

Тест начат Суббота, 17 Апрель 2021, 12:30

Состояние Завершено

Завершен Суббота, 17 Апрель 2021, 18:41

Прошло времени 6 час. 11 мин.

Баллы 6,00/15,00

Оценка 40,00 из 100,00

Вопрос 1

Неверно

Баллов: 0,00 из 1,00

Два Spring-бина инжектят *EntityManager* посредством аннотации *@PersistenceProvider* (см. *com.netcracker.ec.jpa.test.service.injects.InjectingServiceOne, InjectingServiceTwo*). Что можно сказать о двух экземплярах *EntityManager*, которые будут внедрены?

Выберите один ответ:

- ☒ а. Совершенно разные экземпляры *EntityManager*: разный *L1* кеш, разные транзакции и соединения с базой ✖
- Несмотря на то, что сами объекты *EntityManager* разные, фактически они представляют собой "один и тот же *Hibernate*". Будь иначе, каждый отдельный бин, использующий *EntityManager*, имел бы собственное соединение с БД и набор кешей, что, разумеется, нежизнеспособный вариант для любого боевого приложения, где счёт бинов часто переваливает за тысячи.
- ☐ б. Это один и тот же объект ($a == b$)
- ☐ в. Экземпляры *EntityManager* разные, но имеют один и тот же *L1* кеш, работают в одном соединении с БД
- ☐ г. По *Object#equals* – это один и тот же экземпляр

Ваш ответ неправильный.

Сервис `com.netcracker.ec.jpa.test.service.WillItSaveChanges` загружает из БД сущность с использованием Spring Data repository. Предполагаем, что нужный экземпляр существует и загружен. После этого сервис меняет значение одного из полей экземпляра и завершает работу. Будет ли изменение сохранено в БД? Выберите ответ с максимально точным обоснованием

Выберите один ответ:

- ☐ a. Да, Hibernate на границе транзакции проведёт *flush* всего, что изменилось в транзакции, и, следовательно, будет отмечено как *Dirty* процедурой *Dirty Checking*.
- ☐ b. Нет, первичный ключ сущности не поменялся
- ☐ c. Да, Hibernate проведёт *flush*, который обновит в БД все объекты, загруженные в транзакции, вне зависимости от того, менялись они, или нет
- ☒ d. Да, Spring Data неявно вызовет метод репозитория *save* при завершении работы любого метода, где использовался репозиторий. Это достигается проксированием метода *test* рассматриваемого класса. ✗ Вся формулировка вопроса - чистая фантазия автора. Не верьте на слово, проверяйте. Нет в Spring ничего подобного, никто без вашего ведома и разрешения *save* не вызовет, это же произвольный метод с неизвестной фреймворку логикой! Но, тем не менее, данные будут сохранены. Hibernate копит изменения доменной модели (любых размапленных полей) в *Persistence Context* и сохраняет всё изменённое в БД на границе транзакции. Граница транзакции явно определена аннотацией *@Transactional* на методе.
- ☐ e. Нет, транзакция *read-only*, никто ничего не сможет сохранить в БД
- ☐ f. Нет, метод репозитория *save* или его аналог не вызван

Ваш ответ неправильный.

Рассмотрите класс `com.netcracker.ec.jpa.test.entity.StrangeEntity`. Какие утверждения о нём **верны**?

Выберите один или несколько ответов:

- ☐ а. Аннотация `@Id` не требуется для первичного ключа, поле которого называется `id`
- ☒ b. Значение поля `payload` будет переписано значением из БД **✓** Hibernate сначала создаёт пустую сущность конструктором без аргументов, а затем начинает наполнять её поля значениями из соответствующих колонок БД. Что было до заполнения, не имеет значения. Поле будет переписано значением из БД.
- ☒ c. Это не корректная сущность: нет аннотации `@Table` **✗** Аннотация `@Table` вовсе не обязательна. Она **может** использоваться, чтобы переписать маппинг на таблицу по умолчанию, но если маппинг по умолчанию нас устраивает, можно обойтись и без неё.
- ☐ d. Первичный ключ задан некорректно: тип `UUID` не поддерживается Hibernate 5
- ☐ e. Аннотация `@Id` не требуется для поля типа `UUID`: тип поля, соответствующий идентификатору, даст Hibernate понять, что это поле – первичный ключ
- ☐ f. Поле `payload` всегда будет иметь значение A, поскольку инициализируется так при создании сущности
- ☐ g. Это не корректная сущность: не указано имя в аннотации `@Entity`
- ☒ h. Поле `thisWillBeA` всегда будет иметь значение A – оно финальное и технически не может быть изменено **✗** Кто победит: многолетний стандарт языка Java, или один маленький фреймворк? Шок, сенсация, кто бы мог подумать: `final` поля вовсе не `final`. С определённой версии Java `final` **может** быть изменён в рантайме. Hibernate закономерно пользуется этой возможностью. Поле будет переписано значением из БД.
Это крайне неочевидная ситуация, но проверить её достаточно легко - просто попробуйте!
- ☐ i. Значение поля `thisWillBeA` будет переписано значением из БД несмотря на то, что оно финальное
- ☒ j. Первичный ключ задан некорректно: нет аннотации `@Id` **✓** Без аннотации `@Id` Hibernate никак не сможет определить, какое из полей выражает первичный ключ. Эта аннотация обязательна.
- ☐ k. Аннотация `@Id` не требуется для первичного ключа, если на поле присутствует `@GeneratedValue`: так как последняя применяется только для ключей, Hibernate сам сделает вывод, что поле – первичный ключ

Ваш ответ неправильный.


Вопрос **4**

Верно

Баллов: 1,00 из 1,00

Класс `com.netcracker.ec.jpa.test.service.DeleterPersister` загружает сущность из БД (предполагаем, что экземпляр есть и загружен), производит над сущностью `delete` и затем `persist`. Какие запросы (помимо `select` при загрузке сущности) дойдут до БД?

Выберите один ответ:

- ☐ a. `delete` и `insert`
- ☐ b. `update`
- ☐ c. `delete` и `update`
- ☒ d. Никакие  Не дойдёт ни один. Фактическое удаление из БД произойдёт только в случае, если сущность доедет до `flush` в состоянии `deleted`. В данном сценарии она будет спасена до этого момента, а её поля не изменяются в сценарии. Поэтому ничего, кроме `select` на загрузке сущности, не произойдёт.

Ваш ответ верный.


Вопрос **5**

Верно

Баллов: 1,00 из 1,00

Рассмотрите сущность `com.netcracker.ec.jpa.test.entity.EntityWithEmbeddable`. Как должна выглядеть структура таблиц (или таблица), соответствующая такому маппингу?

Выберите один ответ:

- ☒ a. Одна таблица `EntityWithEmbeddable` с колонками `id`, `payload`, `width`, `height`, `r`, `g`, `b`  Верно. Вне зависимости от уровня вложенности, суть `Embeddable` в том, чтобы хранить все колонки в одной таблице. Всё съедет в основную таблицу сущности.
- ☐ b. Таблица `EntityWithEmbeddable` с колонками `id`, `payload`, `square`. Таблица `square` с колонками `width`, `height`, `color`. Таблица `Color` с колонками `r`, `g`, `b`.
- ☐ c. Таблица `EntityWithEmbeddable` с колонками `id`, `payload`, `squareId`. Таблица `square` с колонками `width`, `height`, `colorId`. Таблица `Color` с колонками `r`, `g`, `b`.
- ☐ d. Таблица `EntityWithEmbeddable` с колонками `id`, `payload`, `square`. Таблица `square` с колонками `width`, `height`, `r`, `g`, `b`.
- ☐ e. Никак – вложенные `Embeddables` не поддерживаются Hibernate
- ☐ f. Таблица `EntityWithEmbeddable` с колонками `id`, `payload`, `squareId`. Таблица `square` с колонками `width`, `height`, `r`, `g`, `b`.

Ваш ответ верный.


Вопрос **6**

Верно

Баллов: 1,00 из 1,00

Рассмотрите сущность `com.netcracker.ec.jpa.test.entity.EntityWithEnumeration`. Какой тип будет иметь колонка, соответствующая полю `state`, и что в ней будет храниться?

Выберите один ответ:

- ☐ a. Не будет колонки, маппинг неправильный: поле типа `Enum` без аннотации `@Enumerated` не допускается.
- ☒ b. `INTEGER` или аналог, соответствующий целочисленным данным, будет храниться значение в виде `ordinal enum`-а (1, 2 и т. д.)  По умолчанию Hibernate хранит `enum`-ы по их `ordinals` - порядковым номерам экземпляра `enum`-а в его классе.
- ☐ c. `VARCHAR` или аналог, соответствующий текстовым данным, будет храниться значение в виде имени `enum`-а (`DEAD`, `ALIVE`, и т. д.)

Ваш ответ верный.


Вопрос **7**

Верно

Баллов: 1,00 из 1,00

Рассмотрите класс `com.netcracker.ec.jpa.test.entity.EntityWithTemporal`. Какой тип будет иметь колонка, соответствующая полю `date`?

Выберите один ответ:

- ☐ a. `TIME` или аналог, соответствующий времени без даты
- ☒ b. `TIMESTAMP` или аналог, соответствующий максимальной точности представления (с временем)  Верно. Если не указано обратного, у Hibernate нет иного выхода, чем зарезервировать под колонку максимальную точность: ведь `Date` **может** хранить время.
- ☐ c. Не будет колонки, маппинг неправильный: используется поле типа `java.util.Date`, тогда как для представления даты-времени в сущностях Hibernate необходимо использовать `java.sql.Date`
- ☐ d. `DATE` или аналог, соответствующий дате, но без времени
- ☐ e. Не будет колонки, маппинг неправильный: поле `Date` без аннотации `@Temporal` не допускается

Ваш ответ верный.

Вопрос 8

Неверно

Баллов: 0,00 из 1,00

На каких полях можно использовать аннотацию `@GeneratedValue` (чтобы значение генерировалось)?

Выберите один ответ:

- ☐ a. `@GeneratedValue` не поддерживается Hibernate и не будет работать
- ☐ b. Только на первичных ключах любого поддерживаемого для первичного ключа типа
- ☐ c. Только на полях, которые являются примитивами (*int*, *float* и т. д.), а также *String*
- ☒ d. Только на первичных ключах числовых типов (*int/Integer*, *BigInteger*, *BigDecimal* и т. д.) ❌ *UUID*, например, тоже может генерироваться.
- ☐ e. Только на полях, которые являются примитивами (*int*, *float* и т. д.)
- ☐ f. Только на полях, которые являются примитивами (*int*, *float* и т. д.), а также поле любого *Serializable*-типа
- ☐ g. На любом поле, принадлежащем классу, помеченному `@Entity`

Ваш ответ неправильный.

Вопрос 9

Верно

Баллов: 1,00 из 1,00

Первичный ключ сущности имеет тип *UUID*. Будет ли в сочетании с ним работать `@GeneratedValue` (на Hibernate 5.1)?

Выберите один ответ:

- ☐ a. Да, если написать генератор специально для типа *UUID*
- ☐ b. Нет, генерация *UUID* не поддерживается: *UUID* не является числовым типом
- ☒ c. Да, «из коробки» ✅ Ни добавить, ни убавить, всё верно. В качестве дополнительного задания предлагаю выяснить, *UUID* какой версии генерится "из коробки", и чем версии отличаются друг от друга.

Ваш ответ верный.

Вопрос **10**

Неверно

Баллов: 0,00 из 1,00

Рассмотрите классы `com.netcracker.ec.jpa.test.entity.related.RelatedOne` и `RelatedTwo`. Какими колонками будет выражена связь между ними в соответствующих таблицах в БД?

Выберите один ответ:

- ☐ a. `relatedone.relatedtwo_id` и `relatedtwo.relatedone_id`
- ☒ b. `relatedone.relatedtwo` и `relatedtwo.relatedone` ✖ Почти правильно, но подвёл нейминг. Hibernate, если явно не указано другого, генерит (или ожидает в БД) имя колонки, составленное из имени сущности и имени поля, на которое мы ссылаемся. В данном случае не хватает `_id` у наименований.
- ☐ c. `relatedtwo.relatedone_id`
- ☐ d. `relatedone.relatedtwo_id`
- ☐ e. `relatedone.relatedtwo`
- ☐ f. `relatedtwo.relatedone`

Ваш ответ неправильный.

Вопрос **11**

Неверно

Баллов: 0,00 из 1,00

Рассмотрите классы `com.netcracker.ec.jpa.test.entity.related.RelatedOne` и `RelatedThree`. Изменится ли состав колонок, выражающих связь `OneToMany` между этими сущностями, если аннотацию `@JoinColumn` переместить на поле `relatedOnes` сущности `RelatedThree`?

Выберите один ответ:

- ☒ a. Да: теперь `RelatedThree` становится *owning side* и возьмёт на себя колонку с внешним ключом ✖ И где же будет храниться эта колонка, и как связь "один ко многим" будет выражена, если `id` связанных сущностей должно быть много?
- ☐ b. Нет: связь `ManyToOne` (`OneToMany`) всегда имеет сторону *one* как *owning side*, колонка останется там же несмотря на место упоминания аннотации `@JoinColumn`
- ☐ c. Невозможно сказать: из-за аннотации `@JoinColumn` на стороне, которая может в данном контексте выполнять только роль *inverse side*, Hibernate не может успешно стартовать.

Ваш ответ неправильный.

Вопрос 12

Неверно

Баллов: 0,00 из 1,00

Рассмотрите сущности `com.netcracker.ec.jpa.test.entity.related.RelatedOne` и `RelatedFour`. Какие из утверждений в отношении связи между сущностями через поля `relatedFours` и `relatedOne` верны?

Выберите один или несколько ответов:

- ☐ a. Операция *PERSIST* над *RelatedFour* распространится на связанную с ней сущность *RelatedOne*
- ☐ b. Операция *ALL* над *RelatedOne* распространится на связанные с ней сущности *RelatedFour*
- ☒ c. Поскольку *ALL* включает в себя *DELETE*, на связи будет исполняться *Orphan Removal* ❌ Нет, не будет. *Orphan Removal* логически связан с операцией *Delete*, но по умолчанию не включен, его нужно включать явно.
- ☒ d. Операция *PERSIST* над *RelatedOne* распространится на связанные с ней сущности *RelatedFour* ✔ Совершенно верно - как и следует из конфигурации.

Ваш ответ неправильный.

Вопрос 13

Верно

Баллов: 1,00 из 1,00

Рассмотрите классы `com.netcracker.ec.jpa.test.entity.related.RelatedOne`, `RelatedFive`, а также класс `com.netcracker.ec.jpa.test.service.CycledLoader`. Предполагая, что в БД находятся 200 сущностей *RelatedOne*, с каждой из которых связаны по 100 сущностей *RelatedFive*, сколько запросов в БД будет осуществлено в ходе работы метода `load` класса `CycledLoader`?

Выберите один ответ:

- ☐ a. 101: наблюдаем эффект проблемы *N+1*
- ☒ b. 201: наблюдаем эффект проблемы *N+1* ✔ Один запрос, чтобы прочитать *RelatedOne* и 200 чтобы прочитать *RelatedFive*.
- ☐ c. 300: наблюдаем эффект проблемы *N+1*
- ☐ d. Один – метод загрузки связанных сущностей для *OneToMany* по умолчанию *Eager*.
- ☐ e. $21 = 200/10 + 1$: при загрузке *Lazy-loaded* сущности в цикле автоматически неявно применяется `@BatchSize` с размером батча по умолчанию 10

Ваш ответ верный.

Вопрос **14**

Неверно

Баллов: 0,00 из 1,00

Рассмотрите класс `com.netcracker.ec.jpa.test.entity.VerboseEntity` и класс `com.netcracker.ec.jpa.test.service.ProjectionLoader`. Возвращаемое значение метода `load` представляет собой коллекцию. Что будут представлять собой элементы этой коллекции (предполагаем, что экземпляры сущности `VerboseEntity` есть в БД и их поля заполнены)?

Выберите один ответ:

- ☐ a. Каждый элемент массива – экземпляр сущности `VerboseEntity`, заполнены только поля, запрошенные в проекции
- ☐ b. Каждый элемент массива – экземпляр сущности `VerboseEntity`, заполнены все поля
- ☒ c. Каждый элемент коллекции представляет собой массив, каждый элемент массива – строка ✗ *JDBC достаточно умен, чтобы парсить результат в класс, максимально подходящий типу каждой колонки. Приведения всего к строкам не случится, остальное указано верно.*
- ☐ d. Результат не будет возвращён, возникнет `RuntimeException`, поскольку на самом деле запрос с проекцией возвращает на коллекцию, а массив
- ☐ e. Каждый элемент коллекции представляет собой массив, каждый элемент массива соответствует типу поля, запрошенного в проекции

Ваш ответ неправильный.

Вопрос **15**

Неверно

Баллов: 0,00 из 1,00

Рассмотрите класс `com.netcracker.ec.jpa.test.service.VerboseSearcher`, который использует репозиторный метод в `com.netcracker.ec.jpa.test.service.repo.VerboseEntityRepo`. Сколько JPQL-запросов, сгенерированных репозиторным методом, отправятся в базу в ходе работы метода `search`?

Выберите один ответ:

- ☐ a. Пять – каждый вызов будет провоцировать новый запрос в БД
- ☐ b. Один – первый же вызов репозиторного метода спровоцирует загрузку всех сущностей из БД и в дальнейшем операции будут проводиться в кеше Hibernate.
- ☒ c. Три – первые три вызова с одинаковым входным параметром спровоцирует лишь один запрос в БД – в первый раз – а в дальнейшем будут брать результат из кеша ✗ *Это только выглядит логично. Что, если между первым и вторым запросом содержимое БД изменилось, а мы взяли ответ из кеша? Кеширование включается явно, и здесь оно не включено. Есть пара хороших способов избежать повторения первых трёх запросов: спринговый `@Cacheable` на репозиторный метод; хибернейтовский `@Cache` на сущность + включенный кеш запросов. Но здесь не используется не один, поэтому запросов будут пять.*

Ваш ответ неправильный.

◀ [Материалы курса](#)

Перейти на...

