UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY
A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2024 Artem Belyshev

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Optimalizace výkonu SQL relačních databází v kontextu Java aplikací

Belyshev Artem

Bakalářská práce

2024

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2023/2024

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Artem Belyshev**
Osobní číslo:	**I21319**
Studijní program:	**B0688A140009 Informační technologie**
Téma práce:	**Optimalizace výkonu SQL relačních databází v kontextu Java aplikací**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je analýzy a optimalizace výkonu SQL databází s využitím Java aplikací, zejména s využitím Hibernate a Spring Data JPA. Optimalizace bude provedena, jak na datové vrstvě, tak i ve způsobu kommunikace samotné aplikace. Podcílem práce je průzkum různých optimalizačních technik SQL databází a vhodné využití Hibernate a Spring Data JPA k jejich implementaci. Práce také popíše a uvede nejčastější problémy spojené s výkonem databázových aplikací a představí možná řešení pro jejich elimanici.

Rozsah pracovní zprávy: **min. 30 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

WINAND, M. SQL Performance Explained: Everything Developers Need to Know about SQL Performance M.Winand, 2012, ISBN 978-3950307825
CELKO, J. Joe Celko's Trees and Hierarchies in SQL for Smarties, Morgan Kaufmann, 2012, ISBN 978--0123877338

Vedoucí bakalářské práce: **Ing. Monika Borkovcová, Ph.D.**
Katedra informačních technologií

Datum zadání bakalářské práce: **15. prosince 2023**
Termín odevzdání bakalářské práce: **10. května 2024**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2024

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 10.5.2024

Artem Belyshev

Poděkování

Chtěl bych vyjádřit poděkování Ing. Monice Borkovcové, Ph.D. za její trpělivost, užitečné rady a odborné vedení během konzultací k této bakalářské práci.

Anotace

Cílem bakalářské práce je analýza a optimalizace výkonu SQL databází s využitím Java aplikací, zejména s využitím Hibernate a Spring Data JPA. Optimalizace bude provedena, jak na datové vrstvě, tak i ve způsobu komunikace samotné aplikace. Dalším podružným cílem práce je průzkum různých optimalizačních technik SQL databází a vhodné využití Hibernate a Spring Data JPA k jejich implementaci. Práce také popíše a uvede nejčastější problémy spojené s výkonem databázových aplikací a představí možná řešení pro jejich eliminaci.

Klíčová slova

SQL, Normalizace, Indexace, Výkon, Java Spring Boot, Hibernate, JPA, Apache JMeter, Postman, Prometheus, Grafana

Title

Optimizing the performance of SQL relational databases in the context of Java applications

Annotation

The aim of the bachelor thesis is to analyze and optimize the performance of SQL databases with the use of Java applications, especially with the use of Hibernate and Spring Data JPA. Optimization will be carried out both on the data layer and in the communication method of the application itself. The sub-goal of the thesis is the research of various optimization techniques of SQL databases and the appropriate use of Hibernate and Spring Data JPA for their implementation. The work will also describe and list the most common problems associated with the performance of database applications and present possible solutions for their elimination.

Keywords

SQL, Normalization, Indexing, Performance, Java Spring Boot, Hibernate, JPA, Apache JMeter, Postman, Prometheus, Grafana

Obsah

Seznam obrázků

Seznam zdrojových kódů

Seznam zkratek

SQL	Structured Query Language
JPA	Java Persistence API
CRUD	Create, Read, Update, and Delete operations
XML	Extensive Markup Language
RDBMS	Relational Database Management System
JPQL	Jakarta Persistence Query Language
ORM	Object–Relational Mapping
JVM	Java Virtual Machine
JDBC	Java Database Connectivity
API	Application Programming Interface
HTTPS	Hypertext Transfer Protocol Secure
HTTP	Hypertext Transfer Protocol
APDEX	Application Performance Index
PHP	Hypertext Preprocessor
CI/CD	Continuous Integration and Continuous Delivery
CPU	Central Processing Unit
LDAP	The Lightweight Directory Access Protocol
PromQL	Prometheus Query Language
OAuth	Open Authorization
PL/SQL	Procedural Language for SQL
SID	Oracle System Identifier
IP	Internet Protocol
ASM	Automatic Storage Management
PK	Primary Key
FK	Foreign Key
DTO	Data Transfer Object

Úvod

Cílem této bakalářské práce je analyzovat a optimalizovat výkon SQL databází pomocí Java aplikací se zaměřením na využívání technologií Hibernate a Spring Data JPA, zvýšit rychlost a efektivitu databázových operací a najít nejlepší techniky optimalizace SQL dotazů pro různé objemy dat.

V dnešní době hrají IT technologie klíčovou roli ve všech odvětvích, proto neustále je potřeba zvyšovat jejich efektivitu a rychlost. V této práci bude autor zkoumat různé techniky, které mohou pomoci optimalizovat výkonnost databáze, a zaměří se také na to, jak lze tyto techniky nejlépe implementovat pomocí Hibernate a Spring Data JPA. Důležitou součástí této práce je také identifikovat běžné problémy, které mohou ovlivnit výkonnost databázových aplikací, a navrhnout možná řešení k jejich odstranění. Vzhledem k rychlému vývoji databázových technologií a programových frameworků je tato práce aktuální a přináší nové pohledy do stále se vyvíjející oblasti IT. Zabývá se teoretickými i praktickými aspekty optimalizace a poskytuje ucelený pohled na to, jak lze zlepšit interakci mezi aplikacemi v jazyce Java a databázemi SQL.

Práce je strukturována tak, aby čtenáře postupně provedla teoretickými základy až k praktickým aplikacím a testování různých optimalizačních technik. Tímto přístupem se autor snaží poskytnout komplexní představu o možnostech moderních technik zvyšování výkonu databází a jejich praktickém využití ve programování.

Teoretická část této práce se bude zaměřovat na základní principy SQL a specifika Java frameworků, jako je Spring, které poskytují různé nástroje a techniky pro zvýšení výkonnosti databázových operací. Budou prozkoumány klíčové aspekty, jako je efektivní design databázových schémat, a indexace. Pozornost bude věnována také využití nástrojů jako Apache Jmeter, Grafana pro testování a zlepšování výkonnosti, což umožnilo analýzu a měření výkonnosti aplikací.

Během praktické části bude použit Apache JMeter pro posouzení výkonnosti skutečné Java Spring aplikace. Bude provedeno porovnání statistik databázových dotazů pomocí nástroje Grafana před a po zavedení optimalizačních metod, aby bylo možné posoudit, jak modifikace v konfiguraci a kódu ovlivňují výkonnost a odezvu databáze. Tímto způsobem bude možné získat kvantitativní údaje pro ilustraci přínosů konkrétní optimalizační techniky a pro lepší pochopení jejího dopadu na výkonnost aplikace.

1. Spring Data JPA

1.1. Historie

Java Spring Framework je moderní, výkonný a všestranný framework, který optimalizuje proces vývoje a zvyšuje rychlost tvorby aplikací. Jako součást tohoto frameworku byla v roce 2011 vydána první verze Spring Data JPA s cílem rozšířit portfolio projektů Spring a usnadnit práci s aplikacemi s vysokým zatížením. Tento modul si klade za cíl vytvoření efektivních aplikací prostřednictvím zjednodušení procesu vývoje a podpory. Hlavní myšlenkou bylo vytvořit úroveň abstrakce, která by usnadnila a optimalizovala přístup ke zpracování dat a provádění CRUD operací.

1.2. Využití

Implementace vrstvy pro přístup k datům není snadný úkol, Spring Data JPA si klade za cíl zlepšit implementaci minimalizací úkonů vedoucím k datům, tzn. snížením psaní standardního kódu díky použití JPA rozhraní. Zaměřuje se tedy na vynaložení úsilí, které je skutečně nutné pro řešení již konkrétních problémů týkajících se přímo řešení konkrétní oblasti. [1] JPA nebo Java persistence je specifikace Java, která umožňuje na základě názvů tříd Java provádět přidružení odpovídající databázovým tabulkám. Proces vytváření vrstvy pro přístup k databázi se provádí vytvořením rozhraní úložiště. JPA v současném vývoji se používá ve spolupráci s frameworkem Java Spring a dále v kombinaci s Hibernate, což usnadňuje mapování Java tříd na tabulky z databáze. Pro konfiguraci úložišť v Spring Data JPA lze použít JavaConfig nebo XML jmenný prostor. JavaConfig umožňuje definovat nastavení pomocí Java kódu, což zahrnuje specifikaci bean komponent, repozitářů, a zdrojů dat přímo v kódu aplikace. Konfiguraci lze zapsat v anotovaných třídách s použitím `@Configuration` a dalších souvisejících anotací, což umožňuje flexibilní řízení závislostí a konfiguračních parametrů. Na druhou stranu, XML jmenný prostor nabízí alternativu v podobě XML souborů, kde lze definovat beany a repozitáře v deklarativní formě. Tento přístup spočívá ve vytvoření XML souboru s definicemi bean, které jsou poté načítány Spring kontejnerem.[2]

1.3. Definování entit

Entita v JPA představuje tabulku, která je uložena v databázi, a každá instance entity je záznam v tabulce.

1.3.1. Anotace entity

Například, pokud existuje třída `Vehicle`, která obsahuje informace o vozidle, a cílem je tuto třídu propojit s podobnou tabulkou v databázi, je nutné definovat entitu, aby JPA mohl tuto třídu rozpoznat a přímo ji propojit s tabulkou v databázi. K určení Entity je třeba použít notaci `@Entity` na úrovni třídy a také se ujistit, že entita má konstruktor bez argumentů a primární klíč. Každý objekt JPA musí mít primární klíč, který je nezbytný pro jeho jednoznačnou identifikaci. K tomu slouží notace `@Id`. Pomocí `@GeneratedValue` je možné vybrat strategii generování primárního klíče, která se musí shodovat se strategií generování primárního klíče pro konkrétní tabulku v RDBMS.

```
@Entity
@Table(name = "VEHICLE", schema = "FACTORY")
public class Vehicle {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @NotNull
    @Column(name = "VEHICLE_ID", nullable = false)
    private Long vehicleId;
```

Zdrojový kód 1 – Implementace Spring Data JPA (zdroj: vlastní)

V případech, kdy se název třídy neshoduje se jménem tabulky v databázi, což je možné v případech, kdy se vývojářský tým rozhodne v projektu využít Spring data JPA bude řešením uvést název tabulky pomocí `@Table` na úrovni třídy, pokud je použito DB schéma, pak je nutné jej uvést v parametru schema.

```
@Entity
@Table(name = "VEHICLE", schema = "FACTORY")
public class Vehicle {
```

Zdrojový kód 2 – Implementace Spring Data JPA (zdroj: vlastní)

1.3.2. Anotace sloupce

Pro propojení atributů třídy s atributy tabulky z databáze lze použít `@Column`, která může mít mnoho parametrů, jako je název pole a parametry validace.

```
@Column(name = "CARBODY", nullable = false, length = 30)
private String carbody;
```

Zdrojový kód 3 – Použití notací Column pro parametry (zdroj: vlastní)

1.4. Repositáře a rozhraní

Pro určení rozhraní repozitáře je třeba vytvořit rozhraní pro konkrétní třídu. Rozhraní, které bude komunikovat s databází, by mělo zdědit JpaRepository, tak aby byl umožněn přenos do třídy a jeho identifikátoru jako parametry.

Přiklad vytvoření repozitáře:

```
public interface VehicleRopository extends JpaRepository<Vehicle, Long>
```

Zdrojový kód 4 – Ukázka konfigurace JpaRepository (zdroj: vlastní)

Jedním z klíčových vzorců v současném vývoji je princip rozdělení odpovědnosti, kde každá komponenta musí být zodpovědná za určitou operaci. Použití Spring Data JPA ve Spring Boot odpovídá tomuto principu a umožňuje udržovat jasné rozdíly mezi různými typy entit a operacemi přístupu k datům a přináší řadu výhod:

1) Zapouzdření:
    - Rozhraní repozitáře zapouzdřují operace s daty jednotlivých objektů, což snižuje pravděpodobnost nesprávného přístupu k objektu zvenčí.
2) Modularita:
    - Každé rozhraní je orientováno na jeden typ objektu, což zaručuje jasnost a čistotu kódu.
3) Typová bezpečnost:
    - Používání unikátních rozhraní zaručuje přísnou typizaci, což pomáhá odhalovat chyby v procesu kompilace.[4]

1.5. Dotazy

Spring Data JPA představuje pohodlné způsoby inicializace metod úložiště. Standardní repozitáře funkcí CRUD obvykle obsahují požadavky na úložiště dat. Při použití Spring Data se vyhlášení těchto žádostí dělí do 4 fází:

- Deklarování rozhraní, které rozšiřuje úložiště, a předání mu typu požadované entity a jejího identifikátoru.
- Deklarování metod dotazování.

- Vytváření instancí proxy pro tato rozhraní pomocí konfigurace JavaConfig nebo XML:

```java
@EnableJpaRepositories
class Config { … }
```

Obrázek 1 – Vytváření instancí proxy pro JpaRepository (zdroj [35])

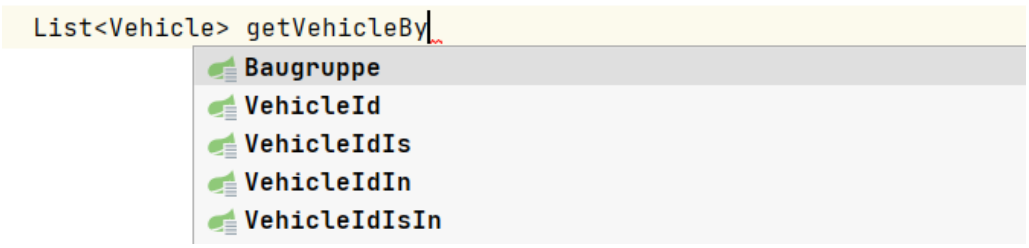
- Inicializace instance úložiště:

```java
class SomeClient {

  private final PersonRepository repository;

  SomeClient(PersonRepository repository) {
    this.repository = repository;
  }

  void doSomething() {
    List<Person> persons = repository.findByLastname("Matthews");
  }
}
```

Obrázek 2 – Inicializace instance úložiště (zdroj [35])

1.5.1. Standardní metody dotazování

Nejjednodušší a nejuniverzálnější způsob vytváření dotazů je takový, kdy jsou metody vytvářeny podle konvence pojmenování, která umožňuje vytvářet dotazy prostřednictvím názvů metod. Kombinováním takovýchto vestavěných prefixů jako jsou `findBy`, `readBy`, `getBy` nebo `queryBy` lze vytvářet smysluplné a dostatečně vypovídající metody dotazování. Při použití tohoto principu pojmenování JPA poskytne tipy, pomocí kterých lze vytvořit dotaz do databáze.



Obrázek 3 – Vytvoření metody pomocí JPA rozhraní (zdroj vlastní)

1.5.2. Vlastní metody dotazování

Navzdory vhodnému mechanismu vytváření šablonových metod pomocí rozhraní existují případy, kdy je potřeba vytvořit složitější dotazy. Spring Data JPA umožňuje i testovat

scénáře, kdy programátor může zapsat téměř jakýkoliv dotaz do databáze pomocí notace `@Query` a JPQL přímo v rozhraní úložiště. Tento přístup poskytuje větší flexibilitu při vytváření dotazů. V něm lze například použít standardní seskupení GROUP BY, JOIN a dokonce definovat hinty pro použití konkrétních indexů v databázi.

```
//Oracle Hint
1 usage
@Query(value = "SELECT v.carserie as carserie, v.carbody as carbody, v.motor as motor," +
        " v.transmission as transmission, v.werk as werk, v.baugruppe as baugruppe, " +
        "v.knr7 as knr7, v.pin13 as pin13 FROM vehicle v JOIN factory f USING(factory_id) " +
        "WHERE f.factory_location = :factoryLocation AND v.carserie = :carserie " +
        "AND v.carbody = :carbody ORDER BY v.motor DESC /*+ INDEX(v IDX_VEHICLE_SEARCH_PARAMS) */",
        nativeQuery = true)
List<VehicleView> findVehiclesByFactoryLocationAndCarserieAndCarbodyProjected(
        @Param("factoryLocation") String factoryLocation,
        @Param("carserie") String carserie,
        @Param("carbody") String carbody);
```

Obrázek 4 – Vytvoření vlastní metody pomocí JPA rozhraní (zdroj vlastní)

1.6. Ukázky použití

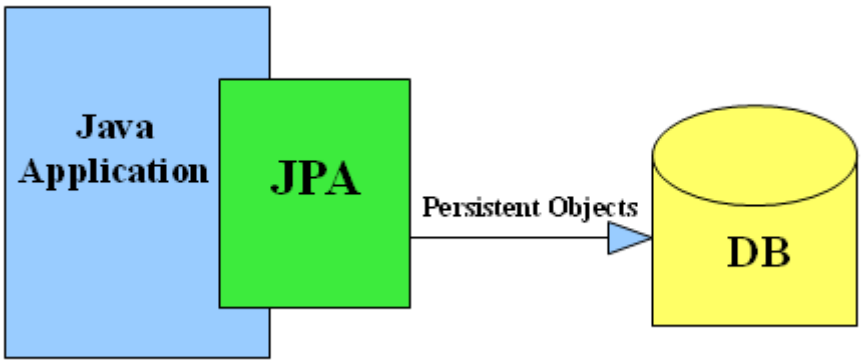
Spring Data JPA je integrován do struktury aplikace pro řízení výroby vozidel této bakalářské práce. Podrobnosti o implementaci Spring Data JPA v této konkrétní aplikaci jsou k dispozici v příloze A, kde je uveden zdrojový kód a konfigurační parametry.

2. Hibernate

Široce používaný open-source framework, který umožňuje zjednodušit komunikační proces s databází. Poskytuje řešení objektově relačního mapování (ORM) a umožňuje vývojářům pracovat přímo s objekty Java bez přímého použití SQL dotazů. Architektura Hibernate je navržena tak, aby bezproblémově spolupracovala s objektově orientovanými programovacími jazyky, jako je Java a relační databáze. [5]

2.1. Persistence dat

V reálném vývoji je nutné zajistit perzistenci dat. To znamená, že aplikační data, se kterými programátor pracuje, musí zůstat zachovaná i po ukončení běhu programu a je nutné, aby stav objektů přesahoval rámec JVM a byl dostupný později.



Obrázek 5 – Vizualizace perzistence (zdroj [40])

Objekt uložený v úložišti bude možné podle potřeby obnovit ve stavu, ve kterém byl uložen. Hibernate poskytuje aplikaci tuto možnost uchování stálosti. [6]

2.2. ORM

ORM je programovací technika používaná k vytvoření abstrakční vrstvy mezi objektově orientovanými programy a relačními databázemi. Tato technika umožňuje mapování dat mezi objekty v programu a databázovými strukturami a zjednodušuje manipulaci s daty v objektově orientovaném programování. V RDBMS jsou data organizována v tabulkách, kde sloupce představují hodnoty, zatímco při programování jsou data uložena ve formě objektů s atributy. Objekty v programu lze vzájemně propojovat uložením odkazu na požadovaný objekt. Například objekt `Vehicle` (vozidlo) může obsahovat odkaz na `Factory` (továrnu),

ve které byl vyroben. Tabulky v RDBMS mají také vzájemné vztahy a umožňují jim ukládat odkaz na související objekt, podle kterého je lze nalézt. V případě vztahu mezi objekty N:M je možné vytvořit asociační tabulku, která bude ukládat odkazy na objekty označující více vztahů mezi nimi. Hibernate umožňuje implementovat tuto logiku v programu pomocí zápisů nad atributy tříd, čímž vytváří rozhraní mezi objekty v programu a entitami v databázi.

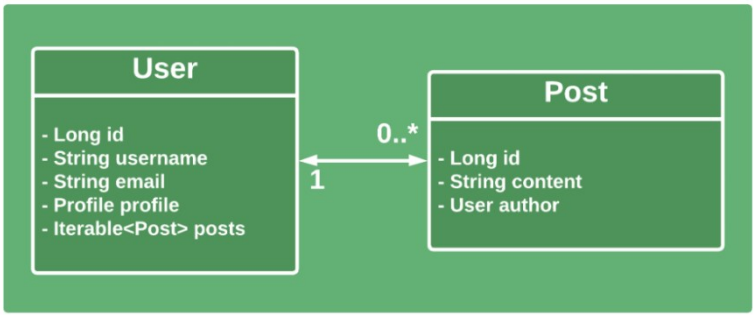
```
@ManyToMany(fetch = FetchType.LAZY)
@JoinTable(
        name = "EMPLOYEEDEPARTMENTS",
        joinColumns = @JoinColumn(name = "EMPLOYEE_ID"),
        inverseJoinColumns = @JoinColumn(name = "DEPARTMENT_ID")
)
private List<Department> departments;
```

Obrázek 6 – Notace N:M (zdroj vlastní)

Tento fragment kódu ze třídy `Emploee` názorně demonstruje použití `@ManyToMany`, pomocí které jsou objekty `Emploee` a `Department` vzájemně propojeny. Asociativní tabulka `EMPLOEEDEPARTMENTS` v databázi ukládá odkazy, se kterými mohou objekty vzájemně interagovat. Zde je důležitá kompletní shoda názvů atribů a tabulek mezi databází. Pokud je například vybrán konkrétní zaměstnanec, tělo odpovědi bude obsahovat také pole objektů s odděleními, ve kterých je členem. [7]

2.3. BatchSize

Při použití Hibernate mohou automaticky vygenerované metody generovat neefektivní dotazy do databáze, zejména když jde o vložené objekty. Často se objevují situace, kdy je nutné získat přístup k objektu, který je spojen vztahem 1:M nebo N:M s jiným objektem. V dané situaci při volání objektu bude docházet k volání svázaných objektů, což způsobí problém N + 1, kde N je počet vložených objektů a 1 je požadovaný objekt.



Obrázek 7 – Vazba objektu (zdroj [9])

Na tomto obrázku je zobrazena souvislost mezi objektem uživatele (User) a publikacemi (Post), které mu patří. Například v případě, když uživateli patří 100 publikací, každé volání metody provede 100 + 1 (N + 1) dotazů do databáze, což může být neefektní v případech růstu objemu dat. Přestože daný problém s velkým počtem databázových dotazů (N+1) lze částečně zmírnit využitím líného načítání, kdy se vložené objekty načítají jen na explicitní požádání a které je navrženo k optimalizaci výkonu tím, že se zamezí zbytečnému načítání dat, existují situace, kdy je skutečně nutné mít přístup ke všem daným objektům současně. V takových případech může být líné načítání spíše překážkou, protože pro každý objekt bude vykonán samostatný dotaz, což vede k velkému počtu databázových dotazů. [9]

```
@OneToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "MANAGER_ID", nullable = false)
@BatchSize(size = 50)
private Employee manager;
```

Obrázek 8 – Použití FetchType.LAZY (zdroj vlastní)

Hibernate umožňuje regulovat počet databázových dotazů při načítání souvisejících objektů pomocí mechanizmu, známého jako batch fetching. K tomu slouží anotace `@BatchSize`, která je obvykle uvedena nad atributem odpovídajícím vnořenému poli objektů, a která určuje, kolik souvisejících objektů může být načteno v jednom dotazu. Tímto způsobem lze optimalizovat počet dotazů na databázi na principu N/M+1, kde N je celkový počet načítaných objektů a M je parametr určený v `@BatchSize`, který specifikuje počet objektů načítaných v jednom dotazu. Tato metoda, i když problém s dotazy N+1 zcela nevyřeší, pomáhá minimalizovat jeho dopady snížením celkového počtu dotazů potřebných k načtení všech objektů. Použití `@BatchSize` umožňuje efektivnější načítání souvisejících objektů tím, že kombinuje více dotazů do menšího počtu hromadných (batch) dotazů, často s využitím SQL klauzule `IN`, kde identifikátory načítaných objektů jsou uvedeny jako parametry. Tímto způsobem lze značně snížit počet vykonaných dotazů k databázi tím, že načte větší množství souvisejících objektů v rámci jednoho dotazu, což efektivně řeší a minimalizuje problém dotazů N+1. [10] Parametr optimální velikosti v `@BatchSize` je definován v rozsahu 10-50, ale závisí na množství dat a vyžaduje zátěžové testování. [11]

2.4. Integrace se Spring Boot

Hibernate je knihovna, která usnadňuje přístup k databázi a umožňuje zpracovávat data efektivněji a jednodušším způsobem, než přímé použití JDBC. V kombinaci se Spring Boot

je to užitečný nástroj pro vytváření výkonných a efektivních Java aplikací, který umožňuje zaměřit se na podnikatelskou logiku, místo řešení nízko úrovňových detailů správy dat a transakcí. Tvůrci balíčků, jako je Gradle, Maven atd., usnadňují konfiguraci a používání Hibernate v monolitických a vícemodulových aplikacích. [6]

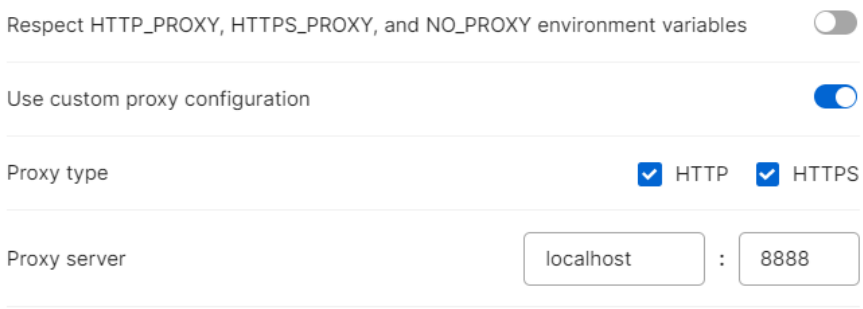
3. Apache JMeter

Apache JMeter je software s otevřeným zdrojovým kódem. Jde o Java aplikaci určenou pro zátěžové testování a měření výkonu uživatelských aplikací. Apache JMeter je určen pro testování výkonu statických zdrojů i dynamických webových aplikací. Umožňuje simulovat velké zatížení serveru, sítě nebo zařízení a analyzovat celkový výkon při různých scénářích zatížení. [12]

3.1. Integrace s Postmanem

Postman je platforma pro tvorbu a testování API. Tento nástroj zjednodušuje každou fázi životního cyklu API a optimalizuje proces vývoje. [13]

Testování v Apache JMeter se provádí voláním endpointů, které lze nastavit ručně nebo lze importovat kolekce endpointů za použití portu, na kterém běží Postman, prostřednictvím změny nastavení uživatelských proxy. V Apache JMeter je proto nutné spustit HTTP(S) Test Script Recorder, čímž se zahájí průběh monitorování portu.



Obrázek 9 – Nastavení proxy v Postman pro komunikaci s Apache JMeter(zdroj vlástní)

Tímto způsobem se tedy automaticky importuje dotaz, který bude volán v Postman, do JMeter, a to se všemi vstupními parametry a lokálními i globálními proměnnými z Postmana, které jsou definovány ve Workspace.

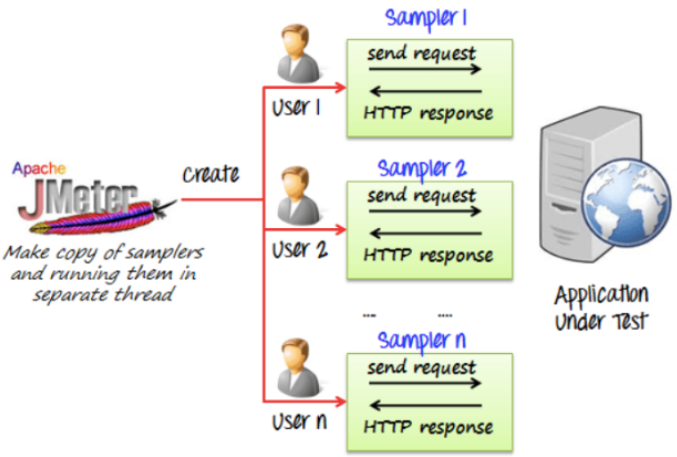
3.2. Nastavení Apache JMeter pro testování Java aplikací

Testování je klíčovou součástí vývoje aplikací. Apache JMeter umožňuje provádět optimalizaci kódu v Java (NodeJS, PHP, ASP.NET) aplikacích a odhalovat slabá místa ve fázích vývoje. JMeter aplikuje zátěž, která přesahuje nosnou způsobilost aplikace, čímž určuje maximální zátěž, kterou server může vydržet. JMeter je desková aplikace napsaná v Java, pro kterou je potřeba JVM verze *6* nebo novější. Před zahájením práce s JMeterem je

třeba se ujistit, že instalovaná verze Java splňuje minimální požadavky pro provoz aplikace. Zatímco správná instalace Java je dostatečná pro základní funkcionalitu JMeter, pro větší monitorování a analýzu výkonnosti Java aplikací může být potřeba nastavení dalších nástrojů, jako jsou Grafana a Prometheus. Je třeba podotknout, že instalace a konfigurace Grafana a Prometheus nejsou povinné pro práci s JMeterem, ale jsou nezbytné pro vizualizaci a detailní analýzu výsledků testů produktivity. [15]

3.3. Scénáře výkonu a zátěžových testů

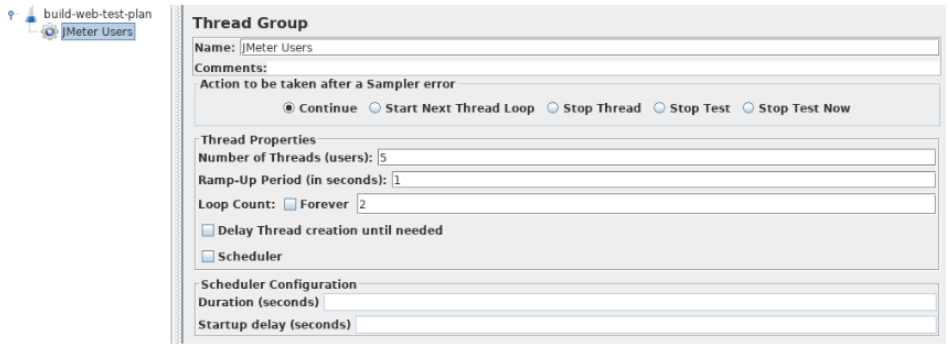
Fáze testování programu pod určitou pracovní zátěží a monitorování výsledků umožňuje určit, zda aplikace funguje očekávaným způsobem při takové zátěži, nebo dělá něco neočekávaného. JMeter provádí testování produktivity pro ověření rychlosti odezvy, spolehlivosti a využití zdrojů a také zaznamenává možné chyby vyplývající z procesu testování. Cílem testování produktivity je detekovat a opravovat veškeré potenciální překážky pro softwarovou aplikaci. [16]



Obrázek 10 – Modelování zatížení serveru (zdroj [14])

Tento obrázek ukazuje, že JMeter podporuje multithreading, což umožňuje více různým skupinám vláken provádět vzorky současně i paralelně.

Pro nastavení JMeter je třeba nejprve přidat skupinu toků, která nastavuje počet uživatelů, kteří zároveň budou posílat dotazy na server, a také frekvenci a množství těchto požadavků.



Obrázek 11 – Nastavení parametrů toků v JMeter (zdroj [17])

Na tomto obrázku je zobrazena konfigurace proudů, ve které 5 uživatelů během jedné sekundy provede jeden požadavek na server. Parametr "Loop Count" definuje, že musí být provedeny dvě opakování daného scénáře testování. Konfiguraci je třeba nastavovat na základě technických požadavků aplikace. Dále je potřeba nakonfigurovat HTTP požadavky, které se konfigurují buď ručně, nebo automaticky přes Postman. Posledním prvkem, který je potřeba do plánu testování přidat, je posluchač (Listener), který má na starosti ukládání výsledků testů všech HTTP požadavků a prezentaci vizuálního datového modelu pomocí posluchačů, jako jsou Graph Results, Results Tree, Response Time Graph atd. Vizualizace výsledků testování, kterou lze vidět prostřednictvím rozhraní JMeter, není konečná, je vhodná pouze pro předběžnou analýzu dat. Pro získání komplexní analýzy dat je nutné spustit testování přes příkazový řádek pomocí příkazu *.\apache-jmeter-5.6.3\bin\jmeter.bat -n -t .\test1.jmx -l logstest.jtl -e -o .\RESULTSTEST*, kde soubor jmeter.bat se používá pro spuštění v operačním systému Windows. Parametr *-n* ukazuje na spuštění JMeteru v režimu bez grafického rozhraní (non-GUI), což umožňuje snížit spotřebu zdrojů a zrychlit proces testování. Parametr *-t* s hodnotou *.\test1.jmx* zadává cestu k souboru testovacího plánu JMeter, který je třeba provést. Parametr *-l* s pokynem logstest.jtl určuje soubor pro zápis výsledků testování. Parametr *-e* umožňuje generování sestav po dokončení testování. Parametr *-o* s hodnotou *\ RESULTSTEST* je definicí cesty pro výstupní sestavu, tedy definuje adresář pro uložení vygenerovaných hlášení. V důsledku provedení příkazu JMeter generuje plnohodnotnou zprávu obsahující statistiky o provedeném testu, jako je čas odpovědi, APDEX statistiky chyb atd.

3.4. Integrace JMeter do procesu vývoje a CI/CD

Automatizované testování hraje důležitou roli v procesu kontinuálního zavádění (CI/CD) a pomáhá identifikovat a řešit problémy v raných fázích vývojového cyklu. Integrace JMeter

s nástroji CI/CD může přinést řadu výhod. Zavedením kontinuální integrace a doručením procesu automatizovaného testování a spuštěním kontrol produktivity mohou být splněna stanovená kritéria účinnosti před nasazením aplikace v produktovém prostředí. Dále může být přínosné i využití cloudové infrastruktury k vytvoření realističtějšího zatížení než v lokálním prostředí a testování aplikace v různých síťových podmínkách. Návod k implementaci integrace lze nalézt na několika internetových stránkách. [18][19]

4. Grafana + Prometheus Monitoring Stack

Grafana v kombinaci s Prometheus představují integrovaný stack pro sběr, monitoring a vizualizaci metrik a jsou klíčovými nástroji pro DevOps specialisty.

Obrázek 12 – Logo Grafana + Prometheus (zdroj [42])

4.1. Grafana

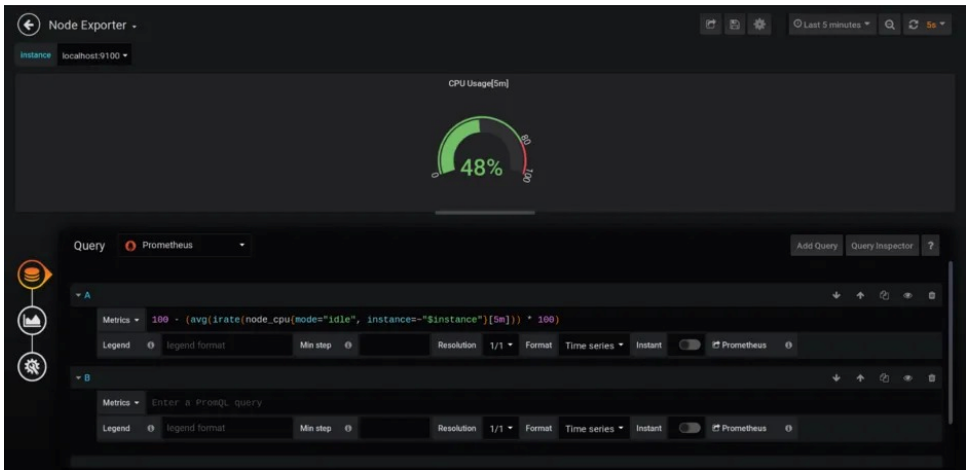
Grafana je software s otevřeným zdrojovým kódem, který umožňuje vizualizovat a zkoumat metriky, logy a výstupy trasování. [21] Umožňuje uživatelům získávat data z různých zdrojů a zobrazovat je v různých customizovaných diagramech stejně jako informovat o problémech a vizualizovat data pomocí grafů.

4.1.1. Možnosti

Grafana je jedním z nejpopulárnějších vizualizačních programů díky své jednoduchosti a možnostem využití, které poskytuje:

- Vizualizace

Disponuje velkou různorodostí možností vizualizace výsledků, která pomáhá snadno prohlížet a analyzovat data. Tyto varianty jsou v "panelech", z nichž se skládá Informační panel (Dashboard) Grafany. Panel je detailní blok vizualizace používaný pro zobrazování dat požadovaných z určitého zdroje dat spojeného s tímto panelem. Tyto informace mohou představovat graf (čidlo, histogram atd.) nebo logy a upozornění. Například lze vytvořit panel snímačů se zdrojem dat z Prometheus a poté si vyžádat údaje o využití CPU uložených v Prometheus pro zobrazení na tomto panelu. V takovém případě bude panel vypadat následovně:



Obrázek 13 – Panel v Grafana(zdroj[22])

Je možné seskupovat panely tak, aby vznikl dashboard, v němž každý panel zobrazuje svou část informací nejvhodnějším způsobem, přesouváním je po pracovní oblasti Grafany. Přestože je Grafana dodávána se zabudovanými a šablonovými panely přičemž lze vytvářet a přidávat své vlastní panely pomocí pluginů.

- Upozornění

Je důležité být informován o všech možných nepředpokládaných či neočekávaných situacích či chybách v práci aplikací, což je nezbytným předpokladem pro zajištění stability systémů a minimalizaci času prostojů aplikací. Grafana umožňuje uživatele informovat o vadách pomocí různých kanálů, jako jsou e-maily, Slack, PagerDuty a další, umožňující uživateli vybrat si nejvhodnější variantu. K aktivaci upozornění je třeba nastavit pravidlo upozornění, které bude aktivovat odesílání upozornění při jeho naplnění. Toto pravidlo funguje jako trigger, který zahájí odeslání upozornění přes vybraný kanál, když je nalezena nějaká chyba. [22]

- Anotace

Grafana umožňuje zanechávat poznámky přímo na grafu. Tato funkce umožňuje označit důležité body grafu, což slouží jako připomínka pro dalšího postup do budoucna, poskytnutí vysvětlení pro člena týmu nebo jednoduše označení speciální události v grafu. [22]

4.1.2. Autentizace

Grafana nabízí širokou škálu možností ověřování, včetně LDAP a OAuth, které poskytují možnost řadit uživatele do různých organizací. Ve verzi Grafana Enterprise je k dispozici další funkce, která umožňuje přidružit uživatele i ke konkrétním týmům. Tato funkce může

být užitečná pro společnosti s vlastními autentizačními systémy, protože Grafana se může s těmito systémy integrovat a automaticky distribuovat přístup k příslušným monitorovacím dashboardům na základě týmové příslušnosti uživatele. [21]

4.2. Prometheus

Vizualizace metrik aplikací v Grafaně vyžaduje mechanismus přenosu dat, který může hrát Prometheus. Tento nástroj, vytvořený společností SoundLoad, funguje jako komunikační most, který shromažďuje a ukládá metriky z aplikace s časovým razítkem, kdy byly zaznamenány, což umožňuje efektivní proces monitorování a jsou později přístupné pro vizualizaci v Grafaně. [23]

4.2.1. Funkce

Prometheus poskytuje řadu základních funkcí:

- Použití flexibilního dotazovacího jazyka PromQL.
- Autonomie jednotlivých serverových uzlů a jejich nezávislost na distribuovaném úložišti.
- Sběr časových řad pomocí HTTP.
- Podpora mnoha režimů budování grafiků a informačních panelů.

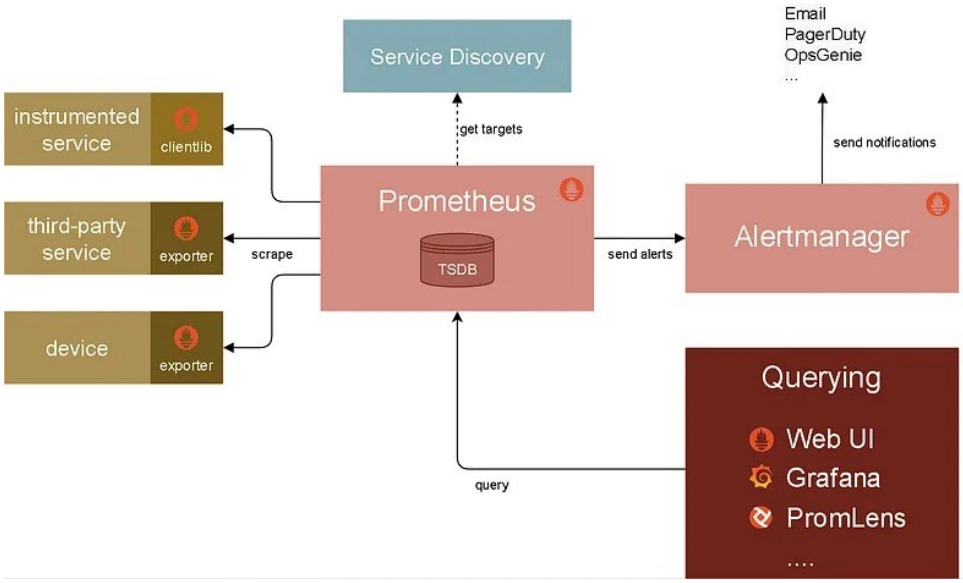
Tyto funkce pomáhají v procesu budování a analýzy metrik. [23]

4.2.2. Metriky

Metrika je záznam změn v průběhu času. Měřící objekty se liší dle aplikace. U webového serveru to může být doba požadavku a odezvy, u databáze počet aktivních požadavků nebo připojení atd. Metriky hrají důležitou roli při porozumění fungování aplikace a analýzy jejích dat. Pokud se například aplikace zpomaluje, je důležité provést detailní analýzu specifických informací pro odhalení problému, kde jsou metriky klíčovým nástrojem. Pomáhají identifikovat příčiny zpomalení a poskytují cenné údaje o výkonu, zatížení systému a době odezvy. Díky metrikám lze také předpovídat potenciální problémová místa dříve, než se projeví dopadem na konkrétního uživatele, a odhalovat veškerá případná slabá místa aplikace. [23]

4.2.3. Architektura

Tento diagram ukazuje strukturu Prometheusa a klíčové prvky jeho ekosystému:



Obrázek 14 – Architektura Prometheus (zdroj [24])

Prometheus je odpovědný za sběr metrik z úkolů, které jsou k tomuto účelu specifikovány. Všechna nasbíraná a zpracovaná data jsou uložena lokálně. Prometheus navíc aplikuje různá pravidla pro tato data pro agregaci a vytváření nových časových řad na jejich základě nebo pro aktivaci systému upozornění, který může být užitečný pro adekvátní reakci vývojáře na různé problémy. Pro vizualizaci a analýzu sesbíraných metrik lze použít Grafana nebo jiné nástroje podporující API Prometheus. [23]

4.2.4. Nastavení Prometheus v Spring aplikaci

Pro nastavení sběru metrik pomocí Prometheus v Java Spring aplikaci je nutné přidat závislost do konfiguračního souboru sestavení projektu. V případě použití balíkových manažerů gradle nebo maven bude konfigurace vypadat takto:

```
implementation 'io.micrometer:micrometer-registry-prometheus:1.12.0'
```

Zdrojový kód 5 – Ukázka konfigurace Prometheus v souboru build.gradle(zdroj vlastní)

Dále, na příkladu použití konfiguračního souboru aplication.yml třeba použít danou konfiguraci:

```
management:
  endpoints:
```

```
  web:
    exposure:
      include:'*'
endpoint:
  health:
    show-details:always
metrics:
  export:
    prometheus:
      enabled: true
```

Zdrojový kód 6 – Ukázka konfigurace Prometheus ve souboru application.yml (zdroj vlastní)

Nyní, při volání konečného bodu http://localhost:8080/actuator/prometheus/, aplikace vrátí metriky ve formátu kompatibilním s aplikací Prometheus. To je vše, co je potřeba k zahájení monitorování metrik dotazů ke konečným bodům a analýze chování aplikace. [25]

5. Oracle Database

Oracle Database představuje systém pro správu relačních databází (RDBMS), který se při zavádění objektově orientovaných funkcí včetně uživatelských typů, následování a polymorfismu mění v objektově-relační (RDBMS). [26]

5.1. Historie

Současná verze databáze Oracle je výsledkem více než 35 let vývoje. Společnost, která je nyní známá jako Oracle Corporation, založili v roce 1977 Larry Ellison, Bob Miner a Ed Oates pod původním názvem Software Development Laboratories. V roce 1979 byl představen Oracle V2, což byl první komerční DBMS založený na SQL. Během následující dekády byly vydány verze 3 až 6, které postupně získávaly nové funkce a možnosti. Databáze se pak dále rozvíjela postupně s každou nově vydanou verzí. Například v roce 1992 Oracle7 zavedla uložené procedury a triggery PL / SQL. Později, v roce 1997, Oracle8 přidal podporu pro nové datové typy, včetně sekvencí pro správu velkých tabulek. Po vydání Oracle8i v roce 1999 se databáze přizpůsobila internetovým výpočtům a poskytovala podporu pro internetové protokoly a integraci s Javou. Oracle Database 10g, která vznikla v roce 2003, položila základy samořídící databáze pomocí Oracle ASM prostřednictvím virtualizace a zjednodušené správy databázového úložiště. V roce 2007 Oracle Database 11g rozšířila možnosti řízení a diagnostiky. Oracle Database 18c a 19c pokračovaly ve vývoji funkcionality s důrazem na zajištění vyššího výkonu a stability. [27]

5.2. Oracle SQL Developer

Integrované vývojové prostředí, které zjednodušuje vývoj a správu databáze Oracle. SQL Developer obsahuje řadu nástrojů pro vývoj a testování databázových dotazů, administrátorskou konzoli pro správu databází, platformu pro migraci databází třetích stran do Oracle a mnoho dalších funkcí. [28]

5.2.1. Použití s databázemi

SQL Developer umožňuje uživatelům vytvářet příkazy SQL a PL/SQL. Tyto příkazy jsou předávány přímo z SQL Developer do databáze Oracle. Pro připojení SQL Developer k databázi je nutné nastavit spojení, použít řetězec připojení, uživatelské jméno, heslo, a také, podle potřeby, nakonfigurovat nastavení síťového protokolu JDBC pro určení způsobu

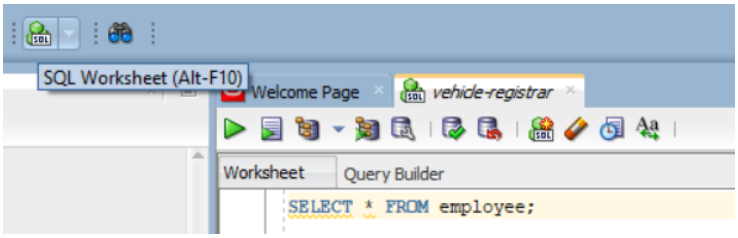
interakce s databází, specifikovat typ spojení, název hostitele nebo IP adresu, port a SID nebo servisní název databáze. [29]

Obrázek 15 – Nastaveni spojení s Oracle Database (zdroj vlastní)

Na tomto obrázku je zobrazen panel pro připojení k databázím. Pro usnadnění práce existuje možnost použití barevného označení panelu nástrojů SQL Developer, což je vhodné při práci s několika databázemi, které mají různé úrovně přístupu. Například pokud stejná aplikace komunikuje se třemi databázemi používanými v různých prostředích – vývojovém, testovacím a produkčním - které mohou mít podobné názvy, pak je barevné označení umožní odlišit i vizuálně, a to pomocí tří různých barev pro zvýraznění odpovídajících pracovních panelů.

5.2.2. SQL Worksheet

Oracle SQL Developer poskytuje SQL tabulka (Worksheet), kterou lze použít k aktualizaci dat prostřednictvím tvorby SQL dotazů. Po vytvoření připojení k databázi je třeba se ujistit, že SQL tabulka je otevřená a připravena k práci. Pokud není pracovní list otevřený, je nutné použít kontextové menu k vytvoření nové tabulky. [30]



Obrázek 16 – Vytvoření pracovního listu (zdroj vlastní)

Po výběru databáze se pro změnu nebo získání dat otevře nový list určený k sestavení SQL dotazů, které budou zaslány přímo k vybrané databáze. Pracovní list také umožňuje

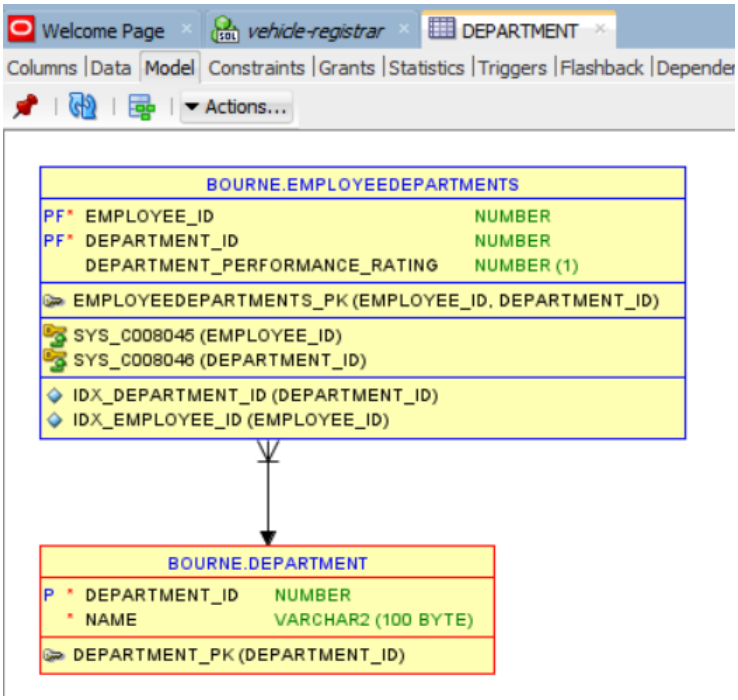
vytvářet procedury, funkce, triggery, indexy, na které se lze obracet jak z kódu, tak přímo z SQL Developer, a které lze využít v dalším procesu vývoje. [30]

5.2.3. Vývojové funkce

Podprogramy jsou bloky PL / SQL, které lze vyvolávat s přenosem parametrů. PL / SQL obsahuje dva typy podprogramů: Procedury a Funkce. Procedura se běžně používá k provádění činnosti a Funkce se používá k výpočtu hodnoty. Podprogramy umožňují rozčlenit program na řízené moduly, které lze znovu použít. [31]

5.2.4. Datový model

SQL Developer také poskytuje možnost prohlížet vazby mezi vytvořenými tabulkami, vizualizovat typy vztahů, primární a cizí klíče a datové typy. Pro hlubší práci s datovým modelováním byl však vyvinut specializovaný nástroj – SQL Data Modeler.



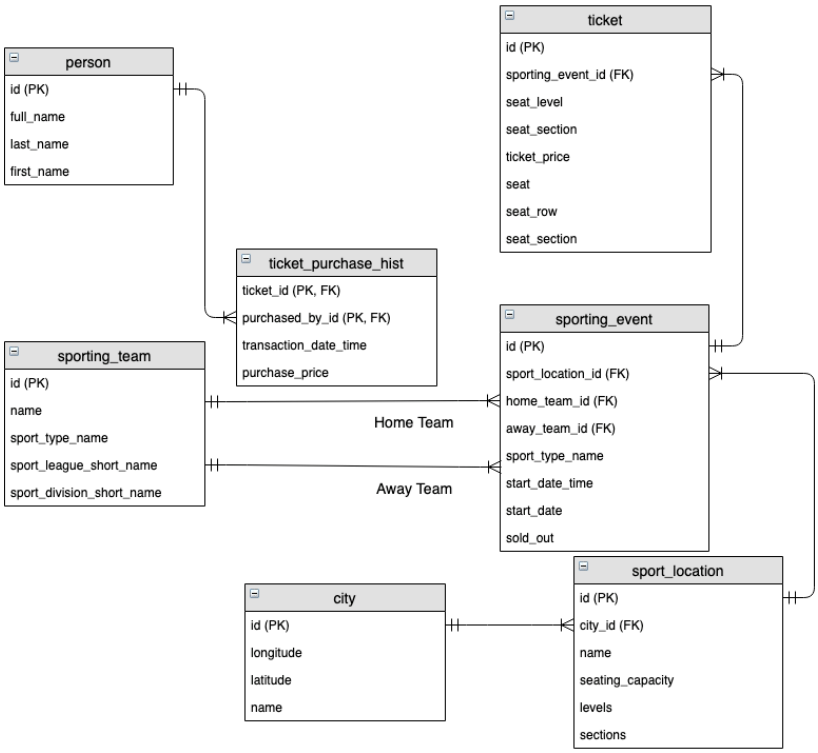
Obrázek 17 – Vizualizace vztahů modelu přes SQL Developer (zdroj vlastní)

5.3. SQL Data Modeler

Grafický nástroj, který zjednodušuje úlohy datového modelování, pomocí kterého může uživatel vytvářet, prohlížet a upravovat logické, fyzické, relační modely. Data Modeler podporuje kolaborativní vývoj prostřednictvím integrovaného řízení zdrojového kódu a lze jej použít v tradičních i cloudových vývojových prostředích. [32]

5.3.1. Vývoj logického modelu

Logický a fyzický model dat slouží jako prostředek k přesnému odrazu funkčních požadavků ve struktuře fyzické databáze. Poskytují různé úrovně technického detailu, které usnadňují vývoj databáze a zároveň plní požadované funkční požadavky. Obchodní analytici a datoví architekti tak využívají logický model dat k vizualizaci operačních či transakčních procesů prostřednictvím diagramů vztahů entit. Tyto modely stanovují, jakým způsobem datové objekty interagují, čímž činí procesy srozumitelnými pro účely podnikání. Vývoj logických modelů pochází abstraktně z fyzické implementace databáze, kde budou zaváděny i nadále.



Obrázek 18 – Logický datový model (zdroj[33])

Tento obrázek ukazuje, že v každé tabulce jsou zastoupeny objekty a jejich atributy vyjádřené v jazyce srozumitelném i pro zadavatele požadavků. Například entita "person" může zahrnovat atributy jako "full_name" a "last_name". Pro identifikaci atributů je v každém řádku tabulky přiřazen PK. Některé objekty mohou mít cizí klíče (FK), což umožňuje poukázat na jejich propojení s jinými objekty ve vztahu typu "1:". Datový návrhář umožňuje ukazovat všechny možné vazby mezi objekty - 1:1, 1:M, N:M. [33]

5.3.2. Relační model

Relační model v SQL Developer Data Modeler působí jako prostředník mezi logickými a fyzickými modely. Podporuje rozhodnutí pro relační návrh nezávisle na omezeních cílové fyzické platformy. Všechny vztahy „N:M“ a všechny hierarchie objektů supertypů/podtypů jsou řešeny během procesu přímého návrhu převádění logického modelu, nebo jeho části, na relační model. Standardizace názvů je rovněž aplikována v průběhu předběžného návrhářského procesu. [34]

5.3.3. Fyzický model

SQL Developer Data Modeler nabízí podporu pro širokou škálu fyzických Oracle objektů a umožňuje uživatelům prohlížet detaily struktury a definice těchto objektů, včetně sekci a podsekcí, přímo ve svém objektovém prohlížeči. Detailní informace jsou dostupné skrze dialogy vlastností, do kterých se dá vstoupit přímo z prohlížeče, bez nutnosti navigace přes další uživatelská rozhraní. Dále SQL Developer Data Modeler zjednodušuje přístup k specifickým konfiguracím, jako jsou nastavení pro oddíly u místních indexů, která jsou přímo dosažitelná z dialogu pro řízení oddílů v tabulkách. [34]

6. Aplikace pro řízení výroby vozidel

6.1. Testovaná problematika

Tato část se zaměřuje na analýzu a optimalizaci výkonu SQL databází a Java aplikace, kterou vytvořil autor a která využívá technologie Hibernate a Spring Data JPA. Cílem je naučit se optimalizovat různé aspekty, od SQL dotazů po ladění a konfiguraci aplikací, za účelem dosažení vyšší efektivity aplikací. Problém optimalizace aplikace lze rozdělit na dvě hlavní části: optimalizaci na úrovni databáze a optimalizaci na úrovni aplikace.

6.1.1. Aplikační úroveň

Optimalizace na úrovni aplikace se zaměřuje na úpravu kódu a ladění konfigurací pro zlepšení výkonu systému. Tato část se zabývá následujícími aspekty:

- Konfigurace Hibernate a Spring JPA

Hlavním zaměřením nastavení platforem JPA Hibernate a Spring Data je optimalizace zpracování vložených objektů, řízení transakcí a zvýšení efektivity plnění žádostí o integraci s databází.

- Vloženost objektů

Optimalizace načítání vnořených objektů (eager a lazy loading) umožňuje snížit počet potřebných dotazů do databáze prostřednictvím správného nastavení momentu načítání dat, tedy nahrávat je pouze podle potřeby nebo bezprostředně po inicializaci objektu. Další výhodou je, že je možné nastavit počet poddotazů pomocí `@BatchSize`, což umožňuje optimalizovat problém N + 1.

- Vytváření projekcí

Správné vytváření projekcí odpovídajících obchodním požadavkům zajišťuje efektivní provádění dotazů a eliminuje použití zbytečných atributů objektů.

- Efektivita dotazů

Monitorování efektivity dotazů zahrnuje optimalizaci dotazů SQL generovaných pomocí Hibernate a Spring Data, stejně jako vlastních customových dotazů či testování „hintů“ pro indexy, a to pro minimalizaci doby jejich plnění a minimalizaci použitých zdrojů, které je

třeba vynaložit pro zpracování dotazů. Toho je dosaženo použitím indexů, ukládáním výsledků do mezipaměti a minimalizací množství dat přenášených mezi databází a aplikací.

6.1.2. Úroveň databáze

Optimalizace na úrovni databáze je zaměřena na zvýšení efektivity zpracování dat a zlepšení výkonu SQL dotazů. Tento proces zahrnuje následující kroky:

- Vytváření indexů

Správné použití indexů pro nalezení určitých záznamů je lepší praxí a funguje rychleji než kompletní prohledávání tabulky. [36] Indexy v databázích často používají binární vyhledávací stromovou strukturu, kde jsou data umístěna v uzlech tak, že pro každý uzel mají všechny uzly v jeho levém podstromu hodnoty menší než hodnota v nadřazeném uzlu a všechny uzly v pravém podstromu mají hodnoty větší než hodnota v nadřazeném uzlu. Tato struktura umožňuje vyhledávat prvky podle indexů v databázi s logaritmickou složitostí LOG2(n), což urychluje operace ve srovnání, například, s lineárním vyhledáváním. [41]

- Složené indexy

Zpracování dotazů je možné urychlit přidáním dalších indexů, ale vzhledem k tomu, že jedno prohledání indexu je rychlejší než dvě, efektivnějším řešením je často optimalizace složených indexů prostřednictvím reverze uspořádání sloupců. Pokud není vyhledávání podle jednoho klíče optimální, umístění nejvíce relevantního pole na první pozici v indexu umožňuje jeho použití, protože první sloupec indexu se vždy používá pro vyhledávání. Je to podobné jako při hledání příjmení v telefonním seznamu bez znalosti křestního jména. Výběr správné sekvence sloupců ve složeném indexu může výrazně zlepšit frekvenci jeho používání a výkon databáze. [36]

- Používání indexových hintů

Jedním ze způsobů, jak zlepšit optimalizaci dotazů, je použít indexových hintů. Přestože se obvykle nekontroluje, jak SQL Server získává požadovaná data, indexový hint nutí

optimalizátor žádostí použít index definovaný hintem. [37]

```
@Query(value = "SELECT v.carserie as carserie, v.carbody as carbody, v.motor as motor," +
        " v.transmission as transmission, v.werk as werk, v.baugruppe as baugruppe, " +
        "v.knr7 as knr7, v.pin13 as pin13 FROM vehicle v JOIN factory f USING(factory_id) " +
        "WHERE f.factory_location = :factoryLocation AND v.carserie = :carserie " +
        "AND v.carbody = :carbody ORDER BY v.motor DESC /*+ INDEX(v IDX_VEHICLE_SEARCH_PARAMS) */",
        nativeQuery = true)
List<VehicleView> findVehiclesByFactoryLocationAndCarserieAndCarbodyProjected(
```

Obrázek 19 – Vytvoření hintu v repozitáři aplikace (zdroj vlastní)

Hinty se sice často používají přímo při vytváření SQL dotazů skrz SQL Developer, ale v případě Java Spring aplikace se zapisují na úrovni repozitáře v kódu.

6.2. Testovací datasety

6.2.1. Základní popis

Pro generování dat v této práci autor použil data generovaná podle relačního datového modelu této aplikace pomocí služeb Mockaroo a ChatGPT. [38][39] Hlavním cílem při generování dat bylo důsledné dodržení datových typů a počtu záznamů v hotových souborech, aby po exportu do databáze bylo možné provádět testování podle stanovených scénářů.

6.2.2. Způsob použití

Proces využívání dat v aplikaci zahrnuje:

1) Načítání dat
    - Data vygenerovaná v souboru se nahrávala do databáze pomocí použití příkazu *@path_to _file.sql* přes sqlplus – terminál příkazového řádku pro správu Oracle Database.
2) Použití dotazů
    - Na data byly spuštěny SQL dotazy pro výběr entit po aplikaci různých optimalizačních technik v kódu aplikace a databázi.
3) Monitoring produktivity
    - Během provádění dotazu pomocí Grafana a Apache JMeter byly sledovány klíčové ukazatele výkonu, jako je využití CPU, doba odezvy pro různé intenzity dotazů atd.
4) Analýza výsledků

- Výsledky byly analyzovány tak, aby určovaly účinnost různých metod optimalizace dotazů při různých objemech dat.

5) Optimalizace a testování změn
    - Na základě získaných dat se prováděla optimalizace SQL databáze a kódu aplikace. Po provedení změn podle scénářů byly znovu otestovány klíčové ukazatele výkonnosti.

6.2.3. Testování optimalizačních technik

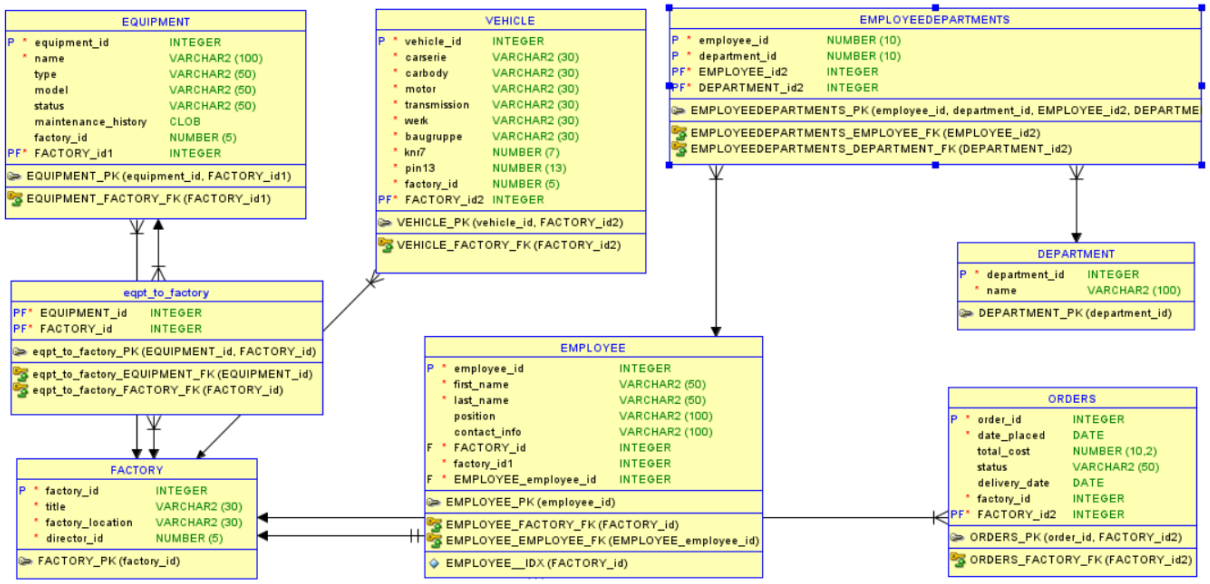
Pro testování různých metodik optimalizace aplikace a SQL dotazů autor uplatnil Apache JMeter jako základní nástroj pro měření produktivity. Testovány byly pouze GET metody - bezparametrické i parametrické. Byly vytvořeny endpointy pro dotazování všech entit aplikace, které byly poté otestovány v aplikaci Postman. Pomocí proxy konfigurace byla nastavena čtečka dotazů, včetně všech parametrů z Postmana do Apache JMeter. To znamená, že jakýkoli požadavek provedený v Postman byl automaticky přečten do Apache JMeter. Pro monitorování, sledování a vizualizaci metrik se uvnitř aplikace používaly Grafana a Prometheus.

6.3. Databázová aplikace

K vytvoření testovací aplikace autor použil Java Spring Framework ve spojení s Oracle SQL databází, jejíž interakce byla prováděna přes Hibernate.

6.3.1. Návrh databáze

Databáze byla navržena tak, aby umožňovala testování různých technik optimalizace dotazů SQL a samotnou aplikaci, včetně všech typů vztahů mezi entitami. Modeluje procesy výroby automobilů v továrnách, účtování vybavení komponent a kusovníků, fakturační organizace a organizaci práce zaměstnanců v různých odděleních.



Obrázek 20 – Diagram databáze, relační model (zdroj vlastní)

6.3.2. Normalizace dat

Data byla normalizována podle třetí normální formy, aby byla zajištěna maximální efektivita při práci s daty v aplikaci. Tato normalizace pomáhá minimalizovat redundanci a zajišťuje, že vztahy mezi tabulkami modelovanými přes SQL ModelMapper jsou pochopeny a správně implementovány jak na úrovni databáze produktu, tak na úrovni aplikace. Tím je zajištěn rychlý a přesný přístup k entitám. Databázové struktury poskytují různé způsoby vyhledávání dat. Aplikace mohou data číst sekvenčně nebo používat speciální metody pro přímý přístup k požadovaným datům, což snižuje počet vstupů a výstupů a urychluje jejich vyhledávání. [41] Redukce na třetí normální formu tedy podporuje lepší využití těchto databázových funkcí, efektivnější vyhledávání a zlepšení celkového výkonu systému.

6.3.3. Návrh aplikace

Vývoj testovací aplikace autor rozdělil do tří fází. V první fázi byl navržen design odpovídající designu databáze. V další fázi byla vytvořena jednoduchá multimodulární aplikace v Java Spring, ve které autor použil Gradle jako systém automatického sestavení a řízení závislostí. V poslední fázi byla konfigurována komunikace mezi jednotlivými moduly aplikace a stejně tak je označena jejich hierarchie, pro jasnější rozdělení odpovědnosti za zpracování dat. Aplikace neobsahuje klientskou část a může být otestována prostřednictvím takových HTTP klientů, jako je Postman, Insomnia atd.

6.4. Testovací scénáře

Za účelem testování účinnosti různých technik optimalizace databáze byly vytvořeny speciální scénáře. V rámci těchto scénářů autor použil všechny klíčové tabulky s různým počtem dat: 1000, 10000 a 100000 záznamů pro každou tabulku, což umožnilo vyhodnotit účinnost různých optimalizačních metod pro různé objemy dat. Po provedení všech variant optimalizace nad tabulkami v každém scénáři byly výsledky testů analyzovány s cílem určit nejefektivnější metody optimalizace pro každý případ. Pro testování byly použity nástroje Apache JMeter a Grafana v kombinaci s Prometheus, přičemž každý z nich plnil specifické funkce:

1) Grafana
    - Sledování využití prostředků CPU během provádění dotazů. Příslušná metrika – `process_cpu_usage`.
    - Sledování zatížení disku – analýza objemu operací čtení a zápisu na disk při zpracování požadavků. Příslušná metrika – `go_memstats_alloc_bytes`.

2) Apache JMeter
    - Určení průměrné doby odezvy – měření doby, za kterou server zpracuje požadavek.
    - Počítání a porovnávání počtu úspěšně zpracovaných požadavků oproti neúspěšným požadavkům.
    - Specifikace chyb v neúspěšných požadavcích – popis a klasifikace chyb, které se vyskytly při zpracování požadavku.

Podrobné výsledky testů optimalizačních technik Apache JMeter a Grafana jsou k dispozici v příloze. B. Hlavní výsledky (CPU, average response time) pak byly převedeny do grafů pro vizuální srovnání. Pro každý scénář bylo použito 5 technik zpracování dat, přičemž se předpokládala různá úroveň optimalizace kódu a databáze:

- Žádná optimalizace

V tomto scénáři nebyly použity žádné speciální techniky ke zlepšení výkonu. Jedná se o základní scénář, který byl použit pro srovnání s jinými optimalizovanými přístupy.

- Indexy a `@BatchSize`

Tato technika zahrnovala použití jednoduchých indexů pro urychlení databázových dotazů a optimalizaci velikosti dávky pro efektivnější zpracování velkého množství dat snížením počtu I/O operací.

- Složené indexy a projekce

Použití složených indexů urychlilo provádění dotazů. Projekce umožňují načíst z databáze pouze ty atributy, které jsou potřebné pro konkrétní dotaz, čímž se sníží množství přenášených dat. Tento přístup byl zaveden, aby se zdůraznil význam přizpůsobení optimalizačních strategií konkrétním podmínkám a požadavkům, zejména při práci s velkým množstvím dat. Projekce v Java Spring lze implementovat jak pomocí běžného použití DTO, tak pomocí rozhraní.

```
public interface FactoryView {

    no usages
    String getTitle();
    no usages
    String getFactoryLocation();
//    Set<EquipmentView> getEquipment();
    no usages
    Set<OrderView> getOrders();
    no usages
    Set<VehicleView> getVehicles();
}
```

Obrázek 21 – Ukázka implementace projekce prostřednictvím rozhraní (zdroj vlastní)

Použití projekcí může výrazně snížit množství přenášených a zpracovávaných dat, čímž se sníží doba odezvy a zatížení procesoru. Pokud je například potřeba název, umístění závodu, informace zakázkách a vozidlech, a není třeba získávat informace o vybavení závodu, které se v programu vrací jako pole souvisejících objektů - tento atribut lze z projekce vyloučit. Tím se sníží množství dat získávaných z databáze a sníží se zatížení systému. Tento přístup je flexibilnější a přispívá k lepší optimalizaci programu. Vyplatí se však mít jasno ve funkčních požadavcích a používat projekce pouze tam, kde je to relevantní.

```
public interface FactoryRepository extends JpaRepository<Factory, Long> {

    1 usage
    List<FactoryView> findAllFactoriesBy();

}
```

Obrázek 22 – Ukázka implementace projekce prostřednictvím rozhraní (zdroj vlastní)

- Složené indexy, hinty v SQL a Spring, projekce

V této metodě byla použita kombinace složených indexů s hinty v SQL, které vedou optimalizátor dotazu k efektivnějšímu plánu provádění.

- Složené indexy, hinty v SQL a Spring, projekce a `@BatchSize`

Nejkomplexnější přístup, který kombinuje použití složených indexů, hintů SQL, projekce a optimalizaci velikosti dávky (`@BatchSize`). Cílem této kombinace bylo maximalizovat rychlost a efektivitu zpracování dotazů a ověřit, kdy je tato kombinace skutečně užitečná.

Všechny testovací případy simulovaly zátěž 150 uživatelů po dobu 5 sekund, což odpovídá přibližně 30 uživatelům za sekundu. Aby byla zajištěna spolehlivost výsledků a eliminován dopad předchozích operací na výkon, byl v Oracle Database použit příkaz *ALTER SYSTEM FLUSH SHARED_POOL*, který před každým novým testovacím scénářem vyčistil *shared pool*.

7. Výsledky z použitých nástrojů

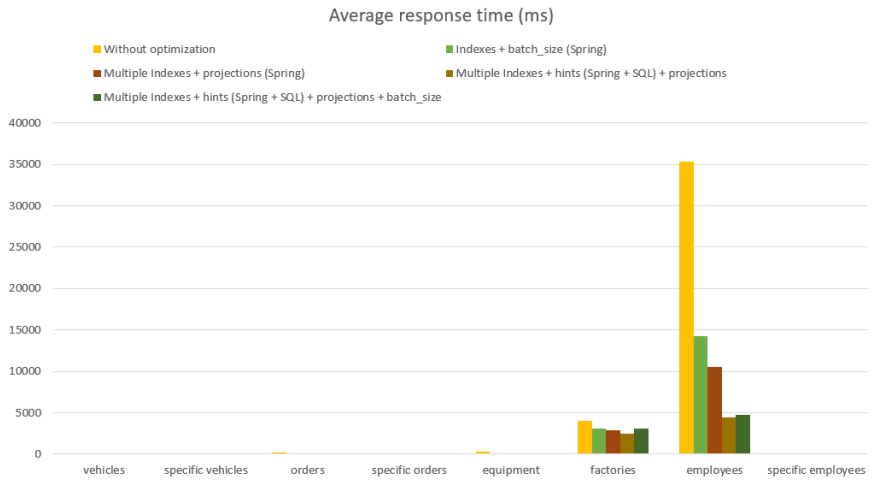
Výsledky testů ukázaly rozdíl ve výkonu mezi různými metodami optimalizace a různými objemy dat. Tato tendence byla patrná zejména u tabulek, jako je `equipment`, `factories` a `employees`, kde kombinované metody optimalizace snížily dobu odezvy na úkor zvýšeného zatížení procesoru.

7.1. Analýza výsledků

Porovnání různých velikostí dat:

- 1000 záznamů

Při testování dotazů na tabulky obsahující 1000 záznamů se téměř všechny optimalizace projevily rychlou odezvou. Jako lepší řešení se však ukázalo použití složených indexů, SQL hintů pro indexy a projekce. To ukazuje, že při práci s malými objemy dat a relativně malými souvisejícími objekty neřeší `@BatchSize` vždy problém `N+1` – může být sice účinný, ale jeho použití není vždy opodstatněné.

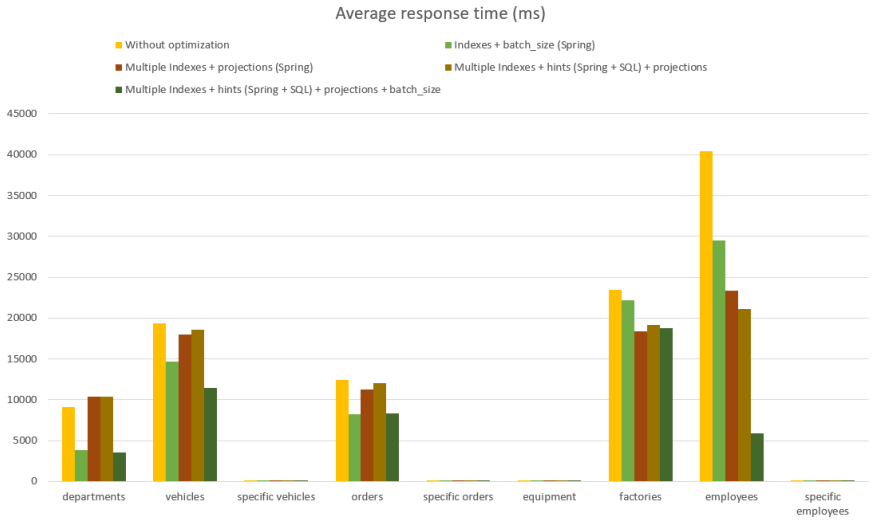


Obrázek 23 – Doba odezvy serveru (ms) při porovnání různých optimalizačních technik (zdroj vlastní)

Přiložený graf ukazuje výrazné zlepšení průměrné doby odezvy při použití všech optimalizačních technik, zejména v kategoriích `factories` a `employees`

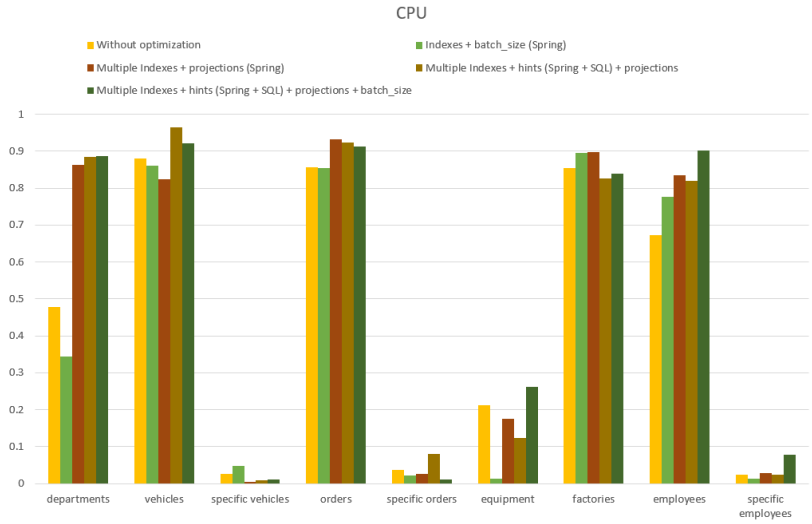
- 10000 záznamů

S rostoucí velikostí dat byl pozorován nárůst doby odezvy, zejména bez optimalizace. Optimalizace pomocí indexů a projekcí pomohla snížit dobu odezvy, ale zvýšila spotřebu procesoru.



Obrázek 24 – Doba odezvy serveru (ms) při porovnání různých optimalizačních technik (zdroj vlastní)

V této situaci bylo nejlepší možností optimalizace použití kombinovaných metod ve spojení s `@BatchSize`, který určuje, na kolik částí má být poddotaz rozdělen, což přispívá k řešení problému `N+1`. Tato metoda však zvyšuje zatížení procesoru a při spuštění programu vyžaduje výkonnější výpočetní prostředky.



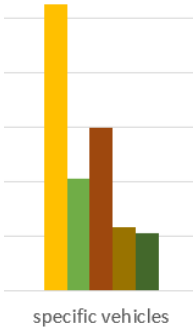
Obrázek 25 – Vytížení procesoru při porovnání různých optimalizačních technik (zdroj vlastní)

Na základě testování se potvrdilo, že rychlost zpracování dotazu výrazně závisí na počtu objektů spojených s entitou, proto je důležité důkladně analyzovat obchodní požadavky, aby bylo možné přesně určit parametr *batch_size*. Pokud je například pro očekávanou velikost vnoření 1000 entit z hlediska rychlosti optimální velikost *batch_size* 25, pak pro velikost vnoření 10000 entit by byla efektivnější velikost 50. Pro dosažení maximálního výkonu je důležité přizpůsobit konfigurační parametry v závislosti na velikosti dat a struktuře dotazu.

- 100000 záznamů

Největší množství dat vykazovalo při absenci optimalizace výrazné zpoždění v době odezvy, což vedlo k přetížení paměti, chybám a pádům programu. Komplexní optimalizační techniky pomohly zlepšit výkon, omezit fatální chyby a udržet program v chodu.

Výsledkem testování je, že pátá metoda, kombinující využití složených indexů, hintů, projekcí a `@BatchSize`, prokázala zlepšení výkonu v parametrických dotazech.



Obrázek 26 – Doba odezvy serveru (ms) při porovnání různých optimalizačních technik (zdroj vlastní)

Pro testování výběru konkrétního vozidla z databáze na základě zadaných parametrů byl vytvořen kompozitní index pro pole `factoryLocation`, `carserie` a `carbody`. Metoda `specific_vehicles` je parametrický dotaz, což znamená, že odpovídá operaci vyhledávání dat podle specifikovaných parametrů. Nejlepší výsledek výkonu, který je zvýrazněn zeleně, je vidět v pátém sloupci výše uvedeného obrázku. Tento výsledek potvrzuje, že správně složené kompozitní indexy zajišťují efektivnější a rychlejší zpracování dat.

V případě dotazů na objekty s velkým počtem souvisejících interních objektů, jako jsou `vehicles`, `orders` a `employees`, se osvědčily optimalizační techniky jako "Multiple indexes + hints + projections + batch_size" a "Indexes + batch_size". Tyto přístupy výrazně zlepšily výkonnost dotazů díky efektivnímu využití indexů a správnému vyladění velikosti `@BatchSize`. Na druhé straně použití projekce bez optimalizace velikosti dávky poskytlo horší výsledky. Tento jev lze vysvětlit tím, že ačkoli projekce mohou být užitečné, jejich účinnost silně závisí na správném vyladění dalších parametrů dotazu, jako je například velikost dávky. Kompletní výsledky testů pro všechny scénáře, prezentované jako metrické hodnoty a grafy, jsou k dispozici v příloze C.

Závěr

Cílem této bakalářské práce bylo analyzovat a optimalizovat výkon OracleDB v Java aplikaci pomocí frameworků Hibernate a Spring Data JPA. Práce byla zaměřena na optimalizaci datové vrstvy a komunikačních procesů mezi aplikací a databází. Byly studovány různé techniky optimalizace SQL databází a způsoby, jak tyto techniky efektivně implementovat pomocí Hibernate a Spring Data JPA. Autor také identifikoval běžné výkonnostní problémy databázových aplikací a představil možná řešení pro jejich odstranění.

Klíčovými optimalizačními technikami bylo použití SQL indexů, návrhových technik a ladění velikosti batch_size, které byly testovány na různých typech dotazů s různou velikostí batch_size. Klíčovým výsledkem bylo potvrzení, že kombinace použití indexů a projekcí entit spolu s optimalizovanou velikostí paketu přináší výrazné zlepšení výkonu. V případech, kdy byly použity pouze projekce bez optimalizace velikosti batch_size, byl výkon poměrně nízký, Tento jev ukazuje, že správné a promyšlené nastavení všech aspektů SQL dotazů může výrazně zlepšit výkon programu.

Práce nabízí programátorům a databázovým profesionálům používajícím Hibernate a Spring Data JPA cenné poznatky o tom, jak tyto technologie efektivně využít ke zlepšení výkonu databází SQL. Nástroje a techniky uvedené v této práce mohou pomoci zjednodušit vývoj a optimalizovat výkon databázových aplikací.

Při zpracování této práce si autor prohloubil své znalosti v oblasti pochopení fungování Hibernate, nástrojů pro testování a monitorování metrik Apache JMeter a Grafana a indexů SQL. Práce tak byla cenným příspěvkem k odbornému rozvoji.

Další možností pro rozvoj této práce je studium interakce frameworku Spring s procedurami a funkcemi SQL a také optimalizace a komplexní analýza HTTP metod POST a PUT. Moderní IT firmy využívají celou řadu databází, z nichž každá může mít své vlastní konfigurační zvláštnosti a specifika interakce se Spring Framework, což je předpokladem pro pokračování výzkumu v oblasti analýzy interakce různých databází a Spring Framework.

Použitá literatura

[1] Spring Data JPA. *Broadcom* [online]. © 2005-2024 [cit. 2024-01-07]. Dostupné z https://spring.io/projects/spring-data-jpa [online].

[2] Spring. Configuration. *Broadcom* [online]. © 2005-2024 [cit. 2024-01-07]. Dostupné z https://docs.spring.io/spring-data/jpa/reference/repositories/create-instances.html

[3] JPA, Hibernate And Spring Data JPA. Medium [online]. Březen 2023 [cit. 2024-01-07]. Dostupné z https://medium.com/@burakkocakeu/jpa-hibernate-and-spring-data-jpa-efa71feb82ac

[4] Best Practices: Creating Repository Interfaces with JPA. *Medium* [online]. Srpen 2023. [cit. 2024-01-15]. Dostupné z https://medium.com/@bubu.tripathy/best-practices-creating-repository-interfaces-with-jpa-d904bee64397

[5] Hibernate Architecture. *Medium* [online]. Leden 2024 [cit. 2024-01-15]. Dostupné z https://medium.com/@himani.prasad016/hibernate-architecture-5f78e7e067fe

[6] Hibernate ORM. Your relational data. Objectively. *Hibernate* [online]. © 2024 [cit. 2024-01-28]. Dostupné z https://hibernate.org/orm/

[7] Java persistence – JPA, Hibernate, ORM. *Skillmea* [online]. Únor 2019 [cit. 2024-01-30]. Dostupné z https://skillmea.cz/blog/java-persistence-jpa-hibernate-orm

[8] What is the Relationship Between Hibernate and Spring? *Medium* [online]. Leden 2023 [cit. 2024-02-02]. Dostupné z https://medium.com/@busrabzgz/what-is-the-relationship-between-hibernate-and-spring-ee060f2eeb27#:~:text=In%20summary%2C%20Spring%20Boot%20and,and%20persist%20data%20in%20an

[9] N+1 Problem in Hibernate and Spring Data JPA. *Baeldung* [online]. [cit. 2024-02-02]. Dostupné z https://www.baeldung.com/spring-hibernate-n1-problem

[10] 3 Ways to Deal With Hibernate N+1 Problem. HackerNoon [online]. Leden 2022 [cit. 2024-02-03]. Dostupné z https://hackernoon.com/3-ways-to-deal-with-hibernate-n1-problem

[11] Chapter 13. Batch processing. *Red Hat Middleware* [online]. © 2004 [cit. 2024-02-04]. Dostupné z https://docs.jboss.org/hibernate/core/3.3/reference/en/html/batch.html

[12] Apache JMeter. *Apache Software Foundation* [online]. © 1999-2024 [cit. 2024-02-04]. Dostupné z https://jmeter.apache.org/

[13] What is Postman? *Postman, Inc.* [online]. © 2024 [cit. 2024-02-11]. Dostupné z https://www.postman.com/product/what-is-postman/

[14] How to Use JMeter to Test Your Web Application. *Clutch.co* [online]. Únor 2023 [cit. 2024-02-15]. Dostupné z https://clutch.co/resources/how-to-use-jmeter-to-test-your-web-application

[15] Performance Test of Web Application using Apache JMeter. *Webkul Software Pvt Ltd* [online]. © 2010-2024 [cit. 2024-02-19]. Dostupné z https://www.uvdesk.com/en/blog/performance-test-web-application-using-apache-jmeter/

[16] Ultimate Guide to JMeter Performance Testing. *Simplilearn Solutions* [online]. © 2009-2024 [cit. 2024-02-20]. Dostupné z https://www.simplilearn.com/tutorials/jmeter-tutorial/jmeter-performance-testing

[17] Building a Web Test Plan. *Apache Software Foundation* [online]. © 1999-2024 [cit. 2024-02-21]. Dostupné z https://jmeter.apache.org/usermanual/build-web-test-plan.html

[18] Integrating JMeter with AWS CodePipeline for Automated Performance Testing. *Medium* [online]. Březen 2023 [cit. 2024-03-11]. Dostupné z https://medium.com/@hypersense/integrating-jmeter-with-aws-codepipeline-for-automated-performance-testing-ddd237479cce

[19] Best practices for JMeter performance testing cloud integration. *Linkedin* [online]. © 2024 [cit. 2024-03-11]. Dostupné z https://www.linkedin.com/advice/1/how-do-you-integrate-jmeter-performance-testing#:~:text=You%20should%20use%20a%20version,from%20your%20CI%2FCD%20tools

[20] Prometheus-Grafana. *Medium* [online]. Červenec 2020 [cit. 2024-03-15]. Dostupné z https://medium.com/@palakj095/prometheus-grafana-a5dccea1f2f9

[21] About Grafana. *Grafana Labs* [online]. © 2024 [cit. 2024-03-15]. Dostupné z https://grafana.com/docs/grafana/latest/introduction/

[22] What is Grafana? *Medium* [online]. Srpen 2023 [cit. 2024-03-21]. Dostupné z https://medium.com/@MetricFire/what-is-grafana-8de44d241765

[23] What is Prometheus? *Prometheus Authors* [online]. © 2014-2024 [cit. 2024-04-05]. Dostupné z https://prometheus.io/docs/introduction/overview/

[24] Unveiling the Architectural Brilliance of Prometheus. *Medium* [online]. Červen 2023 [cit. 2024-04-14]. Dostupné z https://medium.com/@extio/unveiling-the-architectural-brilliance-of-prometheus-af07cca14896

[25] Monitoring Made Simple: Empowering Spring Boot Applications with Prometheus and Grafana. *Medium* [online]. Červenec 2023 [cit. 2024-04-20]. Dostupné z https://medium.com/simform-engineering/revolutionize-monitoring-empowering-spring-boot-applications-with-prometheus-and-grafana-e99c5c7248cf

[26] Relational Database Management System (RDBMS). *Oracle* [online]. © 2024 [cit. 2024-04-21]. Dostupné z https://docs.oracle.com/en/database/oracle/oracle-database/12.2/cncpt/introduction-to-oracle-database.html

[27] Brief History of Oracle Database. *Oracle* [online]. © 2024 [cit. 2024-04-22]. Dostupné z https://docs.oracle.com/en/database/oracle/oracle-database/19/cncpt/introduction-to-oracle-database.html

[28] SQL Developer application interfaces. *Oracle* [online]. © 2024 [cit. 2024-04-25]. Dostupné z https://www.oracle.com/database/sqldeveloper/#:~:text=Oracle%20SQL%20Developer-,Oracle%20SQL%20Developer%20is%20a%20free%2C%20integrated%20development%20environment%20that,Linux%20with%20%2B%205%20Million%20users.

[29] Accessing Oracle Database with Oracle SQL Developer. *Oracle* [online]. © 2024 [cit. 2024-04-25]. Dostupné z https://docs.oracle.com/en/database/oracle/oracle-database/19/ntdbi/accessing-oracle-database-with-oracle-sql-developer.html

[30] Write a Select Statement. *Oracle* [online]. © 2024 [cit. 2024-04-25]. Dostupné z https://www.oracle.com/tools/technologies/howto-sql-worksheet-basic-syntax.html

[31] Using PL/SQL Subprograms. *Oracle* [online]. © 1996-2003 [cit. 2024-04-29]. Dostupné z https://docs.oracle.com/cd/B13789_01/appdev.101/b10807/08_subs.htm

[32] Data Modeling with Oracle SQL Developer. *Oracle* [online]. © 2024 [cit. 2024-05-01]. Dostupné z https://www.oracle.com/cz/database/sqldeveloper/technologies/sql-data-modeler/#:~:text=Oracle%20SQL%20Developer%20Data%20Modeler%20is%20a%20free%20graphical%20tool,dimensional%2C%20and%20data%20type%20models.

[33] What's the Difference Between a Logical Data Model and a Physical Data Model. *Amazon* [online]. © 2024 [cit. 2024-05-02]. Dostupné z https://aws.amazon.com/compare/the-difference-between-logical-and-physical-data-model/

[34] An Introduction to Oracle SQL Developer Data Modeler. Relational Models. *Oracle* [online]. Červen 2009 [cit. 2024-05-05]. Dostupné z https://www.oracle.com/technetwork/developer-tools/datamodeler/sqldeveloperdatamodeleroverview-167687.html#:~:text=The%20SQL%20Developer%20Data%20Modeler,target%20physical%20platform(s).

[35] Query methods. *Broadcom* [online]. © 2005-2024 [cit. 2024-05-03]. Dostupné z https://docs.spring.io/spring-data/commons/reference/repositories/query-methods.html

[36] WINAND, Markus. *SQL Performance Explained: Everything Developers Need to Know about SQL Performance*. vyd. Wien: M.Winand, 2012. ISBN 978-3950307825

[37] Index Hints Give You Control. *Bertwanger* [online]. Červenec 2018 [cit. 2024-05-04]. Dostupné z https://bertwagner.com/posts/should-you-use-index-hints/

[38] *Mockaroo* [online]. © 2024 [cit. 2024-05-05]. Dostupné z https://www.mockaroo.com/

[39] *Chat GPT* [online]. © 2024 [cit. 2024-05-05]. Dostupné z https://chat.openai.com/

[40] Hibernate Persistence Context and Object's LifeCycle. *Aishwaryavaishno* [online]. Červen 2013 [cit. 2024-05-11]. Dostupné z https://aishwaryavaishno.wordpress.com

[41] CELKO, Joe. *Joe Celko's Trees and Hierarchies in SQL for Smarties.* 2. vyd, 2012. Morgan Kaufmann. ISBN 978-0123877338.

Přílohy

Příloha A – Archivované soubory aplikací pro řízení výroby vozidel

Název přílohy: vehicle-registrar-belyshev-graduate-work.zip

Obsah přílohy práce:

- Složka se soubory aplikace.

Příloha B – Výsledky testování optimalizačních technik

Název přílohy: optimization-scenarios.zip

Obsah přílohy práce:

- Podrobné výsledky testů optimalizačních technik Apache JMeter a Grafana

Příloha C – Vizualizace výsledků testování optimalizačních technik

Název přílohy: scenarios-optimization-results.xlsx

Obsah přílohy práce:

- Výsledky testování optimalizačních technik v tabulkách a grafech.