Politechnika Świętokrzyska w Kielcach

Technologie Obiektowe - Projekt

Temat: Obiektowe bazy danych
Porównanie wydajności ObjectDB i MySQL

Nr projektu: **12**	Autorzy**:** **Adam Markowski Bartosz Rokita**

1. Wprowadzenie

Celem projektu jest porównanie wydajności relacyjnej bazy danych w tym projekcie będzie to MySQL oraz bazy obiektowej ObjectDB z wykorzysatniem JPA oraz ObjectDB z wykorzysatniem JDO.

Porównanie będzie polegało na sprawdzeniu czasu wymaganego do odczytu konkretnej paczki danych jak również czasu wymaganego do zapisu w bazie założonej liczby rekordów. Wykonane zostanie również porównanie czasu potrzebnego na modyfikację rekordów/obiektów w bazie danych oraz czas wymagany do usunięcia danych.

2. ObjectDB

ObjectDB jest w całości napisany w Javie, a jego podstawową zasadą jest model obiektowej bazy danych. Głównym aspektem tej bazy danych jest całkowita zgodność z JPA oraz JDO. ObjectDB nie posiada wbudowanych interfejsów API. Dzięki czemu nie jest wymagana żadna obsługa ORM do wykonywania transakcji w bazie danych.

ObjectDB jak było wspomniane jest całkowicie napisany w Javie zatem osadzenie go w aplikacji Java sprowadza się do dodania tylko jednego pliku archiwum jar o nazwie objectdb.jar. Baza danych ObjectDB znajduje się w spakowanym archiwum które możemy pobrać z oficjalnej strony. Po rozpakowaniu folder bin zawiera wymagany plik jar, objectdb.jar i może być dołączony do dowolnego projektu Java jako osadzona baza danych.

ObjectDB poprzez swoją zgodność z JDO, JPA. wymagają od programisty jawnego włączenia ich do projektu Java. Zgodność ze standardem Java Persistence jest szczególnie przydatna w przypadku przenoszenia bazy danych, pod warunkiem, że obsługują one ten sam standard JDO i JPA. Zaletą korzystania z ObjectDB jest brak niezgodności między obiektami ponieważ aplikacja i baza danych ma ten sam schemat oparty na klasach. Należy również wspomnieć że ObjectDB jest przyjazny dla każdego typu aplikacji Java, zarówno samodzielnej, jak i korporacyjnej.

Głównymi zaletami ObjectDB jest obsługa od mały po duże bazy danych, mowa tu o bazach od kilobajtów po terabajty. Baza przechowywana jest w pojedynczym pliku. Maksymalnym rozmiarem pliku bazy danych jest 128 Tb. ObjectDB nie posiada limitu możliwych połączeń bazodanowych, ograniczane są jedynie przez zasoby systemu operacyjnego. Jak już wspomniane wcześniej podłączenie wymaga dodania tylko jednego pliku .jar . Ewentualne

przywracanie bazy jest łatwe do wykonania przy użyciu pliku zapasowego (recovery file). Umożliwia on przywrócenie systemu(bazy) automatyczne bądź ręczne w sytuacji gdy pojawi się awaria.

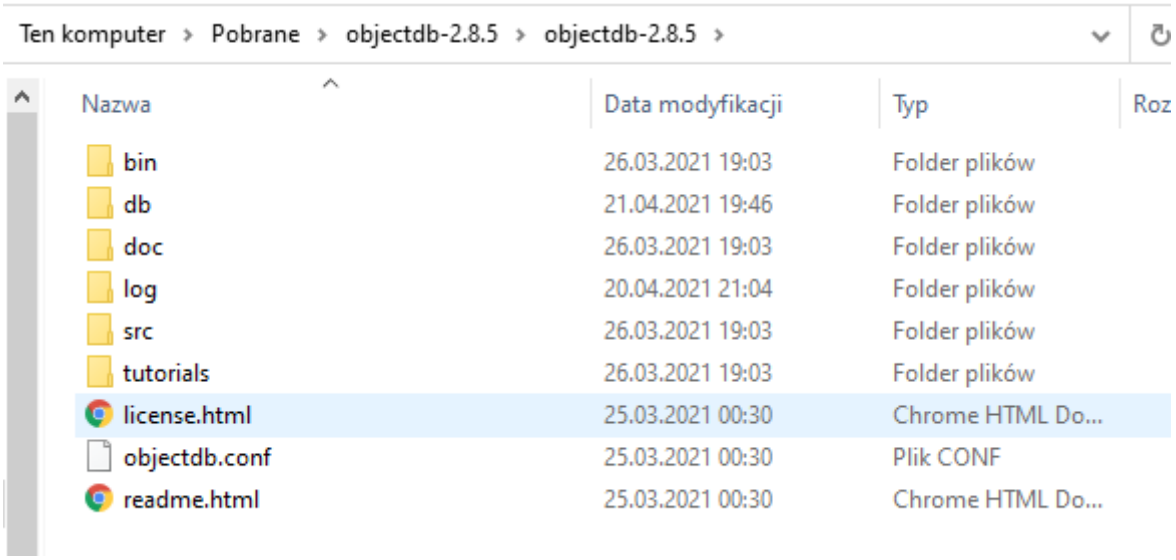
2.1 Konfiguracja ObjectDB

Podczas konfiguracji wykorzystana została wersja 2.8.5 ObjectDB oraz do obsługi Javy program NetBeans w wersji 12.1 od firmy Apache

Pierwszym krokiem jak należy wykonać jest pobranie z oficjalnej strony **ObjectDB** pliku archiwum.

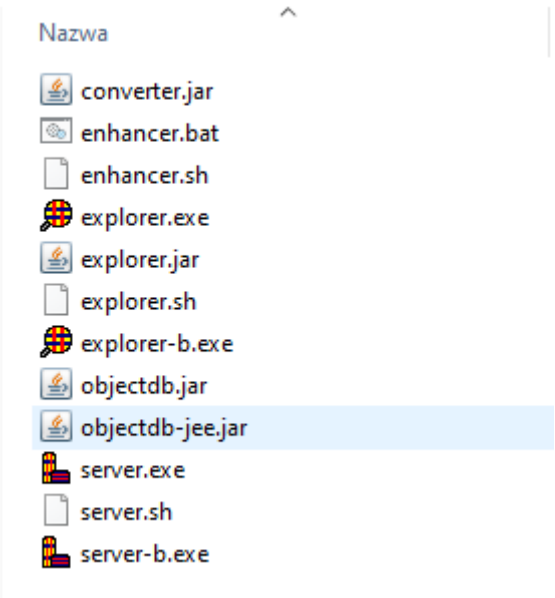
Rysunek 1

Następnie gdy mamy już plik należy go rozpakować w dowolnym miejscu na naszym komputerze.



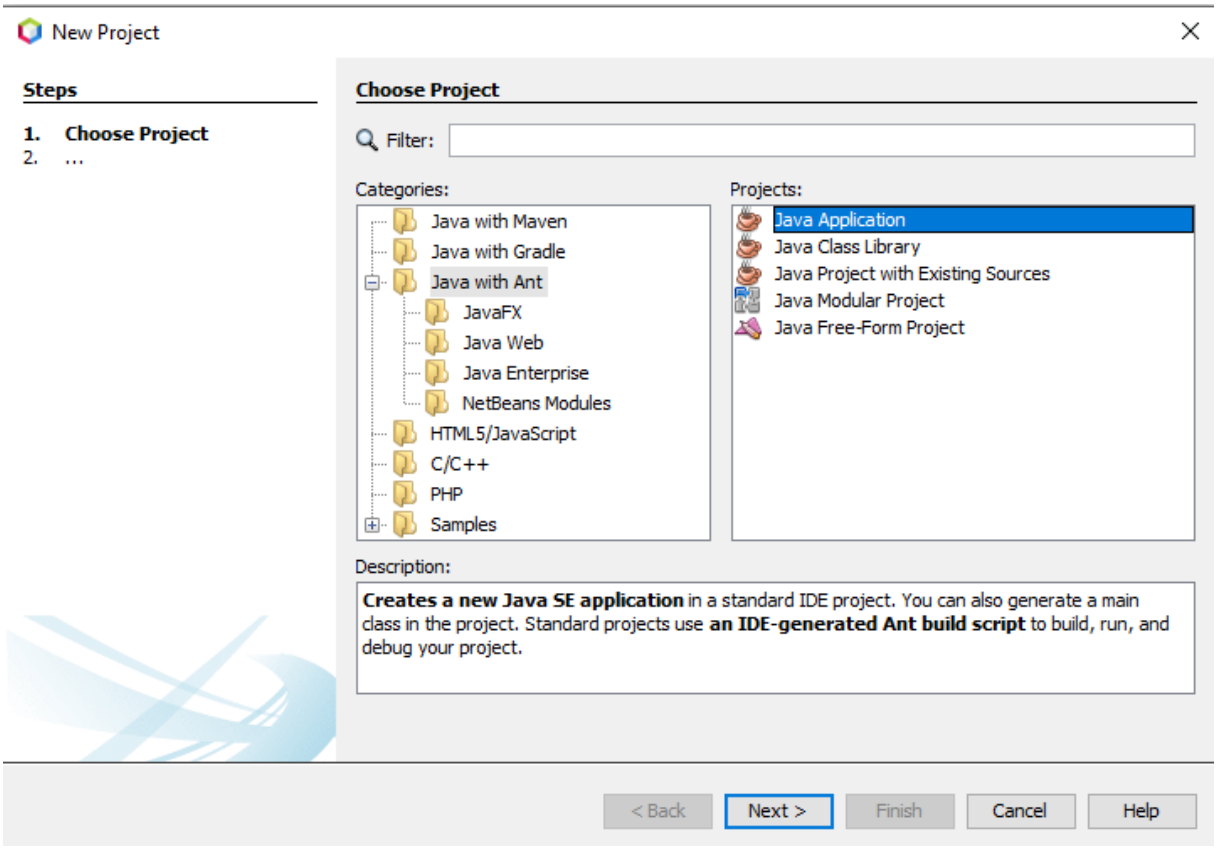
Rysunek 2

Wewnątrz pliki archiwum powinny znajdować się foldery jak na zdjęciu. Interesującym nas folderem jest folder „bin”, który powinien zawierać pliki wskazane na zdjęciu.



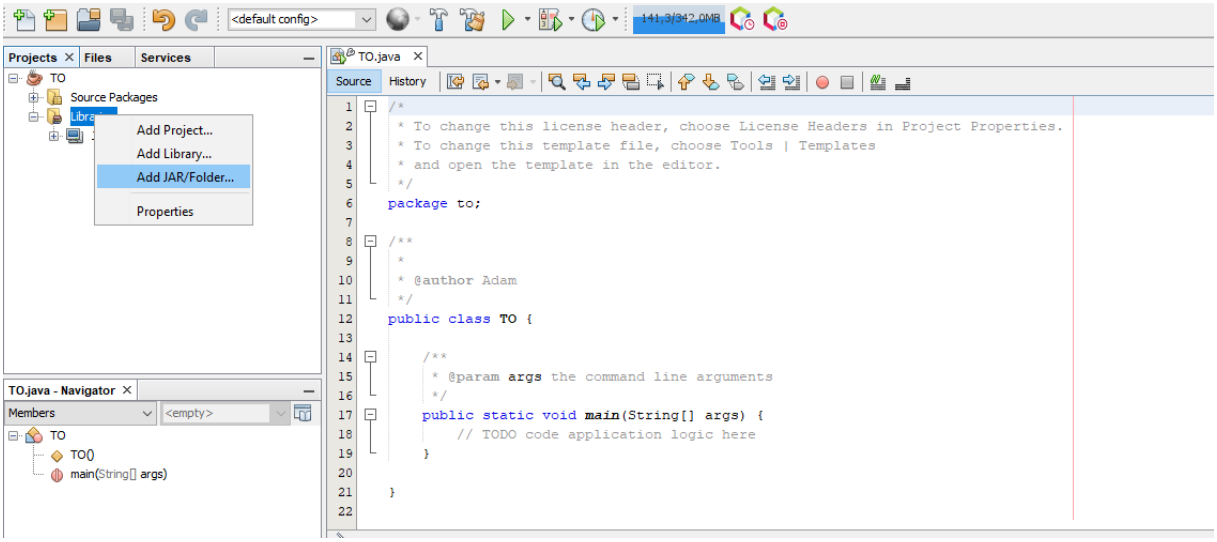
Rysunek 3

Następnie przechodzimy do wybranego środowiska dla języka Java w tym przypadku będzie to NetBeans 12.1. Po uruchomieniu tworzymy nowy projekt aplikacji Java.



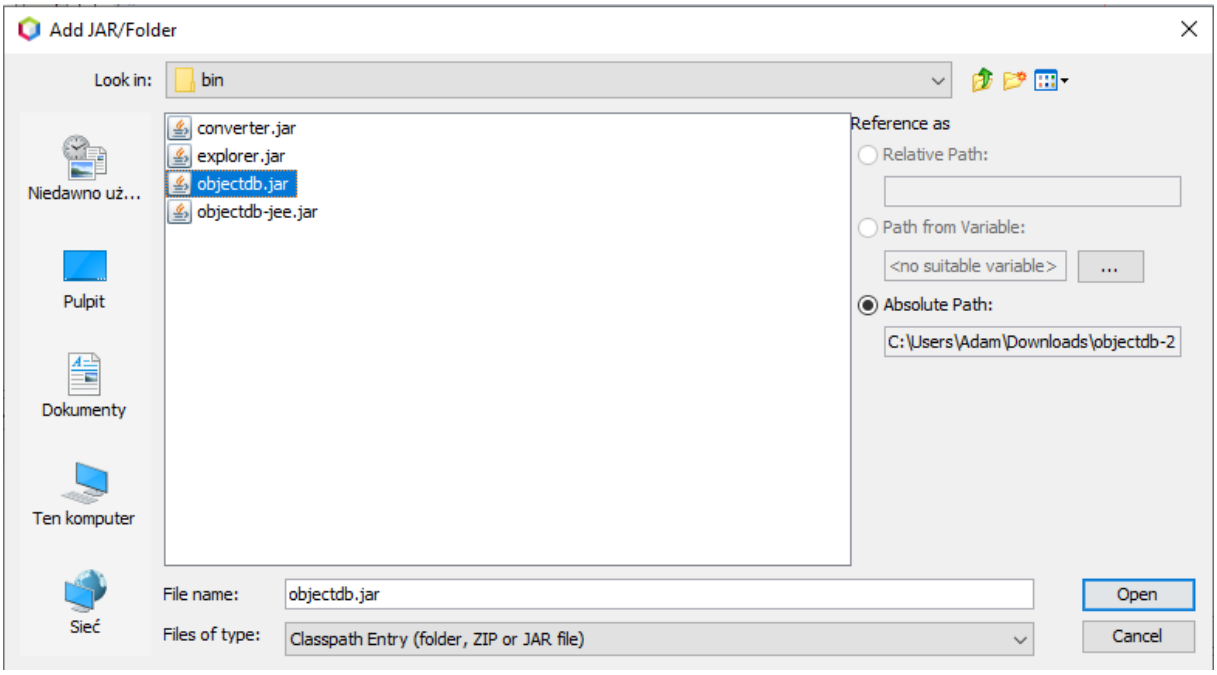
Rysunek 4

Po utworzeniu nowego projektu musimy dodać plik objectdb.jar do naszego projektu. Aby to zrobić klikamy lewym przyciskiem myszy na folder „**Libraries**" na następnie wybieramy „**Add JAR/Folder**"



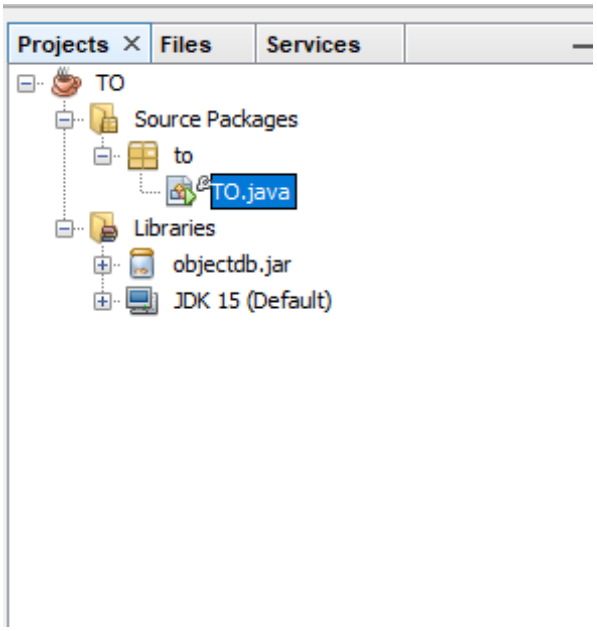
Rysunek 5

Otwiera się okno z wyborem pliku. Należy przejść do folderu w którym „rozpakowany” został nasz plik archiwum z ObjectDB. Przechodzimy do wewnętrznego folderu „bin” i wybieramy plik **objectdb.jar**.



Rysunek 6

Po dodaniu folder „**Libraries**” zostanie uzupełniony o ObjectDB.



Rysunek 7

2.2 Zarządzanie połączeniami ObjectDB

Podczas połączeń z baza danych wykorzystywany jest interfejs **Entity Manager**. Do pracy z baza danych wymagane jest utworzenie instancji **Entity Manager**. Aby utworzyć instancje Entity Managera potrzebujemy utworzyć instancję **EntityManagerFactory** która odpowiada za utworzenie pliku(o ile nie został już stworzony) bazy danych na której chcemy wykonywać operacje oraz łączy w sobie wszystkie połącznia z konkretną bazą danych .

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory("$objectdb/db/TO.odb");
EntityManager em = emf.createEntityManager();
```

Instancja **EntityManagerFactory** jest uzyskiwana poprzez użycie statycznej metody fabrycznej klasy JPA, **Persistence**.

EntityManagerFactory odpowiada za jedną konkretną bazę danych. W niektórych aplikacjach wymagane jest połączenie do wielu baz danych w takiej sytuacji tworzonych jest taka sama ilość instancji **EntityManagerFactory** jak ilość baz danych do których chcemy mieć dostęp.

Kiedy aplikacja kończy używać instancji entity dla konkretnej bazy, taka instancja powinna zostać zamknięta.

```
emf.close();
```

Następnie tworzymy połącznie do bazy z użyciem **Entity Manager'a**. Wszystkie operacje wykonywane na bazie danych są przypisane z konkretną instancją **Entity Manager'a**. Gdy chcemy wykonać operacje na bazie które zmodyfikują przechowywane dane wymagane jest aby uaktywnić „Transakcję".

```
em.getTransaction().begin();
```

Kiedy transakcja jest uruchomiona można wykorzystać metody **Entity Manager'a** takie jak persist lub remove.

Gdy połącznie z bazą nie jest nam już potrzebne należy je zamknąć przy pomocy metody „**close**".

```
em.close();
```

2.3 Modyfikacje obiektów ObjectDB

Przy dodawaniu obiektu do bazy danych należy wywołać metodę **persist** na instancji **Entity Manager'a.** następnie w parametrze musimy podać nasz nowo utworzony obiekt.

Metoda persist łączy zwykły obiekt Javy z instancją zarządzającą połączeniem z baza (Entity Manager) i ustawia status obiektu jako *Managed.* Po zakończeniu transakcji obiekt będzie dostępny w bazie danych.

```
City testcity = new City();
    testcity.setName("TEST_CITY");
    testcity.setPopulation(123123);
    testcity.setCapital(false);
    em.getTransaction().begin();
    em.persist(testcity);
    em.getTransaction().commit();
```

Aby usunąć obiekt z bazy danych, należy uzyskać zarządzany obiekt (zazwyczaj przez pobranie) i wywołać metodę remove w kontekście aktywnej transakcji:

```
em.remove(p);
```

Obiekt encji jest oznaczony do usunięcia przez metodę remove i jest fizycznie usuwany z bazy danych, gdy wykonana zostanie metoda commit na danej instancji **Entity Manager'a**.

Wszystkie operacje przypisane są do danej transakcji, aby zmiany wykonały się na danych musza zostać wysłane(commit).

```
em.getTransaction().commit();
```

Przy wykonywaniu modyfikacji obiektów nie trzeba wywoływać metody **presist**. Operacja musi odbyć się wewnątrz transakcji jednak instancja **Entity Manager'a** automatycznie wykryje zmiany w obiekcie a następnie podczas zakończenia transakcji przeniesie modyfikacje do bazy danych.

```
Country TEST_find = em.find(Country.class, "br");
System.out.println(TEST_find.getUnemployment());
em.getTransaction().begin();
TEST_find.setUnemployment(56);
em.getTransaction().commit();
System.out.println(TEST_find.getUnemployment());
```

Rezultat:

```
compile:
run:
8.1
56.0
BUILD SUCCESSFUL (total time: 2 seconds)
```

Przedstawione wyżej operacje z użyciem metod remove oraz persist musza być wykonywane w obrębie transakcji. Aby zobaczyć obiekty w bazie danych wymagane jest jedynie aktywna instancja **Entity Manager'a.**

Do wyszukiwania obiektów w bazie używany jest język JPQL (JPA Query Language). W przykładzie poniżej wykorzystany został interaface TypedQuery ze standardu JPA. Metoda getResultList wykorzystana została w przykładzie ze względu na oczekiwaną ilość zwracanych rezultatów. Wyniki prezentowane są w postaci listy.

```
    TypedQuery<Country> query =
em.createQuery("SELECT p FROM Country p", Country.class);
   List<Country> results = query.getResultList();
   for (Country p : results) {
       System.out.println(p);
    }
```

Przy oczekiwanym wyniku składającym się z pojedynczego wiersza można wykorzystać metodę getSingleResult.

```
Query ql = em.createQuery("SELECT AVG(p.unemployment) FROM Country p ");
       System.out.println("Average X: " + ql.getSingleResult());
```

Oprócz wykorzystania języka JPQL do wyszukiwania obiektów, można użyć metody **find()** która wyszuka obiekt po jego unikalnym kluczu.

```
Country TEST_find = em.find(Country.class, "br");
```

2.4 Zarządzanie połączeniami ObjectDB JDO

Nawiązywanie połączenia z wykorzystaniem JDO działa na podobnych zasadach jak w JPA. Wymagane jest utworzenie ***EntityManagerFactory*** jak również ***EntityManager'***.

Przy połączeniu z serwerem inaczej niż w przypadku JPA został stworzony obiekt przechowujący ustawienia połączenia z bazą.

```
Properties properties = new Properties();

properties.setProperty("javax.jdo.PersistenceManagerFactoryClass", "com.objectdb.jdo.PMF");

properties.setProperty("javax.jdo.option.ConnectionURL", "objectdb://underctrl.xyz:6136/world.odb");
properties.setProperty("javax.jdo.option.ConnectionUserName", "admin");
properties.setProperty("javax.jdo.option.ConnectionPassword", "admin");
```

Nazstępnie przy twrzeniu ***EntityManagerFactory*** wszystkie ustawienia zostały przekazane

```
PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory(properties);
PersistenceManager pm = pmf.getPersistenceManager();
```

Dla JDO tworzenie transakcji wygląda jak poniżej:

```
pm.currentTransaction().begin();
pm.currentTransaction().commit();
```

Gdzie pierwszy wiersz otwiera transakcję a kolejny ją zakańcza. W JPA do usuwania elementów służyło polecenie ***remove()*** natomiast dla JDO wywołujemy ***deletePersistent()***. To samo tyczy się operacji ***persist()*** w JDO wykonujemy ***makePersistent()***.

2.5 Zarządzenie bazą poprzez Explorer ObjectDB

Object DB posiada wbudowaną aplikację do przeglądania bazy danych. Przy jej użyciu możemy podejrzeć bazy danych lokalne jak również umieszczone na serwerach.

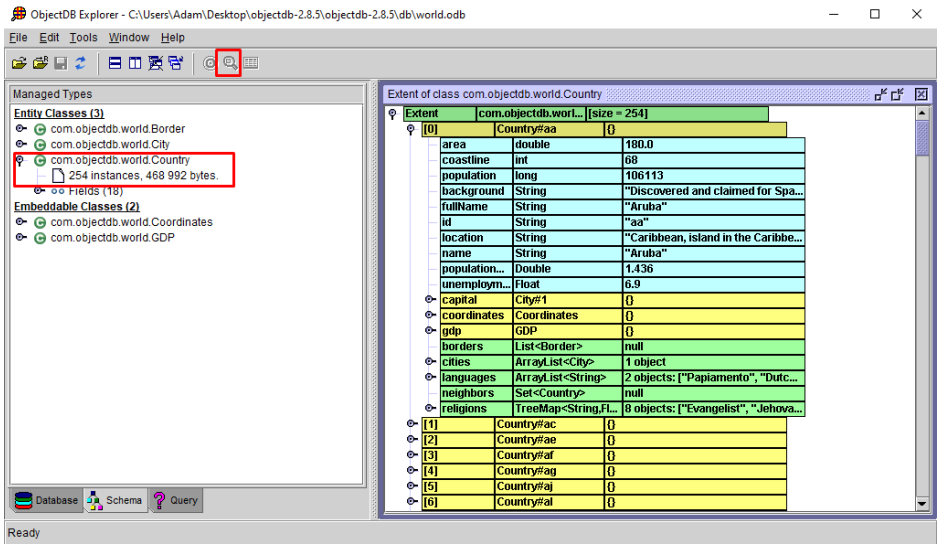
Rysunek 8

Przy użyciu opcji **Local Databases** system wyświetla nam listę baz danych znajdujących się w wybranym katalogu, domyślnie jest to katalog ObjectDB.

Open C/S Connection umożliwia nawiązanie połącznia z bazą znajdującą się na serwerze.

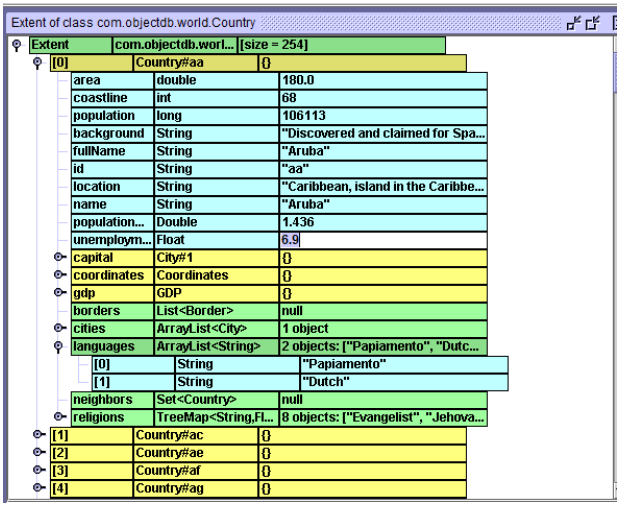
Po załadowaniu bazy uzyskujemy dostęp do encji klas zwartych w naszej bazie danych. W lewym panelu wyświetlają się nam wszystkie elementy(klasy) w bazie danych.

Po wskazaniu konkretnej encji i wybraniu z paska ikony lupy możemy podejrzeć zawartość tabeli wraz z poszczególnymi danymi.



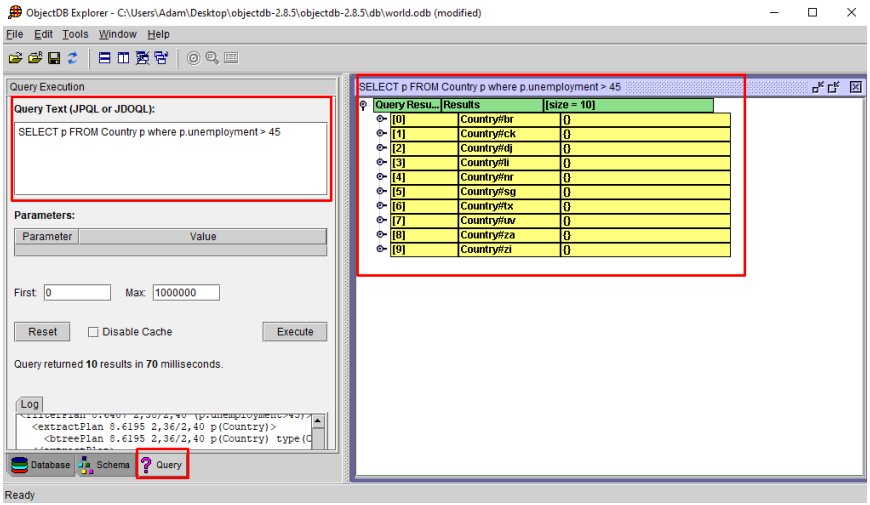
Rysunek 9

Z tego poziomu uzyskujemy również możliwość zamiany konkretnych danych dla przedstawicieli konkretnej klasy. Po dwukrotnym kliknięciu myszy na konkretną komórkę możemy bezpośrednio z klawiatury wpisać nowa wartość która automatycznie zostanie zapisana.



Rysunek 10

W dolnym menu po lewej stronie mamy również zakładkę **Query** która pozawala nam na wykonywanie zapytań w języku JPQL w celu wyświetlania przefiltrowanych danych. Po użyciu przycisku **Execute** aplikacja wyświetli w panelu po prawej stronie wynik wprowadzonego zapytania.



Rysunek 11

3. MySQL

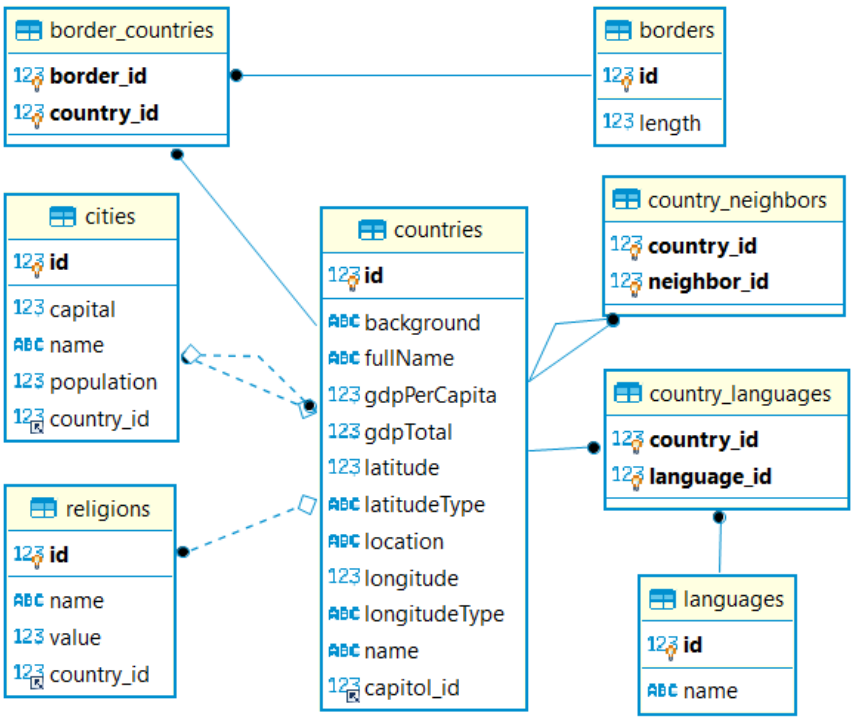
MySQL jest Systemem Zarządzania Relacyjnymi Bazami Danych. Znany i ceniony jest przede wszystkim ze względu na swoją niebywałą wydajność i szybkość działania. Świetnie

nadaje się do obsługi projektów internetowych, ale nie tylko - z powodzeniem używany jest również w wielkich projektach informatycznych.

Do realizacji dostępu do bazy w aplikacji został wykorzystany framework **Hibernate** który zapewnia przekazywanie danych, translację pomiędzy relacyjną bazą danych a obiektami w języku Java.

3.1 Baza danych dla MySQL

Przykładowa obiektowa baza danych wykorzystana w objectdb musiała być zmodyfikowana i przebudowana tak aby spełniała wymogi relacyjnej bazy danych dla której będziemy przeprowadzać testy. Klasy wbudowane oznaczone w objectdb jako **@Embeddable** musiałby być utworzone jako osobne tabele. Przykładem jest encja „***Religions***" w klasie „***Country***"



Rysunek 12

5. Testy Porównawcze

Testy wykonane zostały z podziałem na trzy kategorie:

- Wyszukiwanie obiektów z bazy danych
- Dodawanie obiektów do bazy danych
- Modyfikacja obiektów w bazie danych

Testy zostały wykonane przy użyciu obiektowej bazy danych ObjectDB z wykorzystaniem JPA i JDO oraz bazy relacyjnej MySQL. Podczas wykonywania testów każda operacja została wykonana w 5 powtórzeniach tak uśrednić czas wykonania operacji oraz wyeliminować ewentualne błędy pomiarowe. Błędy mogą wynikać przykładowo z pojedynczego zawieszenia serwera. Wszystkie operacje były wykonane dla poszczególnych tabel/encji bazy danych.

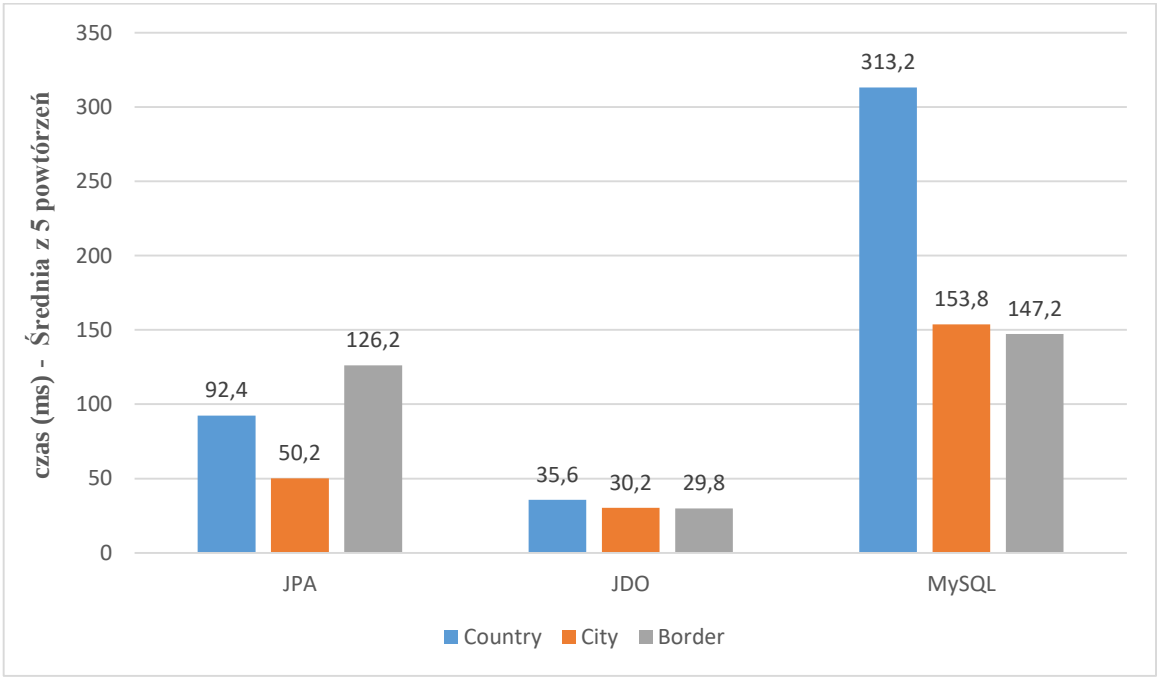
Do testów przygotowano dwa zestawy danych.

	Zestaw 1	**Zestaw 2**
Country	254	8128
Border	312	11488
City	359	9984

5.1 Pobieranie elementu

Test wykonany dla operacji pobierania 1 elementu (Zestaw 1)

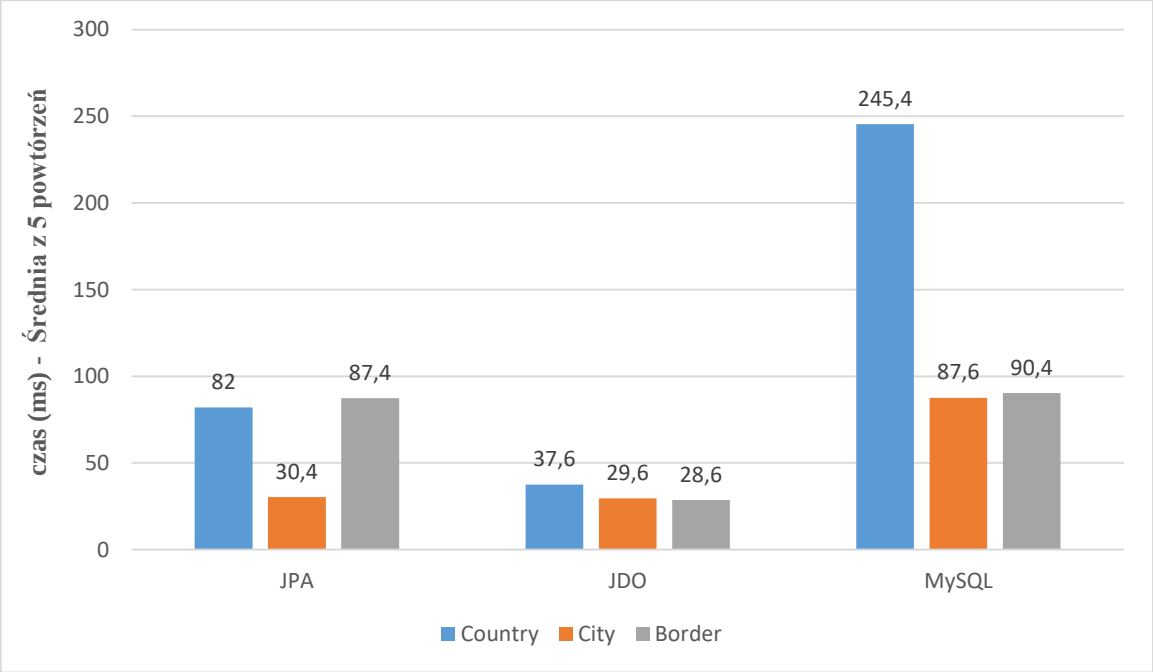
Zestaw 1	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	88	49	127	35	35	29	371	141	141
Próba 2	95	50	127	36	30	29	300	140	135
Próba 3	90	54	130	36	28	30	298	161	139
Próba 4	100	45	124	35	28	29	304	166	155
Próba 5	89	53	123	36	30	32	293	161	166



Wykres 1

Test wykonany dla operacji pobierania 1 elementu (Zestaw 2)

Zestaw 2	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	71	30	82	42	27	27	268	88	94
Próba 2	78	33	87	35	28	29	237	82	85
Próba 3	95	29	90	35	33	28	253	78	87
Próba 4	86	30	92	37	30	30	229	90	95
Próba 5	80	30	86	39	30	29	240	100	91



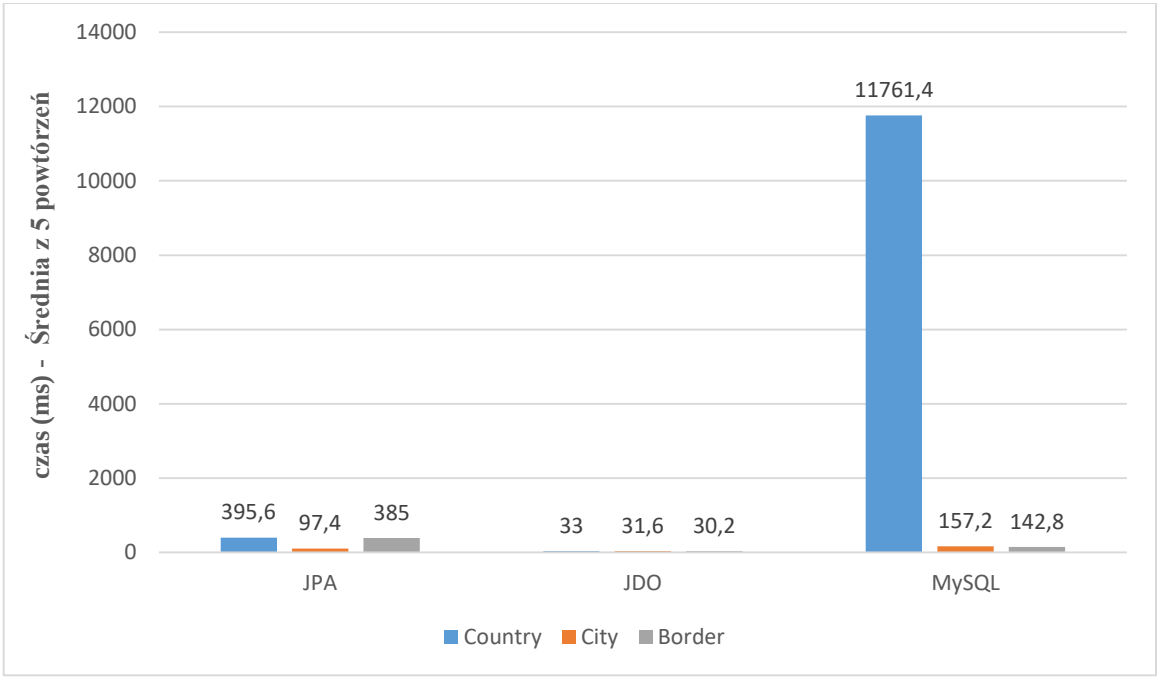
Wykres 2

Bazując na wykresie nr 1 oraz nr 2 można zauważyć znaczącą różnicę czasu potrzebnego do pobrania jednego elementu. Porównując uzyskane czasy dla klasy ***„Country”***, różnica JPA do MySQL wynosi 220,8 ms co stanowi około 338,9 % wzrostu co można przełożyć jako trzykrotne zwiększenie czasu dostępu. Między JPA oraz JDO różnica wynosi zaledwie 32,6 ms na korzyść JDO. Dla klasy ***„City”*** różnica wynosi 103.6 ms pomiędzy JPA oraz MySQL.

Podsumowując czasu dostępu jest znacząco mniejszy dla obiektowej bazy danych w konfrontacji z bazą relacyjną. Dodatkowo ilość danych nie wpłynęła na czas potrzebny na wykonanie operacji dla bazy obiektowej.

Test został wykonany dla pobierania wszystkich elementów (Zestaw 1)

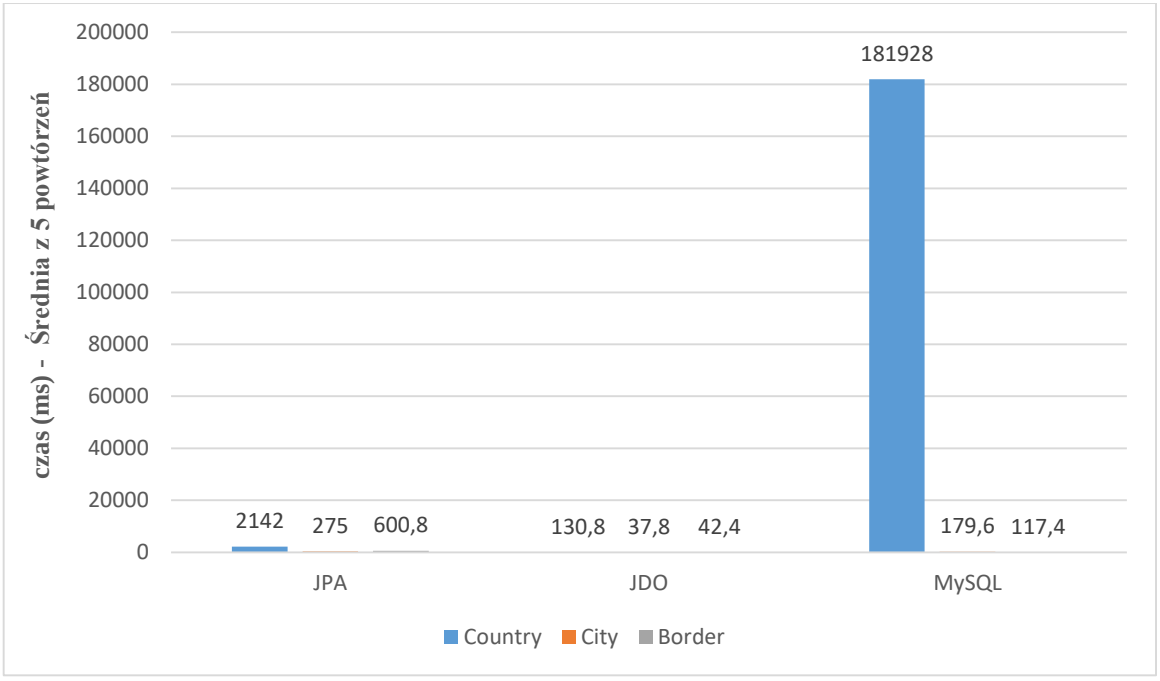
Zestaw 1	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	391	103	390	37	31	30	11994	162	144
Próba 2	378	103	377	37	32	30	11396	145	141
Próba 3	403	88	355	30	31	33	11540	159	139
Próba 4	386	93	412	32	32	29	11992	161	150
Próba 5	420	100	391	29	32	29	11885	159	140



Wykres 3

Test został wykonany dla pobierania wszystkich elementów (Zestaw 2)

Zestaw 2	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	2177	291	652	142	39	49	189724	191	116
Próba 2	2271	266	604	133	36	43	179621	161	115
Próba 3	2110	248	530	144	37	39	180621	199	120
Próba 4	2110	301	578	119	37	41	181023	171	126
Próba 5	2042	269	640	116	40	40	178651	176	110

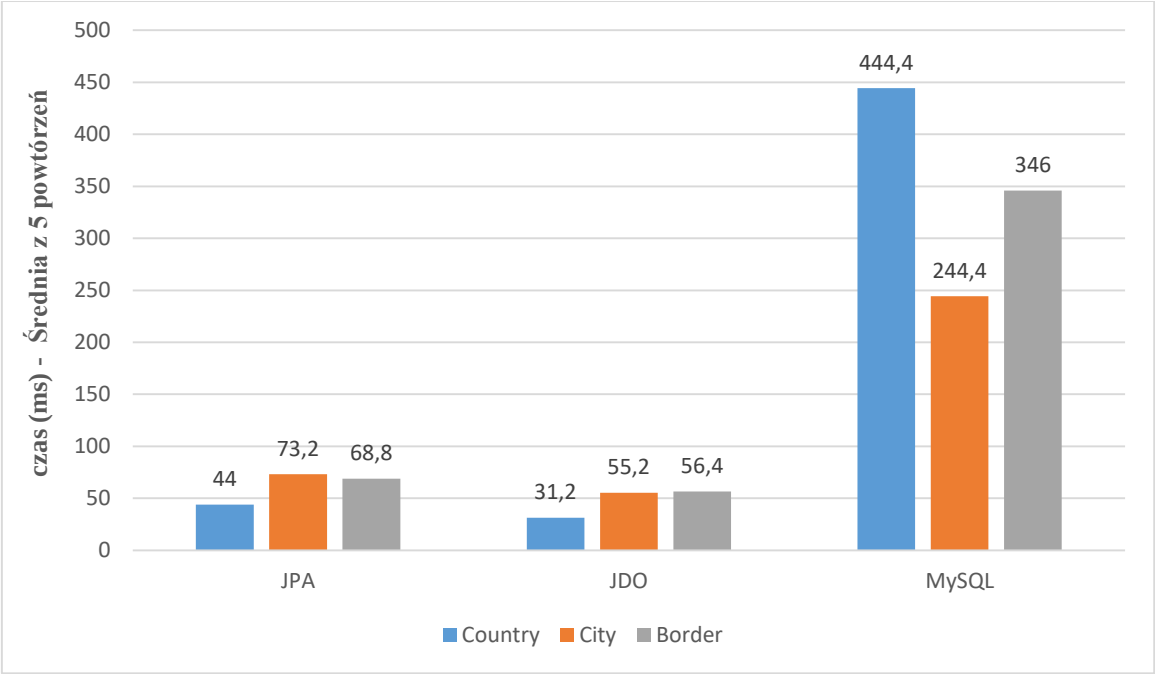


Wykres 4

Rozpatrując wykres nr 3 oraz nr 4 widać ogromną różnicę potrzebnego czasu dostępu do danych. Między JPA oraz JDO wynosi on zaledwie 339 ms dla zestawu 1. Natomiast czas potrzebny na pobranie wszystkich elementów dla klasy „Country" w MySQL wynosi 11761,4 ms co stanowi około 200-krotne zwiększenie czasu dostępu. Ilość danych znacząco wpłynęła na czas potrzeby na pobranie wszystkich elementów z bazy zarówno dla obiektowej jak i relacyjnej bazy danych. Różnica miedzy pobieraniem tabeli „Country" a pozostałymi w bazie MySQL występuje ze względu na zapis Hibernate w pamięci obiektów wywołanych podczas pierwszego zapytania do bazy. Przez co pozostałe wyszukiwania odbywają się znacznie szybciej

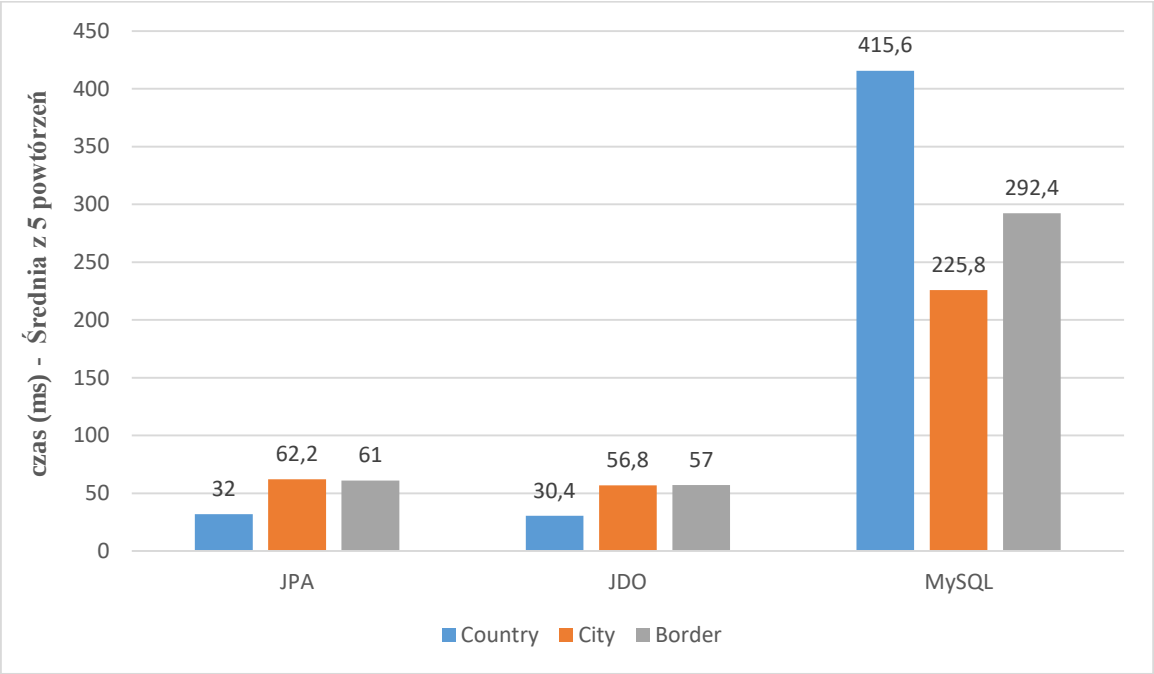
5.2 Dodawanie elementu

Zestaw 1	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	42	70	56	31	54	55	450	235	361
Próba 2	42	72	69	32	56	56	461	277	333
Próba 3	45	72	79	30	55	56	435	241	359
Próba 4	41	76	81	31	55	56	441	234	346
Próba 5	50	76	59	32	56	59	435	235	331



Wykres 5

Zestaw 2	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	36	63	58	30	55	56	418	216	304
Próba 2	34	58	56	31	55	56	410	237	283
Próba 3	30	58	81	30	57	59	430	228	291
Próba 4	37	59	50	31	60	57	402	217	301
Próba 5	23	73	60	30	57	57	418	231	283



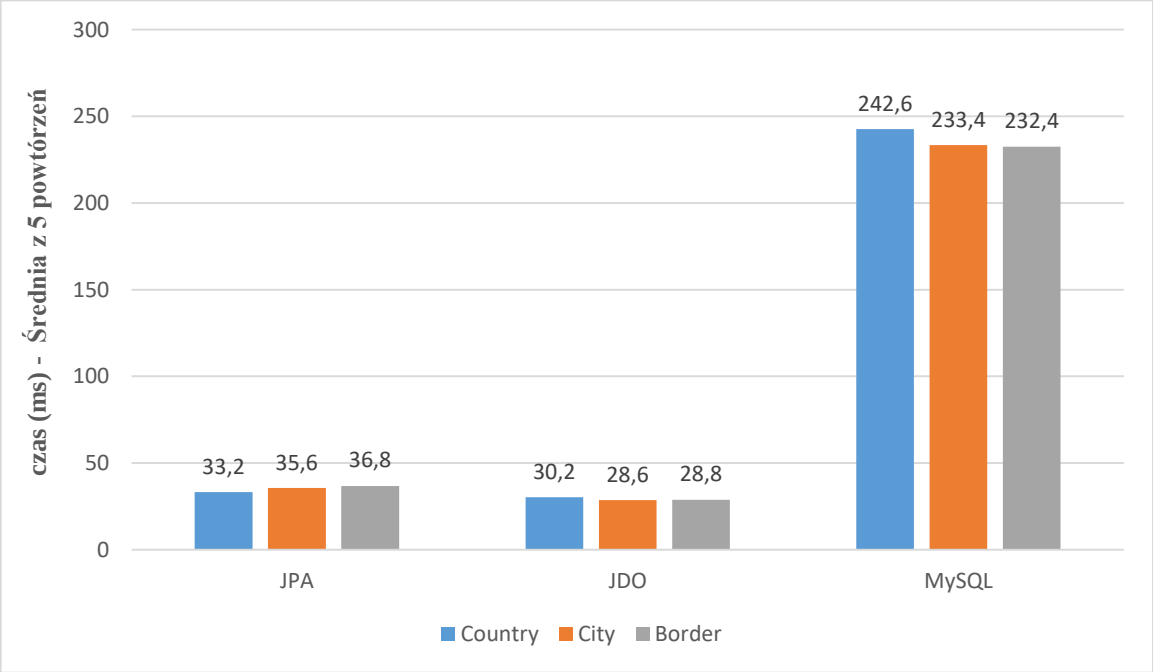
Wykres 6

Na wykresie nr 5 oraz nr 6 zostały zaprezentowane czasy potrzebne na utworzenie elementu. Jak można zaobserwować i w tym przypadku czasy wymagane na dodanie elementu są znacząco wyższe dla bazy relacyjnej.

Ilość danych nie wypłynęła na czas wykonywanej operacji.

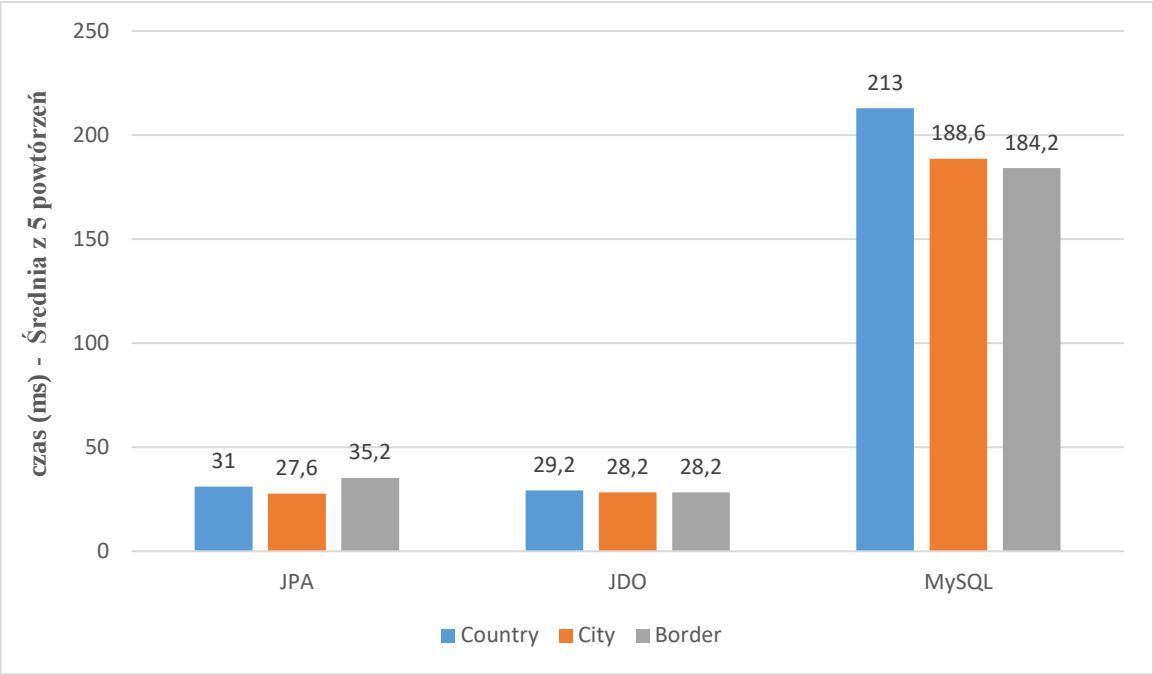
5.3 Edycja elementu

Zestaw 1	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	38	37	31	31	29	29	242	234	236
Próba 2	32	31	39	30	29	29	245	230	231
Próba 3	40	36	37	32	29	29	241	234	235
Próba 4	35	38	47	29	28	29	246	230	234
Próba 5	21	36	30	29	28	28	239	239	226



Wykres 7

Zestaw 2	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	33	22	44	28	28	26	201	197	189
Próba 2	34	31	32	29	28	28	212	179	179
Próba 3	21	40	30	31	28	29	221	186	184
Próba 4	36	16	40	29	28	29	232	181	191
Próba 5	31	29	30	29	29	29	199	200	178



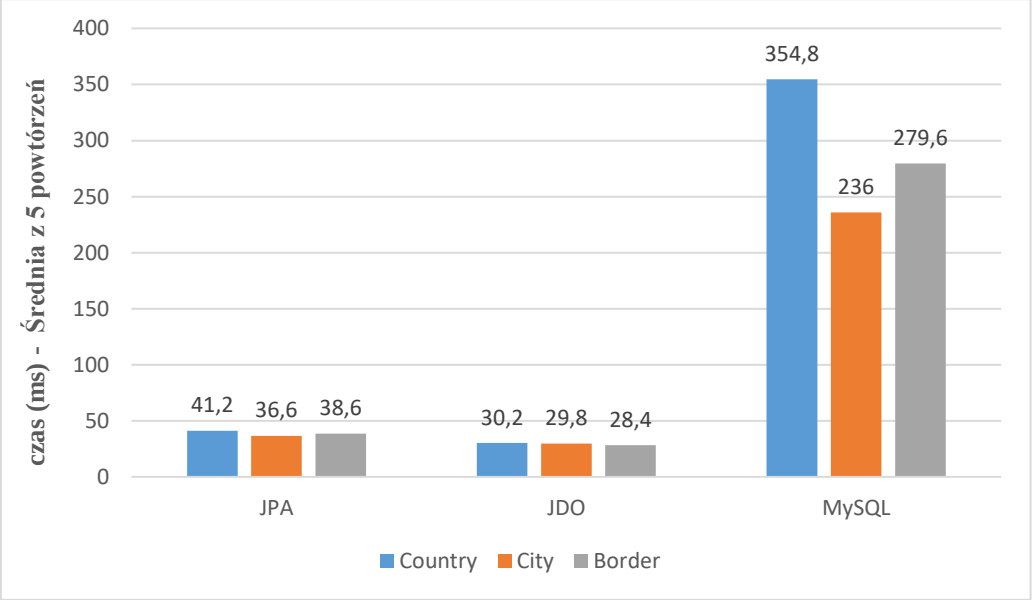
Wykres 8

Z wykresu nr 7 oraz nr 8 jasno widać zwiększony czas potrzebny na edycje elementu w przypadku MySQL. Pomiędzy JPA oraz JDO różnice są niewielkie z korzyścią po stronie JPA.

Zakres danych nie zwiększył wymaganego czasu na przetworzenie operacji dla obiektowych baz danych

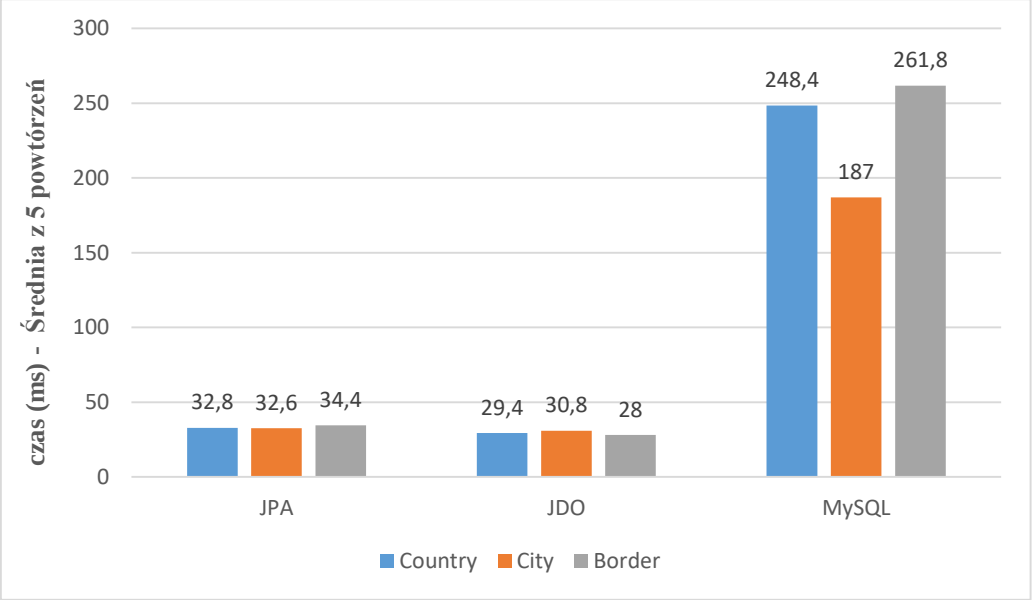
5.4 Usuwanie elementu

Zestaw 1	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	40	41	38	30	30	28	344	245	279
Próba 2	39	39	42	29	30	28	328	233	301
Próba 3	48	35	38	31	31	29	428	237	274
Próba 4	42	32	35	30	29	28	347	235	275
Próba 5	37	36	40	31	29	29	327	230	269



Wykres 9

Zestaw 2	JPA			JDO			Hibernate		
	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER	COUNTRY	CITY	BORDER
Próba 1	71	21	45	27	29	26	252	192	278
Próba 2	22	40	37	29	29	28	236	192	230
Próba 3	20	41	30	30	29	29	251	189	257
Próba 4	20	30	30	32	37	28	260	178	262
Próba 5	31	31	30	29	30	29	243	184	282



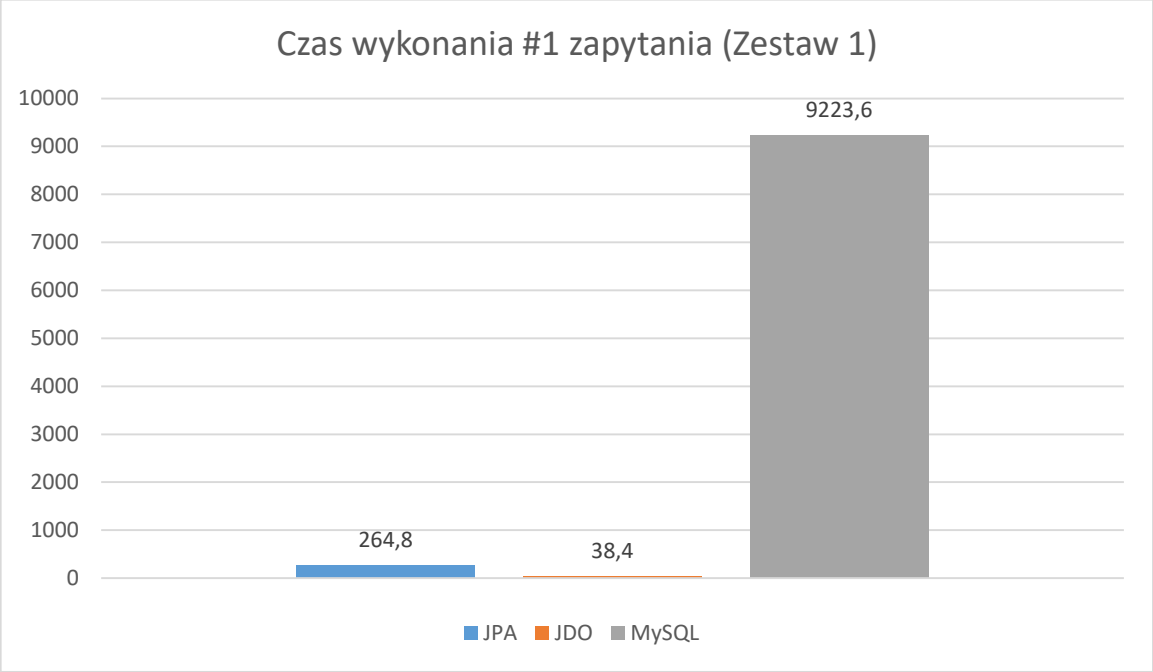
Wykres 10

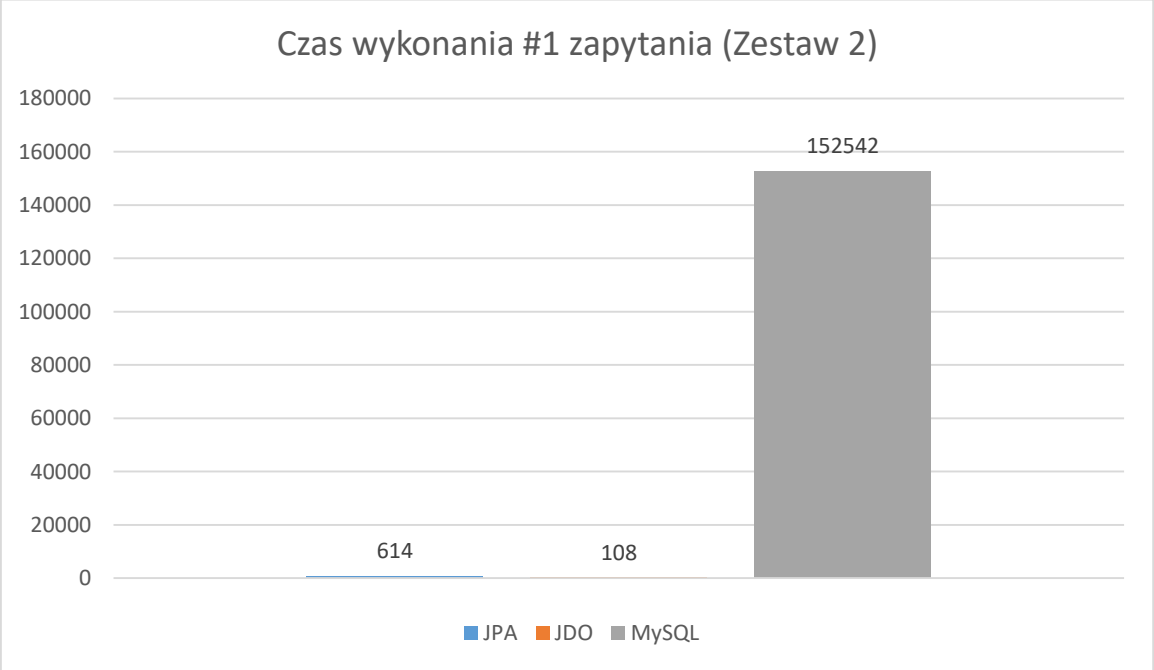
Podobnie jak w poprzednich przypadkach łatwo zauważalne są zwiększone czasy dostępu dla MySQL we wszystkich tabelach. Usuwanie elementów w JPA oraz JDO ze względu na możliwe błędy pomiarowe można uznać za zbliżone do identycznych bez znaczenia statystycznego.

5.5 Zapytania SQL

Pierwsze zapytanie dotyczy wyszukiwania obiektów z klasy ***Country*** która posiada wiązanie z obiektem klasy ***City*** dla nazwy obiektu posiadającego literę ***a*** w nazwie.

SELECT c FROM Country c JOIN c.cities ci WHERE ci.name like '%a%' GROUP BY c

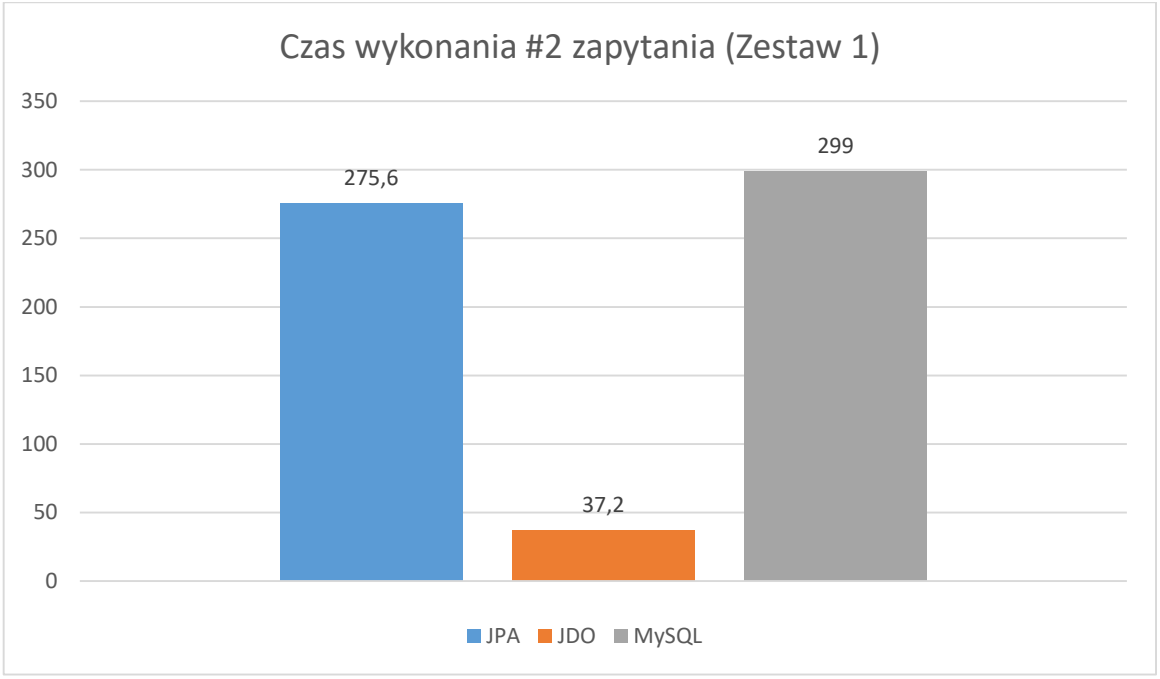


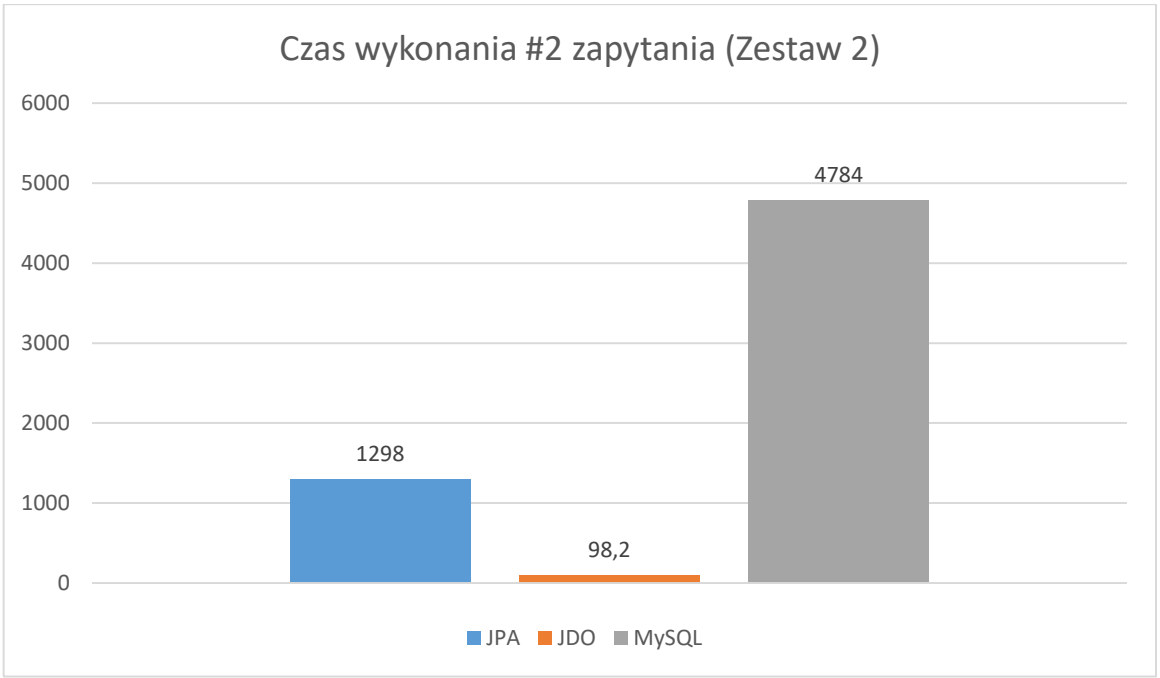


Ilość danych wydłużyła czas wykonania zapytania około 3-krotnie dla JPA oraz JDO. MySQL potrzebował aż 152s o przetworzenia tego zapytania.

Drugie zapytanie wyciąga obiekty klasy ***Country*** które posiadają długość swoich granic powyżej 1000.

SELECT c FROM Country c JOIN c.borders b WHERE b.length > 1000 GROUP BY c





Porównując oba wykresy widzimy ze wzrost liczby danych znacząco wydłużył czas wykonania zapytania dla obiektowej bazy z użyciem JPA. Dla JDO nastąpił około 3-krotny wzrost potrzebnego czasu jednak nie przekroczył on 100ms.

6. Podsumowanie

Podczas przygotowywania projektu poznaliśmy wiele ciekawostek dotyczących ObjectDB jak i samych obiektowych baz danych. Podczas przygotowywania testów musieliśmy się zagłębić w język zapytań JPQL oraz JDOQL aby przygotować kilka prostych zapytań. Poznaliśmy wiele różnic i ograniczeń które występują w tych językach.

Po przeprowadzonych testach oczywistym wnioskiem jaki można wysunąć jest znacząca różnica czasowa w podstawowych operacjach takich jak dodawanie, usuwanie lub modyfikacja danych. Obiektowa baza danych jest oczywistym wyborem dla rozwiązań/aplikacji obiektowych ze względu na identyczne/naturalne odwzorowanie klas. Bez sprzecznie dominuje nad relacyjnymi bazami danych, w których znaczna część czasu poświęcona jest na przetworzenie/zamianę tabel na obiekty

Czy opracowywaniu testów porównanie dotyczyło również obsługi ObjectDB z wykorzystaniem JPA oraz JDO. W tym przypadku testy pokazały zauważalną różnice przy pobieraniu elementów na korzyść JDO. Co do operacji modyfikacji, tworzenia oraz usuwania elementów różnice są nieznaczne z przewagą JPA na poziomie około 12,9 ms.

Literatura

[1] ObjectDB
https://www.objectdb.com/

[2] MySQL Manual
https://dev.mysql.com/doc/refman/8.0/en/