помилку)
- After rerturning advice – після успішного виконання
- After throwing advice – після невдалого виконання
- After (finally) advice – після будь-якого виконання

- Around advice – обгортка методу; дозволяє визначити дії як до, так і після виклику; дозволяє не викликати метод (join point)

Останній є найпотужнішим, але рекомендується обходитися мінімальним типом advice, що задовольняє потреби.

Всі класи, позначені @AspectJ включаються в конфігурацію AOP. Вседенині аспекту визначаються pointcuts:

```
@Pointcut("execution(* transfer(..))") // the pointcut expression
private void anyOldTransfer() {} // the pointcut signature
```

Важливо, що перехоплення відбуваються тільки якщо метод знаходиться в інтерфейсі та є публічним. Щоб забезпечити перехоплення інших методів варто скористатися native AspectJ weaving замість Spring proxy-based AOP.

8. Які основні компоненти фреймворку Spring MVC

Spring MVC - це фреймворк Spring для web-застосунків побудований на основі Servlets API. З чого складається:

- Dispatcher Servlet (Front Controller) - розподіляє запити між контролерами, Path Mapping.
- WebApplicationContext
- @Controller, @RestController - контролери, обробляють запити
- View - подання - JSP, HTML, JSON, XML; View Resolver
- Exception Hanlder
- Interceptors: https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-config/interceptors.html
- Validation: @Valid, @Validated

9. Що таке starter у Spring – створення, використання.

Starter POMs are a set of convenient dependency descriptors that you can include in your application.
Стартери містять усі потрібні залежності для простого початку роботи зі spring.
Підібрати стартери можна або підключивши їх у pom.xml (maven), або згенерувавши потрібний код за допомогою Spring Initializr (https://start.spring.io/).
Spring starter web містить модулі, потрібні для створення web-застосунку.
Spring started jpa містить модулі, потрібні для роботи з базою даних.

https://www.baeldung.com/spring-boot-starters
https://www.geeksforgeeks.org/spring-boot-starters/

10. Робота з кількома різними БД в одному додатку Spring

У Spring Boot можна використати файл application.properties і вказати необхідні дані для підключення до бази даних. Наприклад,

```
spring.datasource.url = [url]
spring.secondDatasource.url = [url]
@Bean
@Primary
@ConfigurationProperties(prefix="spring.datasource")
public DataSource primaryDataSource() {
   return DataSourceBuilder.create().build();
}

@Bean
@ConfigurationProperties(prefix="spring.secondDatasource")
public DataSource secondaryDataSource() {
   return DataSourceBuilder.create().build();
}
```

Також можна налаштувати окрему бази даних для тестів. Часто використовується in-memory database, зокрема h2. Тоді у application.properties, що знаходиться у тестовій директорії, потрібно вказати потрібні дані для підключення до тестової бази даних.

11. Конфігурування Spring додатків: @Configuration, @ComponentScan, @Value, properties, використання змінних оточення

https://docs.spring.io/spring-framework/reference/core/beans/java/configuration-annotation.html

https://docs.spring.io/spring-framework/reference/core/beans/annotation-config/value-annotations.html

https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/context/annotation/ComponentScan.html

Анотація @Configuration використовується найчастіше у Java based конфігурації. Всередині configuration класу знаходиться factory методи для бінів, що позначені анотацією @Bean.

Також цей клас використовується в annotation based конфігурації та є ідентифікатором для контейнера (цей клас поміщується в конструктор при створенні AnnotationConfigApplicationContext). Анотація @ComponentScan показує, який пакет має бути просканований. Класи, що позначені анотаціями @Service, @Component будуть додані до контейнера.

Анотація @Value дозволяє вказувати підтягувати певні значення з конфігурації або змінних оточення. Наприклад, @Value("${catalog.name}").

12. Опишіть Spring Security. Архитектура.

Spring Security - це потужний фреймворк для автентифікації та контролю доступу, який легко налаштовується. Це стандарт де-факто для захисту додатків на основі Spring.

Spring Security - це фреймворк, який фокусується на забезпеченні автентифікації та авторизації Java-додатків. Як і всі Spring-проекти, справжня сила Spring Security полягає в тому, наскільки легко його можна розширити, щоб задовольнити індивідуальні вимоги.

Загальний опис:
https://spring.io/projects/spring-security
Архітектура:
https://docs.spring.io/spring-security/reference/servlet/architecture.html

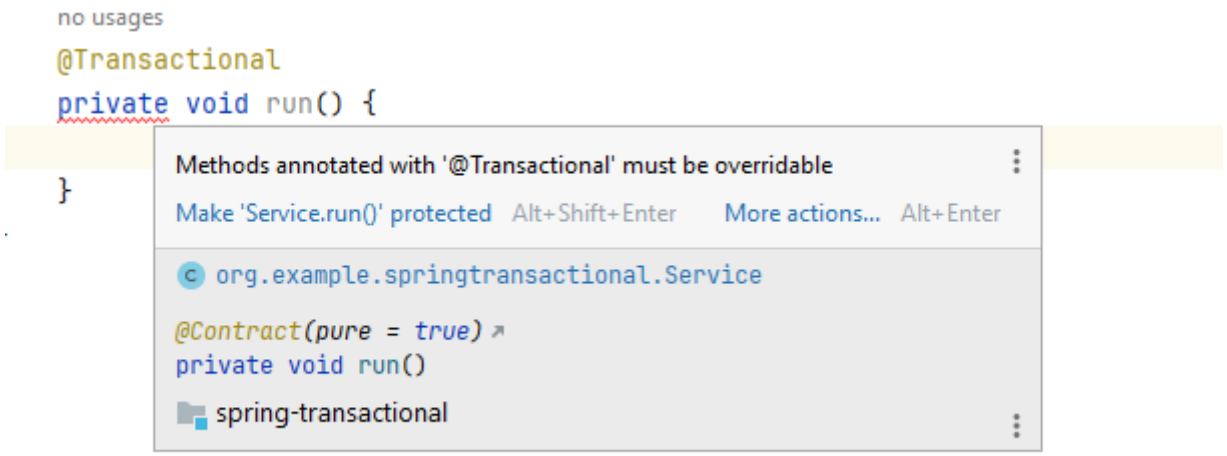
13. Що таке CORS і як з ним працювати у Spring

https://www.baeldung.com/cs/cors-preflight-requests
https://www.baeldung.com/spring-cors
CORS (Cross-Origin Resource Sharing) - політика, що визначає, яким чином скрипти, що знаходяться на одному джерелі (origin), можуть запитувати ресурси із іншого джерела. Джерела вважаються однаковими, якщо у них співпадають схема, домен і номер порта за наявності. Якщо хоч одна із цих властивостей відрізняється, то маємо справу із CORS.
У Spring CORS імплементується застосуванням анотації @CrossOrigin до самого класу-контролера і/або до його методів.
Можна також застосовувати CORS до всієї програми, а не лише до окремого контролера, використовуючи конфігурацію, яка реалізована або як Java-клас (у ньому до об’єкта CorsRegistry додаються/видаляються URI, хедери, які дозволено приймати), або як XML-файл (ця ж інформація, виражена у атрибутах відповідного тега).
Ще трошки про Spring Security треба)

14. Розкажіть про проксі та про @Transactional. Як працює та навіщо? Які можуть бути проблеми? Чи можна навісити @Transactional на приватний метод? А якщо викликати метод з @Transactional всередині іншого методу з @Transactional одного класу – буде працювати?

https://docs.spring.io/spring-framework/reference/data-access/transaction/declarative/annotations.html

The @Transactional annotation is typically used on methods with public visibility. As of 6.0, protected or package-visible methods can also be made transactional for class-based proxies by default. Note that transactional methods in interface-based proxies must always be public and defined in the proxied interface. For both kinds of proxies, only external method calls coming in through the proxy are intercepts.



Щодо Nested Transactional:

In proxy mode (which is the default), only external method calls coming in through the proxy are intercepted. This means that self-invocation, in effect, a method within the target object calling another method of the target object, will not lead to an actual transaction at runtime even if the invoked method is marked with @Transactional. Also, the proxy must be fully initialized to provide the expected behaviour so you should not rely on this feature in your initialization code, i.e. @PostConstruct.

15. Як обробляється http-запит у Spring?

- Spring використовує Embedded Tomcat у якості веб-сервера.
- Dispatcher Servlet - приймає запит.
- Handler Mapping - визначає контролер, що має обробити запит
- Controller - обробляє запит, визначає дані, що повертаються клієнту. Виклик метода в контролері відбувається за допомогою Handler Adapter. https://www.baeldung.com/spring-mvc-handler-adapters
- Обробка результату: Model / View https://www.baeldung.com/spring-mvc-model-model-map-model-view
- Також можливі виклики Interceptor-ів та Error Handler-ів.

16. Обробка exceptions у Spring

Global Exception Handler - підхід для обробки помилок.
@ControllerAdvice - ставиться над класом, що обробляє помилки.,
@ExceptionHandler - ставиться над методом; вказується, яка помилка обробляється

https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc

17. Які є кеші Hibernate і які працюють за замовчуванням?

https://www.geeksforgeeks.org/hibernate-caching/
https://habr.com/ru/articles/135176/

Перший рівень - на кожну сесію, дефолтний
Другий рівень - на кожну Session Factory

Кеш запитів - для кешування результату запиту, що виконується з такими самими параметрами.

18. Чим відрізняється Lazy від Eager у Hibernate?

Lazy - залишає поле не проініціалізованим, дістаючи proxy-об’єкт з бази даних. Потім, при необхідності отримати значення цього поля, робиться додатковий запит до бази даних.
Eager - автоматично дістає з бази даних та створює залежні об’єкти
Модифікуються через @FetchType
За замовчуванням, зв’язки між реляціями мають такі fetch types:
@OneToOne - eager
@OneToMany - lazy
@ManyToOne - eager
@ManyToMany - lazy
Використання eager може вирішити проблему виконання багатьох додаткових запитів (N+1), але не його використання може призвести до виконання складних запитів, що повертають багато зайвих даних (в тому числі великі списки об’єктів).

19. Що таке 'проблема N+1 запиту' під час використання Hibernate? Коли з'являється? Як вирішити? Як виявити?

N+1: проблема, пов’язана з виконанням додаткових запитів до бази даних з ціллю додаткової ініціалізації об’єктів у списку. Вирішується завдяки використанню JOIN FETCH в запиті або за допомогою додаткового запиту, щоб отримати колекцію об’єктів, яких не вистачає та вже в java-коді поставити зв’язки між ними.
https://www.baeldung.com/spring-hibernate-n1-problem
Виявити може бути складно. Деякі підходи: spring-hibernate-query-utils, EntityGraph, DataLoader патерн

20. Як описати complex primary key під час використання Hibernate?

Використовується анотація @IdClass. Це має бути публічний клас, мати дефолтний конструктор, має equals() та hashCode(), має реалізувати інтерфейс Serializable.

https://www.baeldung.com/jpa-composite-primary-keys

21. Як можна відобразити наслідування на БД за допомогою JPA (Hibernate)?

@MappedSuperclass
Single Table
Joined Table
Table per class
https://habr.com/ru/articles/337488/
https://www.baeldung.com/hibernate-inheritance

22. Рівні ізольованості транзакції. Яку проблему вирішує кожний рівень? Який за замовчуванням у більшості реляційних БД?

https://javarush.com/ua/quests/lectures/ua.questhibernate.level18.lecture01
https://learn.microsoft.com/en-us/sql/odbc/reference/develop-app/transaction-isolation-levels?view=sql-server-ver16

Під «рівнем ізоляції транзакцій» розуміється ступінь захисту, що забезпечується внутрішніми механізмами СУБД

1. Transaction None
2. Всі типи читання - read uncommitted
3. Заборона Dirty read (Postgres, Oracle default) - read committed
4. Дозволено тільки Phantom read (MySQL default) - repeatable read
5. Transaction Serializable - serializable

23. Яка роль інтерфейсу Session у Hibernate?

https://docs.jboss.org/hibernate/orm/3.5/api/org/hibernate/Session.html
https://www.tutorialspoint.com/hibernate/hibernate_sessions.htm
Інтерфейс Session використовується для отримання фізичного зв'язку із базою даних. За допомогою Session виконуються CRUD-операції із об'єктами-сутностями (Entity).

Объект-сущность может находиться в одном из 3-х состояний (статусов):

transient object. Объекты в данном статусе — это заполненные экземпляры классов-сущностей. Могут быть сохранены в БД. Не присоединены к сессии. Поле Id не должно быть заполнено, иначе объект имеет статус detached ;
persistent object. Объект в данном статусе — так называемая хранимая сущность, которая присоединена к конкретной сессии. Только в этом статусе объект взаимодействует с базой данных. При работе с объектом данного типа в рамках транзакции все изменения объекта записываются в базу;
detached object. Объект в данном статусе — это объект, отсоединённый от сессии, может существовать или не существовать в БД.

Любой объект-сущность можно переводить из одного статуса в другой. Для этого в интерфейсе Session существуют следующие методы:

persist(Object) — преобразует объект из transient в persistent, то есть присоединяет к сессии и сохраняет в БД. Однако, если мы присвоим значение полю Id объекта, то получим PersistentObjectException — Hibernate посчитает, что объект detached, т. е. существует в БД. При сохранении метод persist() сразу выполняет insert, не делая select.
merge(Object) — преобразует объект из transient или detached в persistent. Если из transient, то работает аналогично persist() (генерирует для объекта новый Id, даже если он задан), если из detached — загружает объект из БД, присоединяет к сессии, а при сохранении выполняет запрос update
replicate(Object, ReplicationMode) — преобразует объект из detached в persistent, при этом у объекта обязательно должен быть заранее установлен Id. Данный метод предназначен для сохранения в БД объекта с заданным Id, чего не позволяют сделать persist() и merge(). Если объект с данным Id уже существует в БД, то поведение определяется согласно правилу из перечисления org.hibernate.ReplicationMode:
ReplicationMode.IGNORE — ничего не меняется в базе.

ReplicationMode.OVERWRITE — объект сохраняется в базу вместо существующего.
ReplicationMode.LATEST_VERSION — в базе сохраняется объект с последней версией.
ReplicationMode.EXCEPTION — генерирует исключение.
delete(Object) — удаляет объект из БД, иными словами, преобразует persistent в transient. Object может быть в любом статусе, главное, чтобы был установлен Id.
save(Object) — сохраняет объект в БД, генерируя новый Id, даже если он установлен. Object может быть в статусе transient или detached
update(Object) — обновляет объект в БД, преобразуя его в persistent (Object в статусе detached)
saveOrUpdate(Object) — вызывает save() или update()
refresh(Object) — обновляет detached-объект, выполнив select к БД, и преобразует его в persistent
get(Object.class, id) — получает из БД объект класса-сущности с определённым Id в статусе persistent

Объект Session кэширует у себя загруженные объекты; при загрузке объекта из БД в первую очередь проверяется кэш. Для того, чтобы удалить объект из кэша и отсоединить от сессии, используется session.evict(Object). Метод session.clear() применит evict() ко всем объектам в сессии.

A Session instance is serializable if its persistent classes are serializable

24. Яка роль інтерфейсу SessionFactory у Hibernate

https://docs.jboss.org/hibernate/orm/6.5/javadocs/org/hibernate/SessionFactory.html
https://www.digitalocean.com/community/tutorials/hibernate-sessionfactory
SessionFactory - клас-фабрика, що має бути присутнім у додатку у вигляді Singleton, за допомогою якого отримується об’єкт Session, за допомогою якого здійснюється робота із базою даних.
Отримати цей об’єкт можна з допомогою трьох методів:
getCurrentSession() - отримання сесії, прив’язаної до контексту
openSession() - створення нового об’єкту сесії, після завершення роботи із яким потрібно його закривати
openStatelessSession() - отримання сесії, для якої не застосовується кешування

25. Що таке Lazy fetching у Hibernate

Lazy Fetching відповідає за послідовну загрузку Entity. Для цього використовується proxy, яке з використанням поточної сесії до бази даних доініціалізовує власні поля при необхідності.
https://www.baeldung.com/hibernate-lazy-eager-loading

26. Кешування в Hibernate

Див. детальніше пункт 17.
https://www.geeksforgeeks.org/hibernate-caching/

27. Що таке Named SQL Query?

The hibernate named query is a way to use any query by some meaningful name. It is like using alias names. The Hibernate framework provides the concept of named queries so that application programmers need not to scatter queries to all the java code.
https://www.baeldung.com/hibernate-named-query

A named query is a statically defined query with a predefined unchangeable query string. Using named queries instead of dynamic queries may improve code organization by separating the JPQL query strings from the Java code. It also enforces the use of query parameters rather than embedding literals dynamically into the query string and results in more efficient queries.
https://www.objectdb.com/java/jpa/query/named

28. Як керувати транзакціями за допомогою Hibernate?

https://javarush.com/quests/lectures/questhibernate.level16.lecture03
https://www.javatpoint.com/hibernate-transaction-management-example

A transaction is associated with Session and instantiated by calling session.beginTransaction().

The methods of Transaction interface are as follows:

void begin() starts a new transaction.
void commit() ends the unit of work unless we are in FlushMode.NEVER.
void rollback() forces this transaction to rollback.
void setTimeout(int seconds) it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
boolean isAlive() checks if the transaction is still alive.
void registerSynchronization(Synchronization s) registers a user synchronization callback for this transaction.
boolean wasCommited() checks if the transaction is commited successfully.
boolean wasRolledBack() checks if the transaction is rolledback successfully.

Про транзакції: ACID: Atomicity, Consistency, Isolation, Durability
https://www.geeksforgeeks.org/acid-properties-in-dbms/

29. Чому ми не повинні робити Entity class як final?

Тому що Hibernate створює proxy, що наслідує Entity клас. Final не дозволяє наслідування.
An entity class must follow these requirements.

- The class must be annotated with the javax.persistence.Entity annotation.
- The class must have a public or protected, no-argument constructor. The class may have other constructors.
- The class must not be declared final. No methods or persistent instance variables must be declared final.
- If an entity instance is passed by value as a detached object, such as through a session bean's remote business interface, the class must implement the Serializable interface.
- Entities may extend both entity and non-entity classes, and non-entity classes may extend entity classes.
- Persistent instance variables must be declared private, protected, or package-private and can be accessed directly only by the entity class's methods. Clients must access the entity's state through accessor or business methods.

30. У чому різниця між Hibernate save(), saveOrUpdate() та persist()?

Both save() and persist() are used to insert a new entity in the database. You're calling them on entities that already exist in the database. So they do nothing.

The main difference between them is that save() is Hibernate-proprietary, whereas persist() is a standard JPA method. Additionally, save() is guaranteed to assign and return an ID for the entity, whereas persist() is not.

update() is used to attach a detached entity to the session.

saveOrUpdate() is used to either save or update an entity depending on the state (new or detached) of the entity.

31. Основні компоненти Kafka

1. **Producer**: Продюсери створюють дані та надсилають їх до тем Kafka. Дані можуть бути будь-якими, від log-подій до дій користувача. Виробники публікують дані в темах, які є логічними каналами або категоріями.
2. **Broker**: Брокери являють собою ядро кластера Kafka. Вони зберігають дані, керують розділами тем і обробляють запити клієнтів. Кожен брокер може розміщувати декілька тематичних розділів, а кластери Kafka складаються з кількох брокерів, які працюють разом.
3. **Topic**: Topics are the channels through which data is organized and categorized. Producers publish data, and consumers subscribe to topics to receive data updates.
4. **Partition**: Topics can be divided into partitions, allowing for parallel processing and scalability. Each partition is replicated across multiple brokers to ensure fault tolerance.
5. **Consumer**: Consumers subscribe to one or more topics and process the published data. They read data from partitions and can maintain their offset, allowing them to track their progress in processing.
6. **Consumer Group**: Consumers reading from a topic can be organized into consumer groups. Each message in a partition is delivered to only one consumer within a group. This allows for load balancing and scalability.

32. Що таке топіки та розділи (partitions) у Kafka?

https://kafka.apache.org/documentation

Topic - це спосіб для організації та збереження подій. Topic може мати 0 або більше продюсерів подій, так само як 0 або більше консюмерів. При цьому події не видаляються з топіку після того, як були оброблені консюмером. Навпаки: можна визначити, як довго події зберігатимуться у топіку.

Topics are partitioned, meaning a topic is spread over a number of "buckets" located on different Kafka brokers. This distributed placement of your data is very important for scalability because it allows client applications to both read and write the data from/to many brokers at the same time. When a new event is published to a topic, it is actually appended to one of the topic's partitions. Events with the same event key (e.g., a customer or vehicle ID) are written to the same partition, and Kafka guarantees that any consumer of a given topic-partition will always read that partition's events in exactly the same order as they were written.

Topic - це щось на кшталт папки у файловій системі, тоді як події (повідомлення) є типу файлами.

33. Dead Letter Queue y Kafka

A Dead Letter Queue (DLQ) is used to store messages that cannot be correctly processed due to various reasons, for example, intermittent system failures, invalid message schema, or corrupted content.

These messages can be later removed from the DLQ for analysis or reprocessing.

https://habr.com/ru/companies/slurm/articles/691682/
https://www.baeldung.com/kafka-spring-dead-letter-queue
https://medium.com/@sannidhi.s.t/dead-letter-queues-dlqs-in-kafka-afb4b6835309

34. Що таке фіксації зсувів і чому вони такі важливі для Kafka

https://kafka-school.ru/blog/kafka-offsets/
https://www.croxyproxy.com/_ru/
https://www.baeldung.com/kafka-commit-offsets
Aka Commit Offsets
Смещение (offset) в Kafka — это индекс (порядковый номер), указывающий на положение записи в разделе (partititon) топика. Информация о смещениях (порядковые номера всех сообщений) хранится в специальном топике `__consumer_offsets`. Kafka узнает о появлении новой записи (сообщений) благодаря механизму фиксации смещений. Фиксация смещений — это обновление текущей позиции (или добавление новой) записи в разделе топика. Существуют следующие виды фиксаций смещений:

- синхронная фиксация смещений;
- асинхронная фиксация смещений.

Каждый из этих видов мы подробнее рассмотрим далее.

Синхронная фиксация смещений
Синхронная фиксация смещений — это автоматическая фиксация текущего смещения записи в момент ее появления. Как только смещение успешно фиксируется, выполнение процедуры завершается. В случае сбоя синхронной фиксации генерируется исключение, и фиксация возобновляется, выполняясь до тех пор, пока смещение не зафиксируется. Следующий код на языке Java отвечает за выполнение синхронной фиксации смещений записей с помощью метода commitSync() [2]:

```
while (true) {
ConsumerRecords< String, String > records = consumer.poll(100);
for (ConsumerRecord<String, String> record : records){
System.out.printf("topic = %s, partition = %s, offset =
%d, customer = %s, country = %s\n",
record.topic(), record.partition(),
record.offset(), record.key(), record.value());
}
try {
consumer.commitSync();
} catch (CommitFailedException e) {
log.error("commit failed", e)}}
```

Из кода видно, что фиксация смещения происходит после получения приложением каждой записи (с помощью метода poll()). Метод commitSync() будет повторять фиксацию каждой новой записи до тех пор, пока не возникнет непоправимая ошибка типа CommitFailedException (например, полный выход из строя Kafka-сервера). Если произойдет такая ошибка, сведения о ней автоматически запишутся в журнал логирования (logging), который содержит информацию об этапах выполнения программы.

Асинхронная фиксация смещений

При синхронной фиксации смещений приложение блокируется (остальные функции и запросы становятся недоступны) до тех пор, пока брокер Kafka не подтвердит успешную фиксацию. Это ограничивает пропускную способность (количество информации, передаваемое в единицу времени) приложения. В этом случае можно использовать асинхронную фиксацию. Асинхронная фиксация смещений — это фиксация, которая выполняется независимо (параллельно) от выполнения остальных функций приложения и не требует обязательного подтверждения факта успешной фиксации от Kafka-сервера. Следующий код на языке Java отвечает за выполнение асинхронной фиксации смещения записей в топике с помощью метода commitAsync():

```
while (true) {
ConsumerRecords<String, String> records = consumer.poll(100);
for (ConsumerRecord<String, String> record : records){
System.out.printf("topic = %s, partition = %s,
offset = %d, customer = %s, country = %s\n",
record.topic(), record.partition(), record.offset(),
record.key(), record.value());}
consumer.commitAsync();
}
```

Главное отличие между асинхронной и синхронной фиксациями состоит в том, что синхронная фиксация будет повторять попытку фиксации смещения до тех пор, пока она не завершится успешно (за исключением полного выхода из строя сервера Kafka). Асинхронная фиксация, в случае возникновения ошибочной ситуации (например, истечение времени ожидания или временный сбой Kafka-сервера), не станет повторять попытку фиксации смещения текущей записи, а сразу перейдет к фиксации смещения следующей доступной (или поступившей) записи.

Таким образом, благодаря механизму управления фиксациями, брокер Kafka может весьма эффективно регистрировать новые записи Big Data в топиках, обращаться к старым, а также удалять из топиков записи, которые более не используются. Это делает Apache Kafka универсальным и надежным средством для хранения и обмена большими потоками данных, что позволяет активно использовать этот брокер сообщений в задачах Data Science и разработке распределенных приложений. В следующей статье мы поговорим про перебалансировку разделов в Kafka.