Taka ogólna prośba - jak już się uczycie, a na wejściówkach pojawiają się nowe pytania, to je dodajcie tutaj i opracujcie. Nie dość, że będziecie mieć wszystko w jednym miejscu to przysłużycie się społeczności. Amen!

Laboratorium 1 - Hibernate

Co to jest JPA?
JPA - Java Persistence API - specyfikacja mapowania obiektowo-relacyjnego, implementowana przez Hibernate, dostępna w Java SE i EE.
Oficjalny standard mapowania obiektowo-relacyjnego (ORM) firmy Sun Microsystems dla języka programowania Java.

Co to jest lazy loading?
Obiekt powiązany nie jest ładowany automatycznie z obiektem agregującym, ale dopiero po wywołaniu jego gettera.
W przypadku kolekcji, ładowane są mniejsze partie danych tak, żeby odpowiadać potrzebom aplikacji - zabezpieczenie przed zalaniem aplikacji miliardem rekordów.
*Jeśli obiektów jest mało to warto wyłączyć lazy loading, bo generuje on osobne -zapytanie przy każdym odwołaniu.

Jak w Hibernate konfigurujemy połączenie z bazą?

- W konfiguracyjnym pliku XML “hibernate.cfg.xml” lub pliku własciwości “hibernate.properties”.Przykładowo podajemy tam connection.driver_class, connection.url, użytkownika i jego hasło do bazy.

Jakie modele dziedziczenia wspiera Hibernate?

- Osobna tabela dla każdego elementu hierarchii.(Person, Customer, Employee)

- Tabela dla każdej konkretnej klasy.(Customer, Employee, bo Person jest abstract)
- Jedna tabela dla całej hierarchii dziedziczenia.(jedna dla wszystkich, dużo nulli)

Jakie sposoby robienia zapytań udostępnia Hibernate?
- SQL
- HQL
- Criteria(bez języka zapytań)

Jakie wymagania musi spełniać klasa persystentna dla Hibernate?
- Najlepiej POJO - Plain Old Java Object
- Wymagany jest konstruktor bezargumentowy
- Wskazane settery i gettery dla atrybutów utrwalanych
- Musi definiować identyfikator
- Może zawierać inne atrybuty i metody z implementacją logiki

Metoda w Hibernate którą łączymy się z bazą.
```
Configuration config = new Configuration().configure();
SessionFactory sessionFactory = config.buildSessionFactory();
Session session = sessionFactory.getCurrentSession();
session.beginTransaction();
// albo sessionFactory.openSession(). ogólnie chodzi o open/getCurrentSession()
```

Na czym polega niezgodność modeli relacyjnego i obiektowego?
- Podstawowa jednostka modelowania: dana vs klasa
- Relacja o krotności 1-1(czy wg **paradygmatów** bazie powinno być wiele tabel?)
- Dziedziczenie (OO reprezentuje zarówno „ma” jak i „jest” - model relacyjny tylko „ma”)
- Porównywanie (równość referencji, wartości, identyfikatorów)
- Relacje m-n (brak w modelu relacyjnym)

Na czym polegają niezgodności w mapowaniu obiektowo-relacyjnym relacji m-n?
W paradygmacie obiektowym relacje m-n są naturalne dzięki referencjom, w paradygmacie relacyjnym aby otrzymać relacje m-n musimy tworzyć tabele pośredniczące (łącznikowe).

Napisać proste zapytanie w HQL. Afair, pobrać wszystkie osoby o danym nazwisku.
from Person p where p.surname =[nazwisko]

Napisz polecenie HQL wybierające z tabeli [nazwa] samochody o kolorze [kolor]. Kolor w tabeli w stringu.
from [nazwa] car where car.color=[kolor]

Stany obiektów trwałych.

- Transient - nowa instancja klasy, nie jest powiązana z sesją i nie ma reprezentacji w bazie.
- Persistent - ma reprezentacje w bazie danych, powiązana z sesją. Wszelkie zmiany stanu obiektu zostaną zapisane w bazie automatycznie. Jeśli chcemy powiązać obiekt transient z bazą żeby był persistent używamy session.persist(obj) albo session.save(obj). Save od razu zwraca identyfikator, persist nie.
- Detached - gdy zamkniemy sesje(session.close()) to instancja przestaje być z nią powiązana. Dalsze zmiany nie zostaną zapisane w bazie. Oczywiście można ją później znowu powiązać z kolejną sesją.

Wady i zalety dziedziczenia "osobna tabela dla każdego poziomu hierarchii".

- Zalety:
  - Pełna normalizacja
  - Zgodność modeli
- Wady:
  - Słaba wydajność - dużo joinów

Wady i zalety modelu "jedna tabela dla wszystkich klas w hierarchii dziedziczenia".

- Zalety:
  - Wydajność !
- Wady:
  - Dużo wartości pustych
  - Brak opcji "not null"

Wymień 3 adnotacje JPA służące do utrwalania obiektów.
// ogolnie wszystkie adnotacje piszemy nad danym polem, klasą etc

- @Entity - dodajemy jak coś zapisujemy w bazie
- @Table - do jakiej tabeli chcemy zapisać Entity
- @Column - kolumna w tabeli (nad polami)
- @Id

- @GeneratedValue - automatyczna generacja, jest wiele gotowych wzorców(w tym oczywiscie standardowa inkrementacja) + możliwość zdefiniowania własnego algorytmu

Wymień główne zadania ORM.
ORM - Object-Relational Mapping = mapowanie pomiędzy obiektami a relacjami.
Główne zadania to :

- sposób definiowania mapowania,
- CRUD- create read update delete
- wyszukiwanie
- transakcje/optymalizacje

Co to jest JDO?
Java Data Object - specyfikacja, ze strony Oracle: standard interface-based Java model abstraction of persistence, developed under the auspices of the Java Community Process.

Wady i zalety stosowania bibliotek ORM.

+ Brak skomplikowanych zapytań
+ Nie trzeba tworzyć stored procedur po stronie bazy
- Jeżeli mamy dużo danych może być bardzo niewydajne
- Trzeba cały czas kontrolować zgodność bazy z obiektami hibernate'owymi

Na czym polegają niezgodności w mapowaniu obiektowo-relacyjnym 1-1?
Hibernate tworzy jedną tabelę zamiast dwóch (tak było na wykładzie, jak się mylę to zmazać)

Klasa w Hibernate, która łączy się z bazą.
Session

Co to jest EJB?

- Enterprise Java Beans
- Specyfikacja, nie implementacja
- Ziarna persystentne
- Bardzo niewydajne, dla każdego ziarna tworzone osobne połączenie

Co to jest Criteria API?
The Criteria API is used to define queries for entities and their persistent state by creating query-defining objects. Criteria queries are written using Java programming language APIs, are typesafe, and are portable. Such queries work regardless of the underlying data store.

Klasa, która odpowiada za pobieranie danych z bazy.
http://stackoverflow.com/questions/20781286/how-to-fetch-data-from-database-in-hibernate#20781443
‘List<Employee> list = s.createCriteria(Employee.class).list();’
Session albo Criteria

Laboratorium 2 - Linq, Entity framework

Entity framework - to obiektowo-relacyjny mapper pozwalający programistom pracować z relacyjnymi danymi używając obiektów. Eliminuje potrzebę pisania kodu dostępu do danych. Rekomendowana przez microsoft technologia dostępu do danych dla nowych aplikacji.

Są **trzy** główne **podejścia** w tworzeniu aplikacji bazodanowych z wykorzystaniem EF:

1. Codefirst - polega na napisaniu klas POCO, a następnie na ich podstawie wygenerowaniu bazy danych
2. Model first - polega na zaprojektowaniu modelu w EDMX (Entity Data Model XML) lub w designerze, a następnie na jego podstawie wygenerowaniu DDL tworzącego tabele i powiązania w bazie
3. Database first - Polega na wygenerowaniu EDM na podstawie struktury bazy danych

Schemat działania przy podejściu Codefirst:

1. Tworzymy klasę z property które mają być mapowane na atrybuty w bazie danych. Np. jak niżej:

```
class Post {
        public int PostId { get; set; }
        public string Title { get; set; }
        public string Content { get; set; }
}
```

2. Tworzymy klasę kontekstową dziedziczącą po DbContext zawierającą zbiory (DbSet) kolekcji, które mają być zarządzane przez EF:

```
class BlogContext:DbContext {
        public DbSet<Post> Posts { set; get; }
}
```

Zapytania o blogi:

```
// pobierz wszystkie blogi z bazy i posortuj malejąco po nazwie
IQueryable<string> query = from b in db.Blogs
```

```
                    orderby b.Name descending
                    select b.Name;
//wypisz wszystkie pobrane blogi
foreach (var item in query) { Console.WriteLine(item); }

Console.WriteLine("Blogi z method synthax");
IQueryable<string> blogNames = db.Blogs.Select(b=>b.Name);
//wypisz wszystkie pobrane blogi z lambdy
foreach (String bl in blogNames) { Console.WriteLine(bl); }
```

LINQ (.NET Language-Integrated Query) to zestaw deklaratywnych zapytań ogólnego przeznaczenia, służących do dostępu do różnych źródeł danych z aplikacji obiektowych.

LINQ2Entites – framework umożliwiający zadawanie zapytań LINQ do źródeł/modeli danych opartych o Entity Framework

Wyrażenia lambda:

- Postać ogólna: **(parametry) => wyrażenie** nawiasy są pomijalne dla 1 parametru gdy jest ich więcej są obowiązkowe (a argumenty oddzielamy przecinkami).
- Typy argumentów można podać explicite (przydatne gdy kompilator nie umie wywnioskować typu)
- W C# operator =>to tzw operator lambda („przechodzi w”)
- Lewa strona to zmienna wejściowa
- „Ciało” wyrażenia lambda to typowe wyrażenia mogące zawierać stałe, zmienne, wywołania metod i wyrażenia logiczne
- Wartością zwracaną jest wartość wyrażenia
- Operator=>jest operatorem prawostronnym z takimi samymi zasadami priorytetowania i precedencji jak operator przypisania (=)
- Wyrażenia lambda mogą zostać przypisane do typu „delegatowego” (del myDelegate = x => x * x;)
- Lub do typu Expression (Expression<del> myET = x => x * x;)
- Wyrażenia lambda mogą składać się z wielu „operacji” (choć zwykle nie więcej niż dwie -trzy). Umieszczamy je w klamrach i oddzielamy średnikami.
- Wyrażenia lambda mogą być przetwarzane asynchronicznie (słowa kluczowe **async** [przed nawiasem z parametrami, bądź wywołaniem] i **await** [używane do czekania na wykonanie wyrażenia asynchronicznego)
- Używamy ich jako parametrów w method based syntax.

<**składnia query expression syntax**> slajdy 14 - 44 - krótkie opracowanie z najważniejszymi klauzulami mile widziane (i przykładem je ilustrującym [najlepiej

jeden przykład ilustrujący kilka klauzul, żeby opracowanie się przesadnie nie rozrosło])

<**składnia method based syntax**> slajdy 54 - 79 - krótkie opracowanie z najważniejszymi klauzulami mile widziane (i przykładem je ilustrującym [najlepiej jeden przykład ilustrujący kilka klauzul, żeby opracowanie się przesadnie nie rozrosło])

Deffered query execution - zapytania L2E są wykonywane zawsze kiedy iterujemy po zmiennej zapytania (tzw. deferred execution)

- W przypadku zapytań zwracających sekwencje/kolekcje wartości zmienna zapytania nigdy nie przechowuje rezultatów –wyłącznie samo zapytanie.
- Wykonanie takiego zapytania jest odraczane do momentu iterowania po zmiennej zapytania (foreach)
- Oznacza to że mamy możliwość wykonywania zapytania tak często jak chcemy
- Podejście użyteczne kiedy nasza baza danych jest/może być updatowana przez inne aplikacje klienckie
- pozwala na łączenie wielu zapytań
- pozwala na rozszerzanie zapytań

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    IQueryable<Product> productsQuery =
        from p in context.Products
        select p;

    IQueryable<Product> largeProducts = productsQuery.Where(p => p.Size == "L");

    Console.WriteLine("Products of size 'L':");
    foreach (var product in largeProducts)
    {
        Console.WriteLine(product.Name);
    }
}
```

Immediate Query Execution

- Jeżeli pytanie zwraca pojedynczą wartość (np wynik operacji agregujących) jest ono wykonywane od razu (żeby obliczyć wartość zagregowaną musimy stworzyć sekwencję/kolekcję do jej wyliczenia)
- Natychmiastowe wykonanie może zostać także wymuszone. Jest to przydatne kiedy chcemy np zachować wyniki zapytania.
- Aby wymusić natychmiastowe wykonanie zapytania nie zwracającego pojedynczej wartości należy wywołać na zapytaniu jedną z metod ToList,ToDictionary lub ToArray

ToArray. Natychmiastowa ewaluacja wyrażenia i konwersja do tablicy:

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;

    Product[] prodArray = (
        from product in products
        orderby product.ListPrice descending
        select product).ToArray();

    Console.WriteLine("Every price from highest to lowest:");
    foreach (Product product in prodArray)
    {
        Console.WriteLine(product.ListPrice);
    }
}
```

ToDictionary. Natychmiastowa ewaluacja wyrażenia i konwersja do postaci słownika:

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;

    Dictionary<String, Product> scoreRecordsDict = products.
            ToDictionary(record => record.Name);

    Console.WriteLine("Top Tube's ProductID: {0}",
            scoreRecordsDict["Top Tube"].ProductID);
}
```

ToList. Natychmiastowa ewaluacja wyrażenia i konwersja do Listy:

```
using (AdventureWorksEntities context = new AdventureWorksEntities())
{
    ObjectSet<Product> products = context.Products;

    List<Product> query =
        (from product in products
         orderby product.Name
         select product).ToList();

    Console.WriteLine("The product list, ordered by product name:");
    foreach (Product product in query)
    {
        Console.WriteLine(product.Name.ToLower(CultureInfo.InvariantCulture));
    }
}
```

LazyLoading vs EagerLoading

- Opóźnione ładowanie (**Lazy Loading**) jest mechanizmem umożliwiającym pobieranie pewnych danych dopiero w momencie, kiedy są potrzebne. Ważne, że pobieranie następuje w sposób niewidoczny dla programisty. W EF wszelkie kolekcje oraz referencje na inne encje domyślnie podlegają opóźnionemu ładowaniu. LazyLoading bywa zwykle oczekiwane/pożądane ponieważ rzadko chcemy uzyskać natychmiastowy dostęp do wszystkich referencji w danej encji.
- Zachłanne ładowanie (**eager loading**) w jednym zapytaniu do serwera bazodanowego zwróci zarówno encję, jak i kolekcje z nią związane. W Entity Framework w celu wykonania zachłannego ładowania należy skorzystać z metody Include podczas wykonywania zapytania, np.:

```
using (var ctx = new NorthwindEntities())
    {
      customer = ctx.Customer.Include(c => c.Order)
                .Where(c => c.CompanyName == "CocaCola").FirstOrDefault<Customer>();

      // możemy również nazwę encji przekazać jako string
      // customer = ctx.Customer.Include("Order")
                .Where(c => c.CompanyName == "CocaCola").FirstOrDefault<Customer>();
    }
```

Pytania kompilowane:

- Jeżeli wiele razy wywołujemy zapytania podobne co do struktury, często możemy zwiększyć wydajność poprzez kompilację takiego zapytania i później jego wywoływanie z różnymi parametrami.
- Np. często pobieramy klientów z danego miasta przy czym miasto definiowane jest w RunTime'ie przez użytkownika w odpowiednim formularzu
- Klasa CompiledQuerydostarcza możliwości kompilowania i cachowania zapytań
- Zawiera metodę Compile (w kilku wersjach)
- Zapytanie jest kompilowane raz –podczas pierwszego wykonania. Po skompilowaniu można korzystać z zapytania dostarczając różne wartości parametrów (typy prymitywne) –metoda Invoke()
- Oczywiście niemożliwa jest rekonstrukcja skompilowanego zapytania (taka która zmieniałaby wygenerowany SQL)
- W nowych EF –autokompilacja zapytań włączona domyślnie

Przykład kompilacji i wykorzystania skompilowanego zapytania przyjmującego parametr typu Decimali zwracającego sekwencje zamówień o wartości równej $200.00:

```
static readonly Func<AdventureWorksEntities, Decimal, IQueryable<SalesOrderHeader>> s_compiledQuery2 =
    CompiledQuery.Compile<AdventureWorksEntities, Decimal, IQueryable<SalesOrderHeader>>(
            (ctx, total) => from order in ctx.SalesOrderHeaders
                            where order.TotalDue >= total
                            select order);

static void CompiledQuery2()
{
    using (AdventureWorksEntities context = new AdventureWorksEntities())
    {
        Decimal totalDue = 200.00M;

        IQueryable<SalesOrderHeader> orders = s_compiledQuery2.Invoke(context, totalDue);

        foreach (SalesOrderHeader order in orders)
        {
            Console.WriteLine("ID: {0}  Order date: {1} Total due: {2}",
                order.SalesOrderID,
                order.OrderDate,
                order.TotalDue);
        }
    }
}
```

Pytania z kartkówek:
Co to, do czego służy + przykład:

- SelectMany (operator/funkcja) -
  sygnatura:
  IEnumerable<TResult> SelectMany<TSource, TCollection, TResult>(
  this IEnumerable<TSource> source,
  Func<TSource, IEnumerable<TCollection>> collectionSelector,
  Func<TSource, TCollection, TResult> resultSelector)

  Przykład:
  products.SelectMany
  (p => p.Categories,
  (p, c) => p.Name + " has category " + c.Name)

  Given a sequence of elements (called *source*) of type TSource, it asks every such element (using *collectionSelector*) for a sequence of – in some way related – elements of type TCollection. Next, it combines the currently selected TSource element with all of the TCollection elements in the returned sequence and feed it in to *resultSelector* to produce a TResult that's returned.
- EagerLoadnig -

- GroupJoin - jest połączeniem Join z GroupBy. Dla encji:

```
class CustomerInfo
{
    public int Id { get; set; }
    public string Name { get; set; }
}

class Order
{
    public int IdCustomer { get; set; }
    public string Name { get; set; }
}
```

możemy mieć następujące przykładowe dane:

```
var customers = new CustomerInfo[3];
customers[0] = new CustomerInfo() { Id = 1, Name = "Piotr" };
customers[1] = new CustomerInfo() { Id = 2, Name = "Pawel" };
customers[2] = new CustomerInfo() { Id = 3, Name = "tertert" };

var orderList = new Order[5];
orderList[0] = new Order() { IdCustomer = 1, Name = "Zamowienie 1" };
orderList[1] = new Order() { IdCustomer = 1, Name = "Zamowienie 1 a" };
orderList[2] = new Order() { IdCustomer = 2, Name = "Zamowienie 2 a" };
orderList[3] = new Order() { IdCustomer = 2, Name = "Zamowienie 2 b" };
orderList[4] = new Order() { IdCustomer = 3, Name = "Zamowienie 3" };
```

I zapytanie:

```
var customerOrders = customers.GroupJoin(orderList, x => x.Id, x => x.IdCustomer, (cust
{
    CustomerName = customer.Name,
    Orders = orders.ToArray()
});

foreach (var customerOrder in customerOrders)
{
    Console.WriteLine(customerOrder.CustomerName);

    foreach (var order in customerOrder.Orders)
    {
        Console.WriteLine("\t{0}", order.Name);
    }
}
```

(tam, gdzie jest ucięte:

```
.GroupJoin(orderList, x => x.Id, x => x.IdCustomer, (customer, orders) => new
```

)

W efekcie otrzymamy listę klientów wraz z zamówieniami

- Immediate Query Execution -

Kartkówka wtorek 11:15 - opisz działanie, podaj składnię i napisz przykład:

- where contains

Działanie oczywiste, sprawdzamy czy dana kolekcja zawiera wartość któregoś pola naszej encji, tak jak to jest na przykładzie.
Składnia: Jak widać

- **Where…Contains** do pobrania produktów o wartościach *ProductModelID* zgodnych z podanymi w tablicy

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{
    int?[] productModelIds = {19, 26, 118};
    var products = from p in AWEntities.Products
                   where productModelIds.Contains(p.ProductModelID)
                   select p;
    foreach (var product in products)
    {
        Console.WriteLine("{0}: {1}", product.ProductModelID, product.ProductID);
    }
}
```

- Uwaga:Jako część klauzuli **Where…Contains można użyć typów** Array, List lub dowolnej kolekcji implementującej interface IEnumerable

z kolekcjami inline'owanymi (tworzone już w zapytaniu)

```
using (AdventureWorksEntities AWEntities = new AdventureWorksEntities())
{
    var products = from p in AWEntities.Products
                   where (new int?[] { 19, 26, 18 }).Contains(p.ProductModelID) ||
                         (new string[] { "L", "XL" }).Contains(p.Size)
                   select p;
    foreach (var product in products)
    {
        Console.WriteLine("{0}: {1}, {2}", product.ProductID,
                                           product.ProductModelID,
                                           product.Size);
    }
}
```

up

- partition

Z grubsza chodzi o metody take i skip wywołujemy je na zmiennej która reprezentuje całe zapytanie var query = (...).take(jakiś int)
Take pobiera pierwsze n wartości(n - podane jako argument)
Skip omija pierwsze n wartości(n - podane jako argument) i pobiera resztę
Zgodnie z **https://msdn.microsoft.com/pl-pl/library/mt693063.aspx** ale nie ma tego na wykładzie jest jeszcze TakeWhile i SkipWhile, gdzie podajemy warunek

skipowania i take'owania jako argument i tak długo jak jest spełniony tak długo wykonujemy daną czynność.

Laboratorium 3 - MongoDB

Co to write concern? Wymienić typy write concern w MongoDB i opisać różnice.
określa gwarancje dot. raportowania sukcesu na operacjach zapisu
typy:

- unacknowledged, (driver zapisuje do mongod i dostaje odpowiedz)
- journaled, (driver zapisuje do mongod, ten zapisuje w dzienniku i wysyła odpowiedz)
- replica acknowledged (driver zapisuje

{ w: <value>, j: <boolean>, wtimeout: <number> }
Mniej więcej opisuje to poziom "zatwierdzenia" dla operacji pisania do pojedynczej bazy/repliki/klastra. J dotyczy journal-logów. W czasie pisania do bazy są tworzone i zapisywane na dysku. Kiedy baza się mocno posypie to jak wstanie wykorzysta je, żeby wrócić do stanu używalności.
w:1 - żądamy potwierdzenia, że nasze pisanie się udało do pojedynczej bazy albo podstawowej w replice (default)
w:0 - sytuacja odwrotna (może to propagować błędy połączenia w apce), jeśli j:true to jakby, to gwarantuje nam, że i tak będziemy mieć to potwierdzenie
river zapisuje do mongod)
acknowledged, (dw:2+ - żądamy potwierdzenia od 2+ elementów repliki (w tym podstawowego)
w:"majority" - to samo, ale oczekujemy potwierdzenia od "większości" elementów (cokolwiek to znaczy)
wtimeout - raczej nie ważne, odsyłam do doc'a jak kogoś interesuje
Na slajdach są spoko screeny nawet, które to obrazują.

Grid FS - co to, jak działa?
Specyfikacja do zarządzania plikami przekraczającymi 16MB. To jest limit BSON'owego dokumentu. Każdy dokument w kolekcji, może mieć max 16MB. Zamiast trzymać plik w jednym dokumencie, GridFS dzieli go na części (chunks) po 255KB każda (oprócz ostatniej, ta bierze ile potrzebuje). Używa dwóch kolekcji, jedna do przechowywania "chunks", druga na metadane pliku. W razie potrzeby driver dokonuje ponownej re-asemblacji potrzebnych części. Daje to np. możliwość dostania się do dowolnej części pliku typu video/audio.
Innym przykładem może być dzielenie takich plików, które może wcale nie mają 16MB+, ale nie chcemy ich całych ładować do pamięci (RAM jest ważny).

Indeksy - co to, jakie typy są w mongo, jak zaimplementowane?
indeksy to struktury danych przyspieszające operację wyszukiwanie
zdefiniowanie na poziomie kolekcji, korzystają z B-drzew (z wyjątkiem tych zdefiniowanych na dane geograficzne)
single field – indeks na pojedyncze pole dokumentu (także pole zagnieżdżonego dokumentu)
compound – indeks na więcej niż jedno pole dokumentu
multikey – indeks na pole tablicowe (dodaje do indeksu każdą wartość w tablicy)
geospatial – indeks obsługujący zapytania przestrzenne
text – obsługuje zapytania na tekstach w dokumencie (language specific – używa stop words, stemming wyrazów)
hashed – zarządza elementami z hashami wartości (wykorzystywane w klastrach)
komenda: ensureIndex(fields, options)

Jakie są rodzaje skalowalności (krótki opis)? Którą z nich trudniej osiągnąć?
wertykalna - zwiększenie zasobów na danym węźle np. przez zwiększenie pamięci
horyzontalna - dodawanie kolejnych węzłów do systemu (system rozproszony)

Metody agregacji (wymienić + składnia)
Aggregation Pipeline
Map-Reduce
Single Purpose Aggregation Operations
https://docs.mongodb.com/manual/aggregation/

Na czym polega Replikacja w MongoDB
replikacja - proces synchronizacji danych na wielu serwerach
zbiór replik jest grupą instancji uruchomionych demonów mongod, które hostują ten sam zbiór danych, dzielą się on na primary (jeden) i secondaries, w razie awarii primary wybierany jest nowy primary przez głosowanie. Awaria jest wykrywana za pomocą komunikatu heartbeat, którego nasłuchują wszystkie repliki. Istnieje możliwość zdefiniowania tzw. arbitra, którego jedynym zadaniem jest niedopuszczenie do remisów przy elekcji nowego primary (dodaje się gdy, mamy parzystą ilość instancji w replice).

NoSQL - co to, scharakteryzować
Not Only SQL - nierelacyjna baza danych, zwykle brak schematu, języka zapytań;
Mniej restrykcyjnie przestrzegane właściwości ACID; łatwo skalowalne;
Wady: często redundancja danych, podatność na niespójności; nowe i niedopracowane narzędzia; zwykle brak standardu do zapytań; problemy przy modelowaniu złożonych struktur

Rodzaje baz NoSQL

- bazy klucz-wartość (ang. key-value)
- bazy kolumnowe (ang. column oriented stores)
- bazy dokumentowe
- bazy oparte na grafach (ang. graph stores)

Co to model dokumentowy?

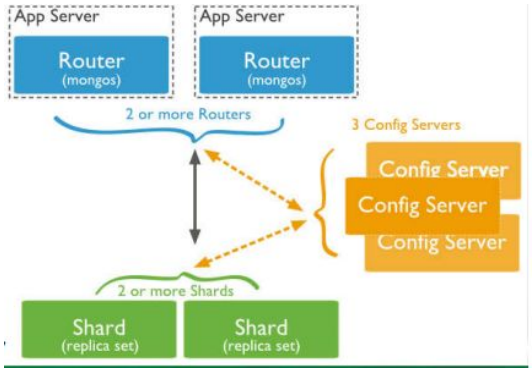
baza danych - zawiera zestaw kolekcji
kolekcja - składa się z dokumentów
dokument - jest zbiorem par klucz-wartość

Różnice między modelem relacyjnym, a dokumentowym

https://sqldev.files.wordpress.com/2011/08/mongo.png
Można dodać, że Mongo ("z tyłu") reprezentuje JSON'owe dokumenty jako BSON (binary-encoded) czyli dokument rozszerzający go, o niektóre typy, ułatwiający kodowanie/dekodowanie przy użyciu różnych języków.
TL:DR user używa JSON'a, a z tyłu wszystko jest w szybszym, lekkim (binarnym) BSON'ie .

Sharding w MongoDB, definicja + opisać komponenty

Sharding - to podział przechowywanych danych na wiele maszyn (skalowanie horyzontalne, odpowiednik w relacyjnych - partycjonowanie)
komponenty:
shards - przechowują dane (często zbiory replik)
query routers (mongo instances) - interfejs dla aplikacji klienta, służą kierowaniu zapytań do właściwego sharda i zwracaniu wyniku klientowi (klient wysyła zapytanie do jednego query routera, ale zwykle te routery są zwielokrotnione)
config servers - przechowują metadane dotyczące klastra, wykorzystywane przez query routers do odnalezienia odpowiedniego sharda.



Sposoby mapowania klas Javy do kolekcji w Mongo

- dziedziczenie klasy po BasicDBObject
- implementacja przez klasę interfejsu DBObject
- Spring Data MongoDB
- Hibernate OGM

Transakcje w MongoDB (Są? Nie ma? Jak?)
Transakcje w Mongo nie są obsługiwane. W razie potrzeby twórcy sugerują ~~użycie~~ implementacje pseudotransakcji. jak? tak:
1. stworzenie obiektu transakcji w stanie initial i pola pendingTransactions we właściwych obiektach
2. zmiana stanu obiektu transakcji na pending i wykonanie operacji skojarzonych z transakcją
3. zmiana stanu transakcji na committed
4. usunięcie informacji o pendingTransaction we właściwych obiektach
5. zmiana stanu transakcji na done

___
Slajd23 - wyjątek, kiedy nie chcemy mieć widocznego pola “_id”, możemy zrobić coś typu {_id:0, something:1}

Laboratorium 4 - IBM Netezza

1. Wymień i krótko opisz podstawowe elementy architektury IBM Netezza

Before we discuss the data distribution mechanism, let us first understand how Netezza stores the data on disk. Each Snippet Processor in the Snippet Processing Unit (SPU) has a dedicated hard drive and the data on this drive is called a data slice. Each disk is divided into three partitions: Primary (user data), Mirror, and Temp (intermediate processing data). All the user data and temp space from each primary partition is copied to the mirror partition in another disk, which is called replication. Tables are split across SPUs and data slices and the data is stored in groups according to rows, while data is compressed according to identical column values (columnar compression).

Każdy Snippet Procesor ma własny dysk. Dane na tym dysku tworzą data slice. Tabele są rozproszone na poszczególne data slice (wg pewnego algorytmu hashującego). Dane są kompresowane (zajmują ~4 razy mniej miejsca).

Dodatkowo dane na dyskach są mirrorowane, jest również przestrzeń na dane tymczasowe.

SPU składa się z własnego procesora, pamięci i FPGA (tu zachodzi większość przetwarzania, np dekompresja i wybór kolumn/wierszy).

2. Jak realizowana jest operacja join w IBM Netezza? Od czego zależy efektywność joinów?

http://stackoverflow.com/questions/28458189/join-types-of-netezza (?)

Collocated Joins – slajd 41
+
https://www.ibm.com/support/knowledgecenter/SSULQD_7.2.0/com.ibm.nz.adm.doc/c_sysadm_dist_keys_collo_joins.html

+ praktycznie to samo co na slajdach, ale napisane w jednym miejscu i bardzo w skrócie :)
http://www.folkstalk.com/2010/11/collocated-join-in-netezza.html

3. Różnice między data skew a processing skew
Obrazki na slajdach 38-39

Data skew / processing skew

data/storage skew

- Jest to nierównomierny rozkład wierszy tabeli między Data Slice'ami.
- Spowodowany nie-do-końca losowym podziałem.
- Może się okazać, że niektóre DataSlice mają znacznie więcej danych niż pozostałe.
- Powoduje to, że czas wykonania rośnie...

processing skew

- Gdy zapytanie potrzebuje wykonać obliczenia na wiele (różnych) sposobów.
- Pośrednie tabele używane w obliczeniach mogą powodować nierównomierny podział, gdy niektóre znacznie częściej będą używane od innych.

~~What is storage skew? When data is loaded to a table, the rows are distributed to a particular dataslice based on the defined distribution key, assuming distribution is not defined as random. The dataslices holding a larger count of table rows will generally work longer, harder, and consume more resources to complete its job. These dataslices can become a performance bottleneck for the queries. This uneven distribution of table rows is known as storage skew.~~

~~What is processing skew? A query will need to process table data in various ways to produce the final result set. Along the way, intermediate tables will be used as a way to filter, project, sort, and aggregate data, along with broadcasting and redistributing data. These intermediate tables may produce an uneven distribution, which can negatively impact the benefits of parallelism. Because this happens as a result of how the data is processed within the query, this is known as processing skew~~ **(to samo wyżej w skrócie jest opisane)**

4. Zone Map
Zone maps are automatically generated internal tables that the IBM® Netezza® system uses to improve the throughput and response time of SQL queries against large grouped or nearly ordered date, timestamp, byteint, smallint, integer, and bigint data types.

Zone maps reduce disk scan operations that are required to retrieve data by eliminating records outside the start and end range of a WHERE clause on restricted scan queries. The IBM Netezza Storage

Manager uses zone maps to skip portions of tables that do not contain rows of interest and thus reduces the number of disk pages and extents to scan and the search time, disk contention, and disk I/O.

slajd 52 -53

Opracowanie wykładu:

Systemy OLTP vs DSS:
OnLine Transaction Processing - dobre rozwiązanie do zarządzaniem i zmianą danych
Decision Support System - dobre rozwiązanie do zapytań (ale bez modyfikacji danych

IBM Netezza:

- silnik analityczny do konkretnych celów
- zintegrowana baza danych, serwer, magazyn danych
- standardowy interfejs (?)
- niski koszt posiadania

Reklamowane cechy:

- 10-100x szybsze od tradycyjnych systemów
- prostota: minimalna potrzeba administracji
- skalowalność: dane mogą iść w petabajty
- wysoka wydajność dla zaawansowanej analizy

Tradycyjna DB vs Netezza:
tradycyjna db zbudowana z jednego modułu: (CPU, pamięć, dane)
Netezza zbudowana z wielu modułów: (SPU, pamięć, dane)

S-Blade i SPU to synonimy
SPA - snippet processing array (tablica SPU)
SPU - snippet processing unit

Schemat komunikacji:
użytkownik wysyła SQL do hosta i otrzymuje od niego odpowiedź.
host z kolei tworzy odpowiedz korzystając z wielu SPU.

Netezza architektura:
urządzenie Netezzy:
1 “lekki” host (z Red Hat linuxem) połączony z wieloma SPU (zbudowane z Field Programmable Gate Array, CPU, pamięci). każde SPU połączone z “disk enclosure”, które składa się z dysków, na których przechowywane są dane.
Aplikacje (połączone z hostem): Advanced analytics, BI, ETL, Loaders.

Przetwarzanie danych:
na przykładzie zapytania:
select distinct productgrp, sum(nrx)

from mthly_rx
where month = ‘1234’
and market = ‘123’
and specialty = ‘gastro’

1. na dysku siedzi sobie skompresowany kawałek (slice) tabeli mtly_rx
2. FPGA go dekompresuje
3. FPGA dokonuje *projekcji* - wybiera dane spełniające klauzule **select**
4. FPGA dokonuje *selekcji* - filtruje dane według klauzuli **where**
5. CPU odpowiada za sumę, sortowanie, joiny, funkcje agregacje - w tym przypadku obliczy **sum(nrx)**

Budowa S-Blade’a a.k.a. SPU:

- 2x intel Quad-Core 2+GHz CPU,
- 16GB DRAM
- 4x Dual-Core FPGA 125MHz
- SAS expander module

Modele Netezzy (x to zmienna)

- nazwa: TFx
- ilość procesorów snippet: 8 * x
- pojemność (TB): podłoga(2,67 * x)
- efektywna pojemność: 4 * pojemność (w Netezzie skompresowane dane zajmują około 4 razy mniej miejsca od oryginalnych danych)

x = {3, 6, 12, 24, 48, 120} [te, które pojawiły się na wykładzie]

Przechowywanie danych:

- każdy snippet procesor ma własny dysk
- na tym dysku są tworzone data slice
- tabele są dzielone na poszczególne data slice
- dane są skompresowane

Dystrybucja danych:

- dane są dzielone do sliceów przez algorytm hashujący
- na dyskach poza zwykłą partycją, na której jest data slice jest też partycja mirror

Mirroring:

- mamy sobie parę dysków nazwijmy je A i B (lub wiecej takich par)
- oba dyski mają partycje: primary. mirror i temp.
- dysk A w mirror ma kopie partycji primary dysku B
- dysk B w mirror ma kopie partycji primary dysku A

Narzędzia:

- nzsql
- NzAdmin
- Aginity
- DBeaver (połączenie przez ODBC/JDBC)

Składnia (dużymi literami syntax SQLowy, małymi nazwy własne, [] - opcjonalne):

1. Zarządzanie użytkownikami (nzsql lub NzAdmin):
    a. tworzenie:
        - CREATE USER nazwa WITH [options];
        - np. CREATE USER jane WITH PASSWORD ‘haslo’;
    b. edycja:
        - ALTER USER nazwa WITH [options];
        - np. ALTER USER jane WITH PASSWORD ‘hasl0’;
    c. usuwanie:
        - DROP USER nazwa;
        - np. DROP USER jane;
2. Grupy (nzsql lub NzAdmin):
    a. tworzenie:
        - CREATE GROUP nazwa WITH [options]
        - np. CREATE GROUP marketing WITH USER jane, joe;
    b. edycja:
        - ALTER GROUP nazwa [ADD|OWNER|RENAME|WITH]
        - np. ALTER GROUP marketing ADD USER john;
    c. usuwanie:
        - DROP GROUP nazwa;
        - np. DROP GROUP marketing;
3. Uprawnienia:
    a. przyznawanie:
        - np. SYSTEM(ADMIN) => GRANT ALL ON TABLE TO jane;
        - np. RETAIL(ADMIN) => GRANT LIST, SELECT ON TABLE TO PUBLIC
4. Bazy danych (nzsql lub NzAdmin, trzeba być ADMINem lub mieć uprawnienia na obiekcie):
    a. tworzenie:
        - CREATE DATABASE nazwa;
        - np. CREATE DATABASE labdb;
    b. edycja:
        - ALTER DATABASE nazwa [RENAME|OWNER] TO nazwa;
        - np. ALTER DATABASE retail NAME TO retaildb;
        - np. ALTER DATABASE retaildb OWNER TO joe;
    c. usuwanie:
        - DROP DATABASE nazwa;
        - np. DROP DATABASE retaildb;
5. Tabele (nzsql lub NzAdmin, trzeba być ADMINem lub mieć uprawnienia na obiekcie):
    a. tworzenie:
        - CREATE TABLE nazwa (nazwa_kolumny typ_danych, …);
        - warto robić używając pliku DDL ($ nzsql -d nazwa_bazy -f plik.ddl)
        - tabele tymczasowe: CREATE [TEMPORARY|TEMP] TABLE …
        - tabela może mieć maksymalnie 1600 kolumn, a wiersz rozmiar 65535
    b. edycja: ALTER TABLE nazwa akcja [ORGANIZE ON {(<columns>) | NONE}]

c. usuwanie:
    - DROP TABLE nazwa;
    - np. DROP TABLE retaildb;
d. truncate (aby usunąć wszystkie wiersze)
    - TRUNCATE TABLE nazwa;
    - ścinać można też bazę

6. CTAS (Create Table AS - to zwykła tabela tworzona jako wynik zapytania):
    - CREATE TABLE nazwa AS <zapytanie>
    - często używane aby zmienić metodę dystrybucji tabeli bez przeładowywania danych: CREATE TABLE nazwa AS <zapytanie> DISTRIBUTE ON nazwa_kolumny
    - można tworzyć tymczasowe tabele CTAS

Typy danych:

Data Type	Values
BYTEINT	8-bit values in range –128 to 127
SMALLINT	16-bit values in range –32,768 to 32,767
INTEGER	32-bit values in range –35,791,394 to 35,791,394
BIGINT	64-bit values in range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
NUMERIC(p,s)/DECIMAL	Numeric with precision p and scale s. Precision can range from 1 to 38, scale from 0 to the precision
FLOAT(p)/REAL	Floating point number with precision p, from 1 to 15.
CHAR(n)	Fixed length string, maximum 64,000
VARCHAR(n)	Variable length string, maximum 64,000
BOOLEAN	With value true (T) or false (F)
TIMESTAMP (DATE/TIME)	Ranging from January 1, 0001 00:00:00.000000 to December 31, 9999 23:59:59.999999.

Dystrybucja danych:

- Tworząc tabele możemy wybrać jak dane z tabeli mają być dystrybuowane na slice'y.
- Służy do tego DISTRIBUTE ON metoda_dystrybucjji przy tworzeniu tabeli.
- 2 sposoby:
    - DISTRIBUTE ON RANDOM - losowa dystrybucja
    - DISTRIBUTE ON nazwa_kolumny/kolumn - dystrybucja oparta o hashowanie
- celem jest rozdystrubowanie danych tak, żeby w jak najlepszym stopniu wspierały wielowątkowość (no bo wydajność)

Data skew - skośność danych

- przykład: dane dystrybuujemy na podstawie kolumny płeć - wtedy 2 jednostki będą mieć wszystkie dane przez co narzut pracy u nich jest duży, a pozostałe jednostki się marnują

Processing skew - skośność przetwarzania

- przykład tym razem mamy 12 sliceów i dane podzielone przez miesiące na pozór jest spoko, ale okazuje się, że jeden z nich ma dużo większy zbiór danych od reszty przez co znowu czasy odpowiedzi na zapytania nie są równe (dla tego miesiąca trzeba przetworzyć więcej danych).

Dystrybucja danych - Collocated joins:

- mamy 2 tabele, które wiemy, że będą często joinowane - należy je zdystrybuować (podzielić na slice'y) według tego samego pola (PK w jednej, FK w drugiej)
- kolokacja oznacza, że odpowiadające dane z joinowanych tabel zostaną rozdystrybuowane do tych samych slice'ów (dzięki czemu każdy SPU będzie mógł zjoinować dane, które posiada)
- jest to bardzo ważne pod kątem wydajności joinów w wielowątkowym systemie jakim jest Netezza

Redystrybucja tabel:

- czasami nie da się rozdystrybuować obu tabel w oparciu o klucz relacji w takim przypadku Netezza zredystrybuuje potrzebne kolumny do innych slice'ów
- (single redistribute) jeśli jedna tabela jest rozdystrybuowana w oparciu o klucz joina (join key) druga będzie rozdystrybuowana (przesuwamy elementy tylko jednej z tabel)
- jeśli żadna tabela joina nie jest rozdystrybuowana w oparciu o klucz joina podwójna redystrybucja jest konieczna
- (double redistribute) obie tabele są dystrybuowane w oparciu o klucz joina (przesuwamy między slice'ami elementy obu tabel, a nie tylko jednej)

Broadcast:

- czasami (np. gdy tabela wchodząca w skład joina jest bardzo mała) optymalizator wybierze broadcastowanie tabeli zamiast redystrybucji
- najpierw wszystkie SPU wysyłają swoje dane z tej tabeli do hosta, a następnie po skonsolidowaniu tabeli w hoście jest ona rozsyłana do SPU, które zawierają tabele, z którą joinujemy (o to żeby nie trzeba było redystrybuować dużej tabeli - lepiej mieć kopie małej w paru miejscach)

Można robić SELECT datasliceid - pokaże nam to, w którym slice'ie jest dany wiersz.

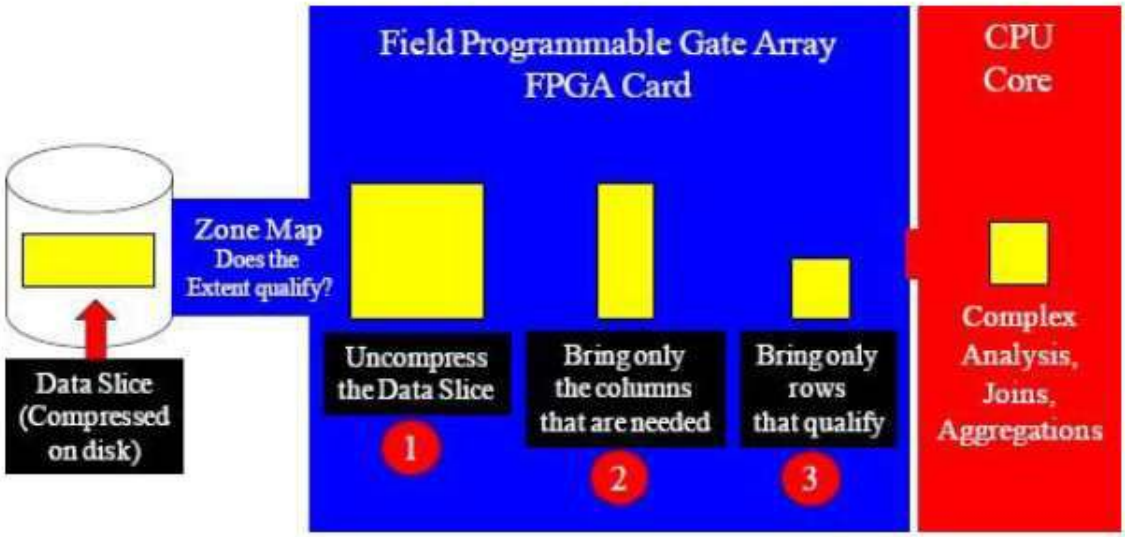
Optymalizator - plany (czyli opis jak będzie wykonywane zapytanie krok po kroku):

- explain - dostarcza szczegółowe wyjaśnienie planu wykonania zapytania zawierające koszty, szacowania itp.
- EXPLAIN [VERBOSE] <query>;
- wersja w języku naturalnym:
- EXPLAIN PLANTEXT <query>;
- wersja grafowa:
- EXPLAIN PLANGRAPH <query>;

Zone maps:

- przechowują informacje o danych w slice'ach - pozwalają na szybsze działanie.
- odpowiednik indeksów
- zapamiętuje wartości minimalne i maksymalne zadanych pól
- generowane automatycznie dla pól typu: date, timestamp, numerycznego nie większego niż 8 bajtów
- nz_zonemap - pozwala sprawdzić, które kolumny da się zmapować

Przetwarzanie danych (w sumie było już wcześniej):



Clustered base tables (CBT):

- tabela zawierająca dane zorganizowane według od 1 do 4 kluczy
- klucz organizujący to kolumna, która została wybrana do klastrowania rekordów (?)
- oparte o krzywe Hilberta
- mapuje wielowymiarowe dane do jednowymiarowych
- CREATE TABLE … ORGANIZE ON (kolumna, kolumna, …)
- klucze organizujące można zmieniać przy użyciu ALTER
- np. ALTER TABLE … ORGANIZE ON NONE (none transformuje tabele do nieCBTowej)

Materializowane widoki:

- tworzymy: CREATE MATERIALIZED VIEW nazwa AS SELECT … FROM … [ORDER BY ...]
- jeśli nie podamy orderby to widok odziedziczy kolejność wstawiania danych z oryginalnej tabeli
- materializowany widok będzie miał taką samą dystrybucję jak tabela, na której bazuje

Inne zagadnienia:

- zarządzanie serwerem (Nzadmin, Aginity)
- ładowanie danych (Nzload)
- reorganizacja tabel i zwalnianie przestrzeni dyskowej (GROOM)
- tranzakcje

- inza

Laboratorium 5 - Bazy grafowe

1. **Graf, wierzchołek, krawędź**
   Graf - struktura matematyczna służąca do przedstawiania oraz badania relacji między obiektami. Zbiór wierzchołków połączonych krawędziami.
   Wierzchołek - w teorii grafów punkt pewnej przestrzeni(zbioru) nad którą zbudowany jest graf.
   Krawędź - para wyróżnionych wierzchołków grafu, które są ze sobą połączone.

2. **Ścieżka w grafie**
   Jest to ciąg wierzchołków, taki że dla każdej kolejnej pary istnieje łącząca je krawędź.
   Innymi słowy jest to ciąg krawędzi, łączących kolejne wierzchołki z ciągu wierzchołków ścieżki.

3. **Co ma schemat w bazie relacyjnej do grafu**
   Schemat w bazie relacyjnej jest grafem.

4. **Czy relacyjna baza danych jest dobrym silnikiem do przechowywania grafu? Dlaczego?**
   Nie, ponieważ:
   - słabo się skaluje
   - są trudne w użyciu (mnóstwo joinów jeśli wiele relacji)
   - uniemożliwia stosowanie optymalizacji charakterystycznych dla grafu

5. **Grafowa baza danych**
   “Baza danych wykorzystująca struktury grafów z węzłami, krawędziami i własnościami do przedstawiania i przechowywania danych oraz do obsługi zapytań semantycznych. Bazą taką jest każdy system pamięci masowej, który zapewnia bezindeksowe sąsiedztwo, co oznacza że każdy element bazy zawiera bezpośredni wskaźnik na sąsiadujące elementy i nie jest konieczne wyszukiwanie, indeksowe.”
   ▪ Grafowe bazy danych mają zastosowanie w modelach, których złożoność przekracza możliwości relacyjnej bazy danych
   ▪ Rozwiązania wydajne, skalowalne
   ▪ Dynamiczna struktura
   ▪ Zastosowanie w wielu sieciach społecznych

6. **Zalety grafowej bazy danych**
   - skalowalność
   - Big Data
   - dynamiczna struktura
   - generyczne typy węzłów i krawędzi
   - brak JOINów
   - “Graf bardziej naturalnie opisuje relacje”

7. **Porównanie modeli relacyjnej bazy danych z grafową bazą danych (co jest czym - odpowiedniki)**

- tabela - etykieta na węzłach
- krotka - węzeł
- kolumny - właściwości węzła
- klucze obce - relacje
- klucze główne - brak
- dane z domyślnymi wartościami - brak
- tablice złączenia - relacje (krawędzie), kolumny w tych tabelach- własności relacji

8. Neo4j - co to jest, główne cechy

Neo4j to grafowa baza danych.
Używanym w niej językiem zapytań jest Cypher.
Przechowuje dane w węzłach(node), które są połączone relacjami.
*Jest zaimplementowana w Javie.

9. Neo4j jako grafowa baza danych

Grafowa:
- graf z właściwościami dla dynamicznego schematu
- idealny dla złożonych, mocno połączonych danych
- zaimplementowane algorytmy grafowe

Baza danych:
- mocno skalowalna
- wysoka wydajność
- szybkie wędrowanie po schemacie
- serwer z REST API lub lokalna wersja embedded w JVM
- *Zapewnia BASE (Basic-Availability, Soft-State, Eventual-consistency) (**prawie** ACID)

10. Czym jest Cypher + najprostszy przykład

Cypher to deklaratywny język do opisywania wzorców w grafie, wykorzystujący ascii-art syntax.
Pozwala nam na zdefiniowanie tego co chcemy otrzymać, zaktualizować lub dodać do grafu bez podania kolejnych instrukcji.
Przykład:

(a) -[:LIKES]-> (b)

11. Węzeł w Cypher

Węzeł przechowuje pewne dane.

Do reprezentacji węzła używane są nawiasy okrągłe “()” które oplatając węzeł tworzą kółko: MATCH (node:Label) RETURN node.property

Możliwe jest przypisanie do zmiennych, aby później odwoływać się do danego węzła: MATCH (node1:Label1)-->(node2:Label2) WHERE node1.propertyA = {value} RETURN node2.propertyA, node2.propertyB

12. Relacje w Cypher

Relacja określa jak dwa węzły są ze sobą skojarzone.
Do reprezentacji relacji używana jest strzałka “-->” pomiędzy dwoma węzłami.
Możliwe jest podanie dodatkowych informacji w nawiasach kwadratowych:
▪ typ relacji -[:KNOWS|:LIKES]->

- nazwa zmiennej -[x:KNOWS]->
- dodatkowe informacje -[{since:2010}]->
- długość ścieżki -[:KNOWS*..4]->

13. Najkrótsza ścieżka w grafie - co to, jak zrealizować w Cypher

To najkrótszy ciąg wierzchołków(czy też krawędzi) łączący dwa wybrane wierzchołki.
path = shortestPath((node1) - [:RELACJA*1..7]-(node2))
np.: path = shortestPath((user)-[:KNOWS*..5]-(other))

14. Co to jest collaborative filtering i gdzie jest wykorzystywany

Jest to technika służąca do budowania personalizowanych rekomendacji w internecie - np. polecanie co jeszcze można dokupić do produktu.
Przykład: Znajdowanie użytkowników o podobnych gustach:
(user)-[:PURCHASED]->(product)<-[:PURCHASED]-()-[:PURCHASED]->(otherProduct)

15. Czym jest Gremlin (w Neo4J)

- Język zapytań - Graph Traversal Language
- DSL dla grafów