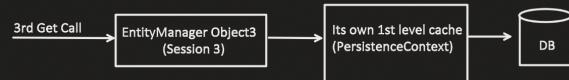
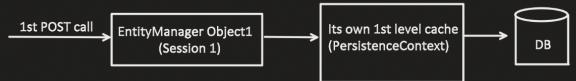


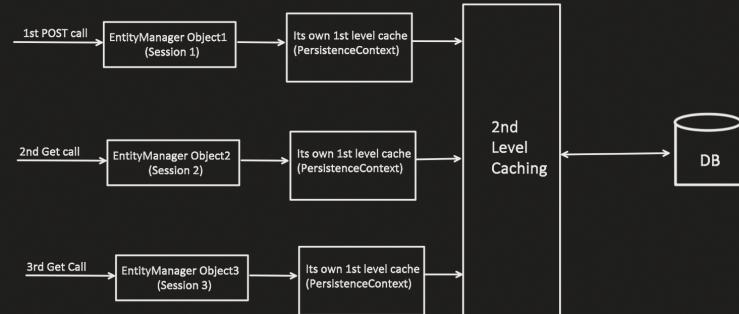
JPA (PART4) - Second Level Caching

From previous video, First level caching, we already know that, for each HTTP REQUEST, different EntityManager Object (session) is created and its have its own Persistence context (1st level cache)



Now, In **Second Level caching** or **L2 caching**, We will achieve something like this:

Report Abuse X



Lets first see, one happy flow, and see what all it takes to enable the 2nd level caching

pom.xml

```

<dependency>
    <groupId>org.ehcache</groupId>
    <artifactId>ehcache</artifactId>
    <version>3.10.8</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-jcache</artifactId>
    <version>5.5.2.Final</version>
</dependency>
<dependency>
    <groupId>javax.cache</groupId>
    <artifactId>cache-api</artifactId>
    <version>1.1.1</version>
</dependency>
  
```

application.properties

```

spring.jpa.properties.hibernate.cache.use_second_level_caches=true
spring.jpa.properties.hibernate.cache.region.factory_class=org.hibernate.cache.jcache.JCacheRegionFactory
spring.jpa.properties.java.cache.provider=org.ehcache.jsr107.EhcacheCachingProvider
logging.level.org.hibernate.cache.spi=DEBUG
  
```

```

@RequestController
@RequestMapping(value = "/api/")
public class Usercontroller {

    @Autowired
    UserDetailsService userDetailsService;

    @PostMapping(path = "/user")
    public UserDetails insertUser(@RequestBody UserDetails userDetails) {
        return userDetailsService.insertUser(userDetails);
    }

    @GetMapping("/user/{id}")
    public UserDetails getuser() {
        return userDetailsService.findById(primarykey: 1L);
    }
}
  
```

```

@Service
public class UserDetailsService {

    @Autowired
    UserDetailsRepository userDetailsRepository;

    public UserDetails saveuser(UserDetails user) {
        return userDetailsRepository.save(user);
    }

    public UserDetails findByID(Long primarykey) {
        return userDetailsRepository.findById(primarykey).get();
    }
}
  
```

```

@Cache(usage = CacheConcurrencyStrategy.READ_WRITE,
      region = "userDetailsCache")
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

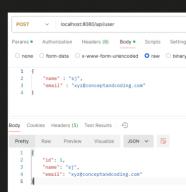
    // Constructors
    public UserDetails() {
    }

    public UserDetails(String name, String email) {
        this.name = name;
        this.email = email;
    }

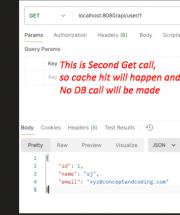
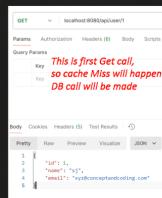
    // Getters and setters
}

```

- During Insert, data is directly inserted into DB, no Cache insertion or validation happens



- During Get, JPA will check, if data is present in cache? If Yes, its cache hit and return else its cache miss and it will fetch from DB and put into Cache.



```

2024-11-25T20:43:25.766+05:30 INFO 7872 --- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet      : Completed initialization in 1 ms
Hibernate:
    insert
        into
            user_details
        (email, name, id)
        values
        (?, ?, ?)
2024-11-25T20:43:27.706+05:30 DEBUG 7872 --- [nio-8080-exec-3] o.h.c.s.support.AbstractReadWriteAccess : Getting cached data from region ['userDetailsCache']
2024-11-25T20:43:27.707+05:30 DEBUG 7872 --- [nio-8080-exec-3] o.h.c.s.support.AbstractReadWriteAccess : Cache miss : region = 'userDetailsCache'
Hibernate:
    select
        ud1_0.id,
        ud1_0.email,
        ud1_0.name
    from
        user_details ud1_0
    where
        ud1_0.id=?
2024-11-25T20:43:27.729+05:30 DEBUG 7872 --- [nio-8080-exec-3] o.h.c.s.support.AbstractReadWriteAccess : Caching data from load [region:'userData']
2024-11-25T20:43:28.292+05:30 DEBUG 7872 --- [nio-8080-exec-4] o.h.c.s.support.AbstractReadWriteAccess : Getting cached data from region ['userDetailsCache']
2024-11-25T20:43:28.293+05:30 DEBUG 7872 --- [nio-8080-exec-4] o.h.c.s.support.AbstractReadWriteAccess : Checking readability of read-write cache
2024-11-25T20:43:28.295+05:30 DEBUG 7872 --- [nio-8080-exec-4] o.h.c.s.support.AbstractReadWriteAccess : Cache hit, region = 'userDetailsCache'

```

- Why in pom.xml, 3 dependencies required?

```

<dependency>
<groupId>org.ehcache</groupId>           → Provides the core implementation of Second level caching
<artifactId>ehcache</artifactId>
<version>3.10.8</version>
</dependency>

```

```

<dependency>
<groupId>org.hibernate</groupId>           → Hibernate specific Caching logic comes with this. Like we use Annotations over entity @Cache, we used CacheConcurrencyStrategy, so specific logic need to be executed, and this library help us with that.
<artifactId>hibernate-jcache</artifactId>
<version>6.5.2.Final</version>
</dependency>

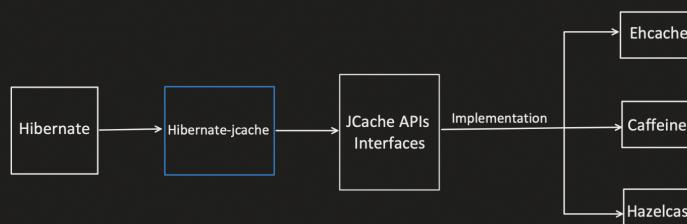
```

```

<dependency>
<groupId>javax.cache</groupId>           → Provides the interface for Jcache, hibernate interact with these APIs.
<artifactId>cache-api</artifactId>
<version>1.1.1</version>
</dependency>

```

Helps to achieve Loose coupling. We can change from Ehcache to some other Jcache compliant caching provider without changing code.



2. Lets understand application.properties and Region:

```
spring.jpa.properties.hibernate.cache.use_second_level_cache=true
spring.jpa.properties.hibernate.cache.region.factory_class=org.hibernate.cache.jcache.JCacheRegionFactory
spring.jpa.properties.java.cache.provider=org.ehcache.jsr107.EhcacheCachingProvider
logging.level.org.hibernate.cache.spi=DEBUG
```

This tell hibernate to use hibernate-jcache class to manage caching, we can also provide here direct ehcache factory class, means bypassing JCache interface

Region:

Helps in logical grouping of cached data.
 For each Region (or say group), we can apply different caching strategy like
 - Eviction policy
 - TTL
 - Cache size
 - Concurrency strategy etc.

Which helps in achieving granular level management of cached data (either Entity, Collection or Query results)

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE,
       region = "userDetailsCache")
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    // Constructors
    public UserDetails() {
    }

    public UserDetails(String name, String email) {
        this.name = name;
        this.email = email;
    }

    // Getters and setters
}
```

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE,
       region = "orderDetailsCache")
public class OrderDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String productName;
    private int quantity;
    private double price;

    // Getters and Setters
}
```

ehcache.xml
(file within "src/main/resources/" path)

```
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:noNamespaceSchemaLocation="http://www.ehcache.org/ehcache.xsd">

    <cache alias="userDetailsCache"
          maxElementsInMemory="100"
          timeToLiveSeconds="60"
          evictionStrategy="LIFO" />

    <cache alias="orderDetailsCache"
          maxElementsInMemory="1000"
          timeToLiveSeconds="200"
          evictionStrategy="FIFO" />
</ehcache>
```

3. Different **CacheConcurrencyStrategy**

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE,
       region = "userDetailsCache")
public class UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;
    private String email;

    // Constructors
    public UserDetails() {
    }

    public UserDetails(String name, String email) {
        this.name = name;
        this.email = email;
    }
}
```

S.No.	Strategy
1.	READ_ONLY
2.	READ_WRITE
3.	NONSTRICT_READ_WRITE
4.	TRANSACTIONAL

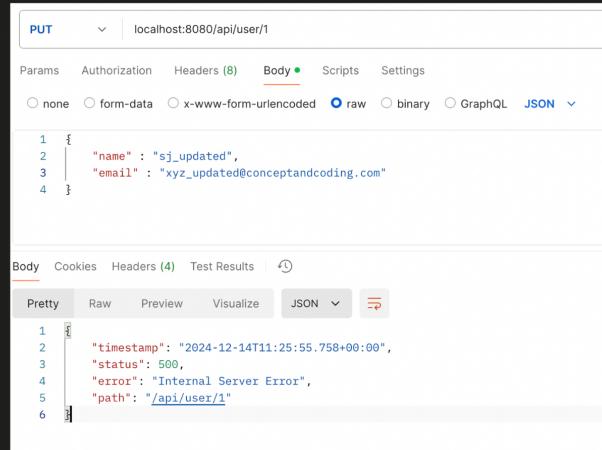
```
// Getters and setters  
}
```

1. READ_ONLY

- Good for Static Data
- Which do not require any updates
- If try to update just entity, exception will come

```
@RestController  
@RequestMapping(value = "/api/")  
public class UserController {  
  
    @Autowired  
    UserDetailsService userDetailsService;  
  
    @PostMapping(path = "/user")  
    public UserDetails insertUser(@RequestBody UserDetails userDetails) {  
        return userDetailsService.saveUser(userDetails);  
    }  
  
    @PutMapping(path = "/user/{id}")  
    public UserDetails updateUser(@PathVariable Long id, @RequestBody UserDetails userDetails) {  
        return userDetailsService.updateUser(id, userDetails);  
    }  
  
    @GetMapping("user/{id}")  
    public UserDetails getUser(@PathVariable Long primarykey) {  
        return userDetailsService.findById(primarykey);  
    }  
  
}  
  
@Entity  
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY,  
      region = "userDetailsCache")  
public class UserDetails {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    private String name;  
    private String email;  
  
    // Constructors  
    public UserDetails() {}  
  
    public UserDetails(String name, String email) {  
        this.name = name;  
        this.email = email;  
    }  
  
    // Getters and setters  
}
```

```
@Service  
public class UserDetailsService {  
  
    @Autowired  
    UserDetailsRepository userDetailsRepository;  
  
    public UserDetails saveUser(UserDetails user) {  
  
        return userDetailsRepository.save(user);  
    }  
  
    public UserDetails updateUser(Long id, UserDetails user) {  
        UserDetails existingUser = userDetailsRepository.findById(id).get();  
        existingUser.setName(user.getName());  
        existingUser.setEmail(user.getEmail());  
        return userDetailsRepository.save(existingUser);  
    }  
  
    public UserDetails findById(Long primaryKey) {  
        return userDetailsRepository.findById(primaryKey).get();  
    }  
}  
  
@Repository  
public interface UserDetailsRepository extends  
    JpaRepository<UserDetails, Long> {  
}
```



PUT localhost:8080/api/user/1

Params Authorization Headers (8) Body Scripts Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {  
2   "name" : "sj_updated",  
3   "email" : "xyz_updated@conceptandcoding.com"  
4 }
```

Body Cookies Headers (4) Test Results

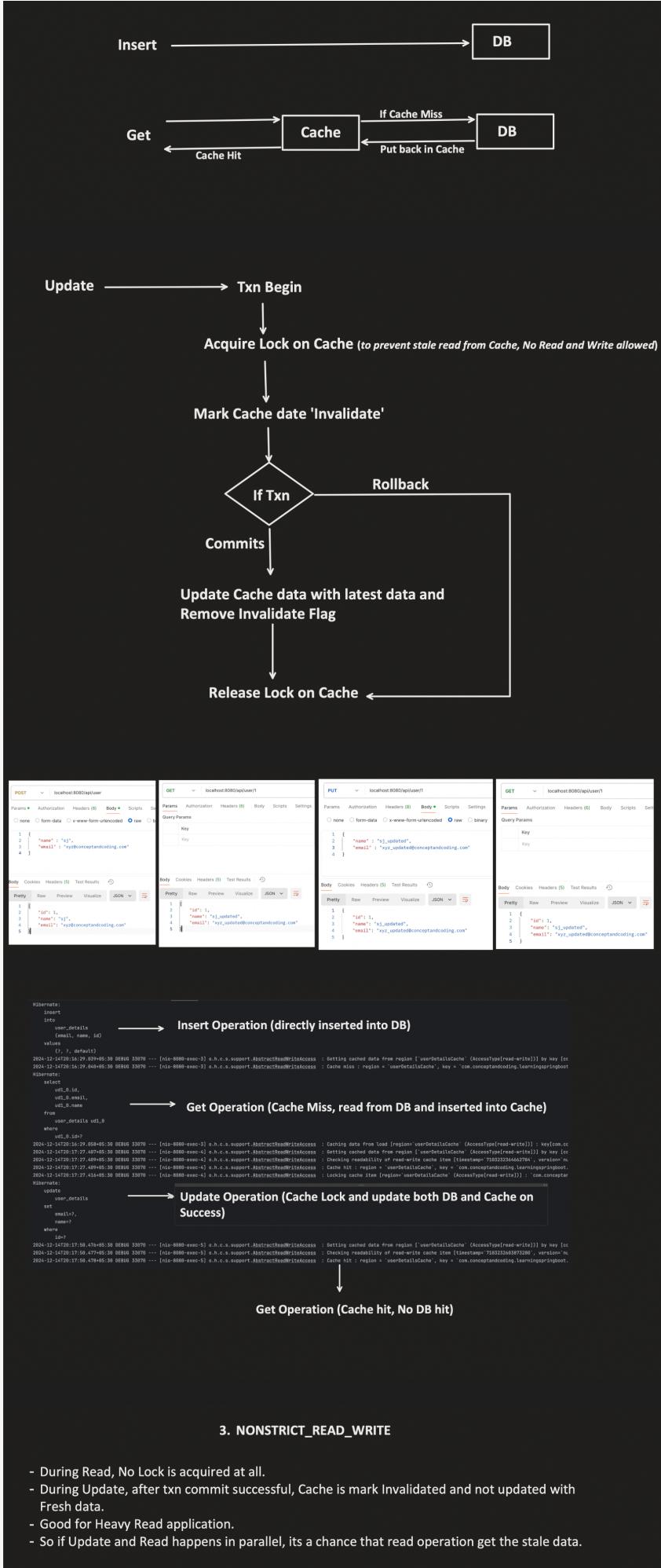
Pretty Raw Preview Visualize JSON

```
1 {"timestamp": "2024-12-14T11:25:55.758+00:00",  
2 "status": 500,  
3 "error": "Internal Server Error",  
4 "path": "/api/user/1"}  
5  
6
```

```
java.lang.UnsupportedOperationException: Can't update readonly object  
at org.hibernate.cache.spi.support.EntityReadonlyAccess.update(EntityReadonlyAccess.java:22) ~[hibernate-core-6.5.2.Final.jar:6.5.2.Final]  
at org.hibernate.action.internal.EntityUpdateAction.updateCache(EntityUpdateAction.java:32) ~[hibernate-core-6.5.2.Final.jar:6.5.2.Final]  
at org.hibernate.action.internal.EntityUpdateAction.updateCacheItem(EntityUpdateAction.java:220) ~[hibernate-core-6.5.2.Final.jar:6.5.2.Final]
```

2. READ_WRITE

- During Read, it put Shared Lock, other Read can also acquire Shared Lock. But no Write operation.
- During Update, it put Exclusive Lock, other Read and Write operation not allowed.



4. TRANSACTIONAL

- Acquire READ lock and Also WRITE lock.
- Updates the cache too, after txn commit successfully.
- Any other READ operation during cache lock, goes directly to DB.
- Any other WRITE operation during cache lock, waits in queue.