# جلسه اول

# جلسه دوم

اپلیکیشن سرور

Apache tomcat یکی از اپلیکیشن های جاوایی است. ضعیف ترین ولی محبوب ترین! زیرا کاربری آسان دارد.

وقتی در بستر وب بخواهیم سرویس دهیم لازم است از اپلیکیشن سرور استفاده کنیم زیرا بخشی از پردازش سمت وب انجام می شود.

نسخه اپلیکیشن سرور باید با نسخه جاوا همخوانی داشته باشد. مثلا نسخه جاوا 11 باید تامکت 9 استفاده کنیم. ورژن 11 به بعد باید از تامکت 10 باید استفاده کنیم. برای جاوا 8 هم باید از تامکت 8 استفاده کنیم.

در فایل تامکت یک پوشه bin است و در این پوشه یک فایل startup.bat وجود دارد. زمانی که این فایل را باز می کنیم اپلیکیشن سرور اجرا می شود. در صورت اجرا نشدن ممکن است java\_home اشتباه تعریف شده باشد و یا اینکه نسخه تامکت با جاوا همخوانی نداشته باشد. اگر این فایل باز شود و اجرا شود و بعد از مدتی بسته شود ممکن است مشکل در پورت داشته باشیم. در پوشه conf فایل server.xml داریم که داخل آن یک پورت ست شده است. در محیط عملیاتی به صورت پیش فرض پورت ما 80 است. اگر پورت 80 باشد نیازی به اعلام پورت نیست و صرفا با اعلام url یعنی بر روی پورت 80 اجرا می شود. اگر در browser بزنیم <http://localhost:8080/> آنگاه تامکت ران می شود و عکس یک گربه نمایش داده می شود.

پوشه lib در داخل تامکت که در آن کتابخانه های تامکت روشه.

پوشه logs که هر اتفاقی که روی سرور بیوفته اینجا لاگ میشه.

پوشه temp وقتی فایلی روی اپلیکیشن آپلود می کنیم، اول توی این پوشه میشینه و بعد که آپلود کامل شد به پوشه مورد نظر منتقل میشه.

پوشه webapps که در آن اپلیکیشن های ما قرار می گیرد.

پوشه work که در آن complie از jsp به servlet اینجا انجام می شود.

در صورتیکه اسم پروزه ذکر نکنیم یعنی root

در صورتیکه پورت اعلام نکنیم یعنی 80

اگه اعلام نکنیم کدام صفحه یعنی صفحه index

هم عرض فولدر root نباید فولدر دیگری بسازیم.

Virtual deployment

اتصال tomcat به intellij

یک پروژه جدید می سازیم سپس روی اسم پروژه میریم و دابل شیفت می زنیم سپس add framework support را می زنیم. در پنجره باز شده تیک web application را می زنیم. سپس اوکی می کنیم. سپس کنار دکمه ران وارد add configuration می شویم. در اینجا انواع اپلیکیشن سرورهای جاوا را نشان می دهد. ما بر روی tomcat local server کلیک می کنیم. سپس آدرس فولدر تامکت را ست می کنیم و fix می زنیم و سپس ران می کنیم اپلیکیشن سرور بالا می آید.

# جلسه سوم

هدر

یه سری اطلاعات به صورت ناخواسته با هر درخواستی که میدیم سمت سرویس دهنده ارسال می شود. که به آن header می گویند. کوکی ها هم بخشی از هدرها هستند. بعضی از هدرها هم ممکن است بروزرها ارسال نکن. یا در مد پرایوت یه سری اطلاعات ارسال نمی شود.

سشن

محفظه دوست داشتنی! هر کاربری که وارد نرم افزار وب بیس ما می شود اپلیکیشن سرور قبلش چک می کند آیا برای این کاربر سشن آیدی اختصاصی ایجاد کرده است یا خیر. در صورتیکه ایجاد نکرده باشد یک سشن آیدی جدید ایجاد می کند. درغیراینصورت خیر. این مورد را اپلیکیشن سرور از روی کوکی های روی هدر می فهمد. یعنی jsessionid اگر خالی باشد آنگاه جدید می سازد. یه رشته رندم تولید می کند و رویسشن ذخیره می کند. از این پس هردرخواستی که بر روی اپلیکیشن سرور ارسال می شود با اون سشن آیدی ارسال می شود.

به ازای هر سشن آیدی یک محفظه اختصاصی سمت رم سرویس دهنده ایجاد می شود. حالا برنامه نویس می تواند چیزی را روی سشن بخواباند و کاربر هر بار وارد می شود از اطلاعات آن استفاده کند. امنیتش بیشتر از کوکی است. زیرا محتوا سمت رم سرویس دهنده ذخیره می شود و فقط سشن آیدی سمت کلاینت ارسال می شود. البته حمله ای تحت عنوان session hijack وجود دارد. یعنی کاربر سشن شما را ربایش می کند. روی سشن هر نوع آبجکتی را می توان ذخیره کرد. پیشنهاد این است چیزی که روی سشن ذخیره می کنیم سریالایز شده باشد.

ری دایرکت response.sendRedirect("two.jsp");

در زمان ری دایرکت کردن از jsp 1 به روی jsp 2 اتفاقات زیر می افتد:

ابتدا کل پردازش صفحه 1 تمام می شود سپس ری دایرکت بر روی صفحه 2 انجام می شود. حتی اگر ری دایرکت را در ابتدای صفحه 1 انجام داده باشیم.

اگر در صفحه 1 تگ html داشته باشیم و بخواهیم در صفحه 1 چیزی نمایش دهیم این نمایش انجام نمی شود. یعنی کاربر پس از ری دایرکت فقط محتویات صفحه 2 را می بیند. صفحه 1 فقط پردازشش انجام می شود.

بعد از ری دایرکت اگر در صفحه 2 ریفرش بزنیم دیگر صفحه 1 اجرا نمی شود و فقط 2 اجرا می شود.

اگر در صفحه 1 به پارامتری در صفحه jsp دسترسی داشته باشیم در صفحه 2 به آن پارامتر دسترسی نخواهیم داشت و اگر بخواهیم پارامترهای 1 را به 2 منتقل کنیم باید آنرا مثلا روی سشن بخوابانیم.

Sendredirect در اصل server side redirect را انجام می دهد. یعنی کاربر اصلا متوجه آن نمی شود حتی اگر بک هم بزند به 1 برنمی گردد بلکه به صفحه قبل از 1 برمی گردد.

request.getRequestDispatcher("two.jsp").forward(request, response);

در حالت dispatch url صفحه روی همان 1 می ماند ولی محتوای صفحه 2 نمایش داده می شود.

در حالت قبلی یعنی ریدایرکت 1 کامل اجرا می شد سپس سراغ 2 می رفت و محتوای 2 را چاپ می کرد. ولی در dispatch انگار 2 در دل 1 اجرا می شود. یعنی در خطی که دیسپچ کنیم میره 2 را اجرا می کند و نمایش می دهد سپس سراغ ادامه 1 می رود.

وقتی رفرش می کنیم جفت صفحات با هم اجرا می شود ولی در ریدایرکت فقط 2 اجرا می شد.

Secutity policy رعایت نمی شود. یعنی پروزه در اپلیکیشن سرور یک دایرکتوری web-inf دارد که این مثل ناموس اپلیکیشن سرور می ماند. یعنی اگه بمیرد هم اجازه نمیده ریکوئست داخلش بیاد. چون خطرناکه و jar فایل های ما داخل آن قرار دارد. ولی با استفاده از dispatch می توان از یک jsp به یک jsp داخل فولدر web-inf ریکوئست فرستاد که این خطرناک است. در حالی که خود اپلیکیشن سرور اگر ما مستقیما درخواستی را به فایلی در داخل این فولدر بفرستیم جلوی ما را می گیرد و خطای not found می دهد.

request.getRequestDispatcher("two.jsp").include(request, response);

کاملا مشابه با dispatch.forward است فقط تفاوت اینجاست که در این حالت محتوای صفحه 1 هم چاپ می کند. یعنی انگار صفحه 1 را هم include می کند ولی در forward فقط صفحه 1 را پردازش می کند و چیزی از محتوای صفحه را چاپ نمی کند.

در دیسپچاینکلود اولویت بافر با صفجه تارگت است

در دیسپچ می توانیم به پارامترهای صفحه 1 نیز دسترسی داشته باشیم.

تفاوت عملکرد اپلیکیشن سرور در دو حالت دیسپچ و ری دایرکت در این است که در حالت دیسپچ برای صفحه 1 و صفحه 2 هر کدام آبجکت های request و response مجزا ایجاد می کند. مشابه با سایر صفحات. و به ازای هر ترد.

ولی در دیسپچ اپلیکیشن سرور برای این دو صفحه یک آبجکت request و response ایجاد می کند.

برای دستوراتی مثل select از دیسپچ استفاده می کنیم و برای execution مثل dml ها از ری دایرکت استفاده می کنیم.

یک محفظه ای تحت عنوان request attribute داریم که امن هم است ولی یک ایراد دارد فقط در صفحه بعدی به شرط dispatch قابل دسترس است. این attribute با سشن اتریبیوت فرق دارد.سشن اتریبیوت را کاربر تا زمانی که سشن اش منقضی نشده دسترسی وجود دارد ولی این ریکوئست اتریبیوت را صرفا در صفحه بعد به شرط دیسپچ دسترسی دارد.

ریکوئست اتریبیوت یک hashmap non static است در داخل کلاس httpservlet request.

سشن یک hashmap static است که کلید آن میشه سشن آیدی و value آن می شود یک hashmap دیگر که اتریبیوت های این سشن آیدی داخلش است.

سشن را نباید سنگین کرد.

سشن بعد از 30 دقیقه منقضی می شود. این تایم را می توان در داخل تنظیمات web.xml تغییر داد. Session config session timeout.

Servlet

تمام تکنولوژی های جاوا زیرشون servlet است. حتی خود jsp نیز به servlet کامپایل می شود.

Jsp یک صفجه html بود که می توانستیم داخلش تگ های scriplet تعریف و جاوا بنویسیم داخلش.

ولی servlet یک کلاس جاوایی است که می توانیم داخلش محتوای وب روی صفحه نمایش بدیم و پرینت کنیم.

در اصل لایه کنترلر ما را servlet تشکیل می دهد.

برای تعریف یک servlet در جاوا باید کلاسی بسازیم که از hhtpservlet extend کند و همچنین متد service انرا override کند.

که در متد service دو ورودی از جنس httpservletrequest و httpservletresponse داریم. این اشیا را servlet container در داخل اپلیکیشن سرور می سازد و به servlet پاس می دهد.

حالا اینکه کی این متد سرویس اجرا شود را باید تعریف کنیم.

دو راه برای این کار وجود دارد. یکی می توان در داخل web.xml servlet و servlet mapping تعریف کنیم.

<servlet>  
 <servlet-name>x</servlet-name>  
 <servlet-class>org.j2os.controller.Hello</servlet-class>  
</servlet>  
  
<servlet-mapping>  
 <servlet-name>x</servlet-name>  
 <url-pattern>/javad.do</url-pattern>  
</servlet-mapping>

در واقع زمانی که درخواستی بر روی javad.do ارسال شود اپلیکیشن سرور یک اینستنس x از کلاس hello می سازد و متد service آن اجرا می شود.

وظایف servlet container

ساخت اینستنس از کلاس servlet

مپ کردن اینستنس به یک url خاص.

ساخت اینستنس های request response

فراخوانی متد service از کلاس servlet

هم چنین می توان به جای تعریف مپینگ در web.xml بیایم در داخل کلاس servlet از انوتیشن @WebServlet("/javad.do") استفاده کنیم.

Servlet از نظر پرفورمنس خیلی خوبه ولی مشکلی که دارد این است که stateless است.

# جلسه چهارم

Artifact در واقع فرآیند deployment است. در project structure وجود دارد. Web application و یا java ee را لازم داریم.

یک حالت archive دارد یک حالت exploded. در حالتیکه بخواهیم real deployment انجام دهیم بر روی archive می گذاریم. اگر بخواهیم پروژه را با intellij بالا بیاوریم روی exploded می گذاریم. تیک include in project build هم می زنیم.

متد init متدی از کلاس Servlet است که فقط اولین بار در هر بار بالا آمدن servlet اجرا می شود. یعنی فقط با ریکوئست اول روی آن Servlet اجرا می شود.

و وقتی که اپلیکیشن سرور دستی استاپ می شود متد Destroy کال می شود. پس Destroy متد قابل اطمینانی نیست زیر اگر سرور به دلایل دیگری استاپ شود اجرا نخواهد شد. و فقط با استاپ کردن دستی این اتفاق می افتد.

می توان در داخل web.xml یک سری پارامتر تعریف کرد مثلا context-param. حالا در متد init از طریق instance config به آن دسترسی خواهیم داشت.

Jstl

خیلی از تگ های jstl deprecate شده است.

ابتدا باید کتابخانه های آن را اضافه کنیم.

در jsp حق نداریم چیزی را set کنیم و فقط پارامترهارا get می کنیم. از scriplet استفاده نمی کنیم زیرا دست برنامه نویس را باز می گذارد و معماری mvc را رعایت نمی کند.

در معماری mvc request based ویو باید در داخل اپلیکیشن سرور باشد. . ولی در معماری mvc api based ویو ما ربطی به کار بک اند کار ندارد و آنرا فرانت اند کار می نویسد. ولی در request based لایه ویو را ما و فرانت اند کار با هم می نویسیم.پس در معماری mvc api based بحث فریمورک های انگولار و ری اکت و ناکس js اینها مطرح است و در معماری request based دیگر اینها مطرح نیست.

جلسه پنجم

Hibernate

Orm

Object relational mapping framework

به ازای هر ردیف یک شی از کلاس می ساختیم دیدگاه رابطه ای است. همچنین اگر روابط مان هم از نوع رابطه ای باشد می شود orm

به جهت راحتی کار با orm های مختلف جاوا و یکپارچگی استانداری تحت عنوان jpa تعریف کرده اند. Java persistence api

Jpa فقط یه مشت annotation و interface است. و همه orm ها این استاندارد را پیاده سازی کرده اند و ما کافیست همین استاندارد را استفاده کنیم و کدمان دچار تغییر نمی شود حتی اگر بخواهیم فریمورک orm را تغییر دهیم.

کجا از hibernate استفاده نکنیم؟

Fetch لحظه ای سنگین. یک داده سنگینی در سطح دیتابیس دارم و نمی خوام خرد خرد بیاد میخوام همش با هم بیام. مثلا در شبکه عصبی می خوایم حجم زیادی از دیتا را در یک مرحله به مدل بدهیم تا یاد بگیرد. اینجا fetch لحظه ای سنگین داریم و لازم است کل دیتا در یک بار fetch کردن خوانده شود در این حالت استفاده از hibernate صحیح نمی باشد. اینجا یا باید بریم سراغ batis و یا jdbc. اگر بخواهیم به ازای هر ردیف یک شی بسازیم مموری بسیار زیادی اشغال می کنیم. اگر بخواهیم از jdbc استفاده کنیم با خود resultset کار کنیم.

پس بیشتر گرایش hibernate در سیستم های transactional است نه warehouse.

ما hibernate را لخت یاد نمی گیریم بلکه آنرا integrated با jpa یاد می گیریم.

از hibernate 5 شروع می کنیم.

لازم است کتابخانه های hibernate را به پروژه اضافه کنیم. زیر hibernate نیز همان jdbc است.

میتوان در هنگامی که از hibernate استفاده می کنیم دیگر از repository استفاده نکنیم. در حقیقت نقش repository را hibernate استفاده می کند. در پوشه src باید پوشه های به اسم meta-inf ایجاد کنیم که در داخل آن باید یک فایل persistence.xml را اضافه کنیم. این فایل به توجه به اینکه orm framework ما چی هست تغییر می کند.

Transaction\_type = resource\_local

در این فایل user و pass و آدرساتصال به دیتابیس وارد می شود.

منظور از hibernate dialect همان زبان sql دیتابیسی است که به آن وصل می شویم. مثلا برای اوراکل می توان Oracle12cDialect را وارد کرد.

ورژن dialect همیشه باید پایین تر یا برابر با ورژن خود دیتابیس باشد.

با استفاده از show\_sql می توان دستورات دیتابیسی که صادر می شود را دید.

c3p0 نام connection pool مربوط به hibernate است.

HBM2DDL

Create-drop برای این است که خود hibernate تیبل ها را می سازد.

Create: دراپ کردن اسکیما و ساخت یک اسکیمای جدید

Update: آپدیت کردن اسکیما بر اساس انتیتی

Validate: هیچ تغییری را اعمال نمی کند و فقط چک میکند که ساختار دیتابیس با انتیتی یکسان باشد.

None: هیچ کاری نمیکنه.

در سطح entity می توان با استفاده از annotation های زیر مواردی را تعریف کرد.

@table نام تیبلی که می خواهیم در دیتبایس با این انتیتی در ارتباط باشد.

برای هر یک از پراپرتی ها می توان یک @column گرفت و در آن نام و دیتا تایپ columnDefinition هر فیلد را مشخص کرد. هم چنین nullable و unique بودن را نیز می توان مشخص کرد. در حالت پیش فرض فیلد از نوع string را varchar2(255) تعریف می کند.

برای استفاده از این فریمورک بهتر است یک کلاس jpa تعریف کنیم و در داخل آن متد GetEntityManager را بنویسیم.

EntityManagerFactory

EntityManager

EntityTransaction

manager.persist(person)

اینسرت کردن

manager.find(Person.class, 1L);

پیدا کردن بر اساس آیدی

manager.find(Person.class,1l)

person.setName("HAMID");

person.setFamily("HAMIDI");

در این حالت آپدیت انجام می شود!

manager.merge(person);

آپدیت کردن با استفاده از merge. در این حالت بدون find کردن مستقیم آپدیت می کند. این متد خطرناک است و در صورتیکه بخواهیم از آن استفاده کنیم باید از Record Version استفاده کرده باشیم.

متد persist اگر بهش آیدی بدیم آپدیت میکنه اگر ندیم اینسرت میکنه.

هایبرنت میگه میخوای آپدیت کنی. باید حتما آیدیش رو ست کنی و بهم بدی. یه شرط دیگه اینه که شی رو باید از خودم تحویل گرفته باشی. یعنی اول باید آنرا find کنی بعدش بیای persist بزنی تا برات آپدیت کنم. اتفاقا این یک مزیت خوب است زیرا در زمان آپدیت مجبور می شویم از آخرین تغییرات باخبر شویم. ممکن است رکورد وجود نداشته باشد و یا حتی آن شی تغییر کرده باشد. اگر بخواهیم بدون find کردن آپدیت کنیم آنگاه به خطای detached entity passed to persist برخورد می کنیم.

اگر شی میخوای آپدیت کنی از خودم بگیر به خودم بده.

لاک در دیتابیس 3 سطح دارد:

لاک در سطح row:

لاک در سطح table:

لاک در سطح دیتابیس:

دیتابیس اوراکل بای دیفالت لاکش در سطح row اتفاق می افتد.

یک سری موارد لاک در سطح دیتابیس نیست و لازم است در اپلیکیشن پیاده سازی شود. مثلا یک آبجکت را من به دو نفر دادم مثلا json را return کردم. نفر اول اومده اون شی رو از من گرفته و آپدیت کرده و داده به من آپدیت کردم تموم شده رفته و کامیت شده و transaction هم بسته شده. ولی آبجکتی که دست نفر دوم است، تغییر جدید بر روی آن اعمال نشده است. چون اصلا نفر دوم با من ارتباطی نداشته است. حالا نفر دوم آبجکت را تغییر می دهد و مجدد به من میفرستد. من باید بگم عذر میخوام الان دیگه وضعیت تغییر کرده و مثلا یکی دیگه بلیط رو خرید رفت. اون آبجکتی که من بهت دادم الان دیگه مثل قبل نیست.

متد merge خطرناک است زیرا سراغ آخرین تغییرات نمی رود. در صورت استفاده از این متد باید از record version استفاده کرد.

در سطح اپلیکیشن هندل کردن این موارد سه روش دارد:

Timeout: تایم اوت یعنی شما هر دوتاتون اون رکورد رو میخواید فقط به یکی تون میدم شما مثلا 60 ثانیه وقت داری تغییرات رو اعمال کنی اگه اعمال کردی که من تغییرات رو میدم اگرنه قفل برداشته می شود و رکورد در اختیار نفر دوم قرار می گیرد.

Compare: یعنی نفر اول آپدیت می کند من آپدیت را می گیرم سپس نفر دوم هم آپدیت می کند. من آخرین وضعیت جدید را به نفر دوم نشون میدم و میگم این تغییرات روش اعمال شده. ورژن قبلی و ورژنی که کاربر اول تغییر داده. سپس از نفر دوم می پرسم کدوم رو صلاح میدونی انجام بشه. ای ن برای مواقعی است که کاربران هم سطح هستند و مثلا کارشناس مالی هستن و دارن روی یک پرونده مالی تغییرات اعمال میکنند. و مسئله رقابتی نیست.

Record version: (optimistic lock) در زمانی که رقابتی است استفاده می شود. این معمولا بهترین راه است. رکوردی که نفر اول و نفر دوم می خواد رو من به هر دو تا نفر میدم. هر کی زودتر آپدیت کرد آپدیتش پذیرش میشه و هر کی دیرتر آپدیت کرد آپدیتش پذیرش نمیشه. مثلا شما تو فروشگاه آنلاین جنس رو انتخاب کردی سبد کالا رو پر کردی میری جلو نهایتا ولی اجازه نهایی نکردن را نمی دهد. هایبرنیت خودش این مورد را می تواند هندل کند. با تعریف یک پراپرتی version و گذاشتن @version روی آن در entity می توان بر روی آن جدول و انتیتی رکورد ورژن را هندل کرد.

در متد remove نیز در صورتیکه find نکنیم خطای detached exception می خوریم.

در صورتیکه یک شی را find کنیم سپس بر روی آن تغییری اعمال کنیم سپس transaction.commit بزنیم حتی بدون آنکه قبل از آن متد merge یا persist را زده باشیم تغییرات در دیتابیس اعمال می شود.

سیاست هایبرنیت: اگر transaction ای را begine کردی سپس آنرا کامیت کردی و در داخل آن شی ای را از من تحویل گرفتی و بر روی آن شی تغییری دادی حتی اگر آنرا به متدهای persist و merge هم پاس نداده باشی من خودم آنرا در دیتابیس آپدیت می کنم.

اگر بخواهیم آپدیتی صورت نگیرد یا می توان transaction را حذف کرد و یا اینکه آن شی را detach کنیم.

از نظر هایبرنیت آبجکتی attach است و می تواند روی آن تغییرات اعمال کند که خودش آنرا به ما داده باشد. اگر آبجکت detach شده باشد آنگاه در هنگام persist خطا می دهد. ولی اگر merge کنیم این مورد دور زده می شود و انگار هایبرنیت دیگر این مورد را چک نمی کند که آیا شی attach هست یا نیست.

Hibernate همیشه hashcode آبجکت هایی که به ما می دهد را نگه داری می کند. و از روی آن تشخیص می دهد که کدام آبجکت detach شده یا attach می باشد. وقتی ما detach انجام می دهیم hashcode آن از هایبرنیت حذف می شود.

Jpql زبان کوئری jpa است. در این زبان با نام انتیتی ها کار می کنیم.

نکته: در صورتیکه اپلیکیشن فرم ساز بخواهیم بسازیم درست نیست که ساختاری داشته باشیم که مشتری هر سری به ازای هر فرمی که می خواهد ما بخواهیم تیبل جدید بسازیم. اگر چنین فرم سازی بخواهیم می توانیم 3 جدول به نام های tablenames و fieldnames و values بسازیم. و یا اینکه بهتر است بریم سراغ noSQL چونکه داده ما structure مشخصی ندارد و ممکن است قابل پیش بینی نباشد لذا بهتر است در این حالت از noSQL استفاده کنیم.

استفاده از namedQuery: می توان کوئری های مرتبط با یک entity را در بالای سر کلاس آن entity تعریف کنیم.

در نسخه 6 هایبرنیت می توان با خیال راحت از nativequery استفاده کرد. ولی در نسخه های قبلی بهتر است از همان createquery استفاده کنیم.

زیرا استفاده از nativequery بار اضافی ایجاد میکند. پروژه که بالا میاد یک بار با استفاده از کد رفلکشن ساختار کل کلاس و انتیتی ها رو میکشه بیرون. وقتی کد jpql می نویسیم از دانش first scan استفاده می شود. هزینه آن فقط تبدیل jpql به sql است. ولی وقتی از sql استفاده کنیم آنگاه از دانش first scan استفاده نمی کنیم. ولی در هایبرنیت 6 دیگر تفاوتی بین jpql و sql وجود ندارد.

از @version برای رکورد ورژنینگ استفاده می کنیم. همان optimistic lock است.

Pessimistic lock برای قفل گذاشتن روی جدول است. در واقع همان select for update است. از این لاک در کور بنکینگ استفاده می شود. مثلا جدول balance را در کوربنکینگ همان کار را می کنند. این قفل را نباید خیلی استفاده کنیم. به آن قفل بدبینانه هم گفته می شود. در jpa هم می توان بر روی یک انتیتی قفل pessimistic lock گذاشت. لازم است قفل در جدول بالانس گذاشته شود که اجازه ندهیم دو تراکنش همزمان روی یک حساب انجام شود. یک راهکار دیگر message service است مثلا کافکا یا activemq یا rabitmq. در این حالت ریکوئست ها رو کانتینر مثلا کافکا می رود.

انواع رابطه ها

One to one

Many to one

One to many

رابطه many to many را نی گیریم زیرا افزونگی داده ایجاد می کند و در واقع لازم است این رابطه با تعریف یک جدول دیگر به دو رابطه one to many و many to one شکسته شود.

ولی در واقع فقط یک رابطه one to many داریم.

رابطه one to one محض هم هیچ وقت نداریم. در واقع زمانی رابطه از نوع one to one است که دو تا جدول به همدیگه وصل بشن و نقش primary key های یکدیگر را بازی کنند. در این حالت اصلا نیازی نیست که دو جدول تعریف کنیم و اصلا باید فقط یک جدول بگیریم و چرا بخواهیم جدول جدید بگیریم! نرمال سازی اضافی نباید انجام بدیم زیرا سیستم را کندتر می کند. حجم داده را با نرمال سازی کاهش می دیم ولی بار و زمان پاسخ را افزایش می دیم. نرمال سازی کامپرس سازی داده با لول 1 است. مثلا در سیستم بانک نرمال سازی خیلی توجیهی ندارد زیرا مشکل حجم دیتا نیست بلکه زمان پاسخ اهمیت دارد. راهکارهای دیگری برای کاهش حجم دیتا مثل پارتیشن کردن در سطح column انجام داد.