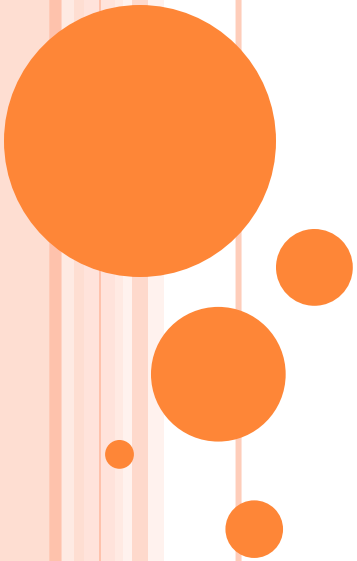


Лекція 6. Advanced datastore

Глибовець А.М.

Вступ

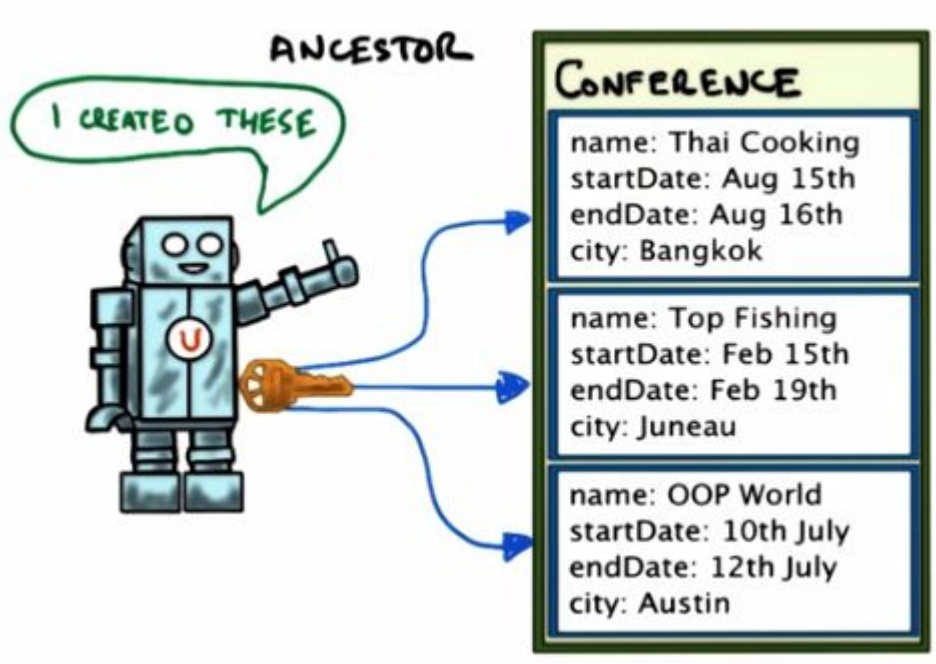
- Минулого разу ми познайомилися з основними операціями по роботі з Datastore
- Нам цього достатньо?
- Як ви думаєте, чого не вистачає?

Вступ

- Transactions
- Consistence
- Data relationships

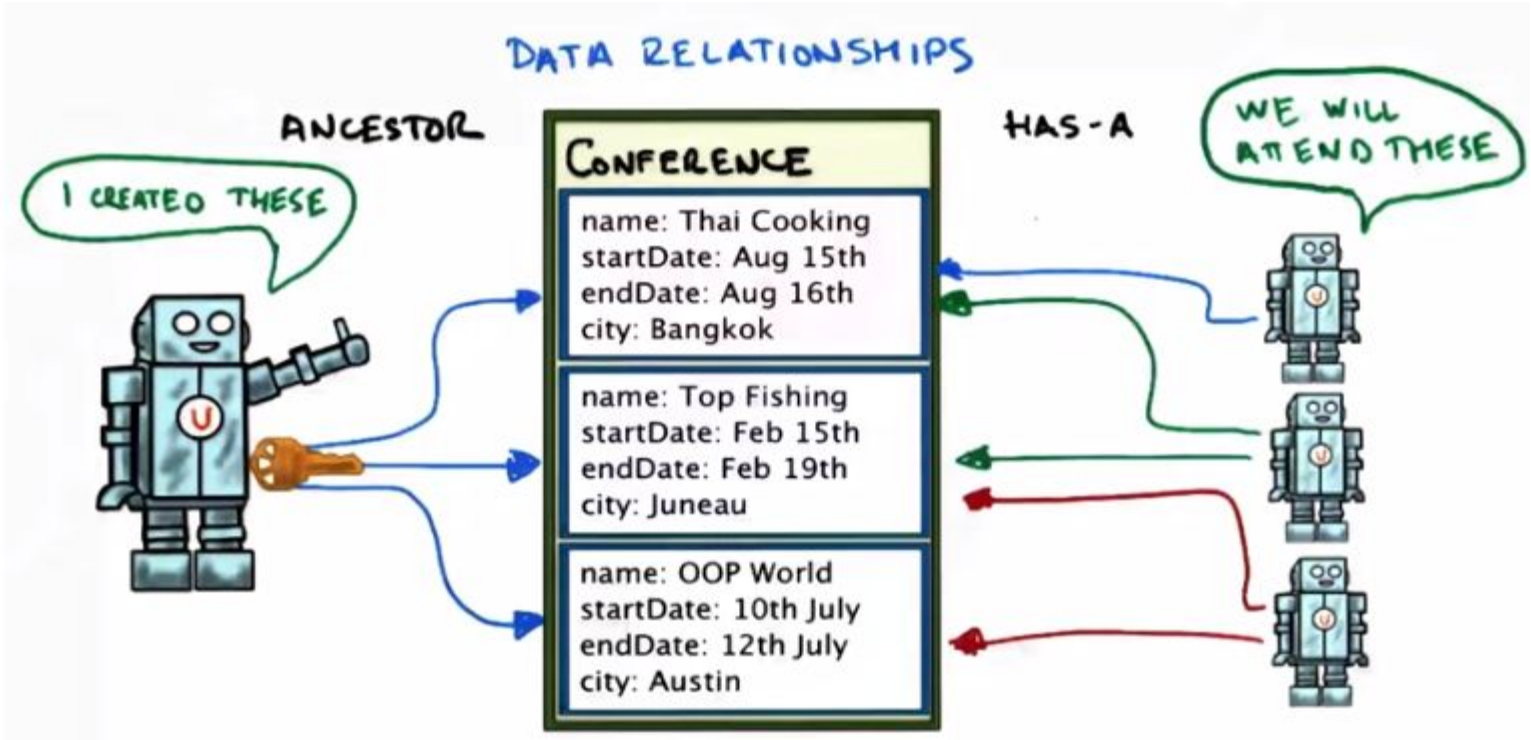
Data Relationships

- Ancestor
  - assigned at creation
  - can never be changed
    - в нашому застосуванні будуть конференції і користувачі, що створюють конференції
    - хто буде ancestor?
      - **USER**



Data Relationships

- HAS-A
  - в нас на конференцію будуть записуватися різні користувачі
  - таким чином в однієї конференції буде багато відвідувачів
  - тому вони будуть мати зв’язок HAS-A

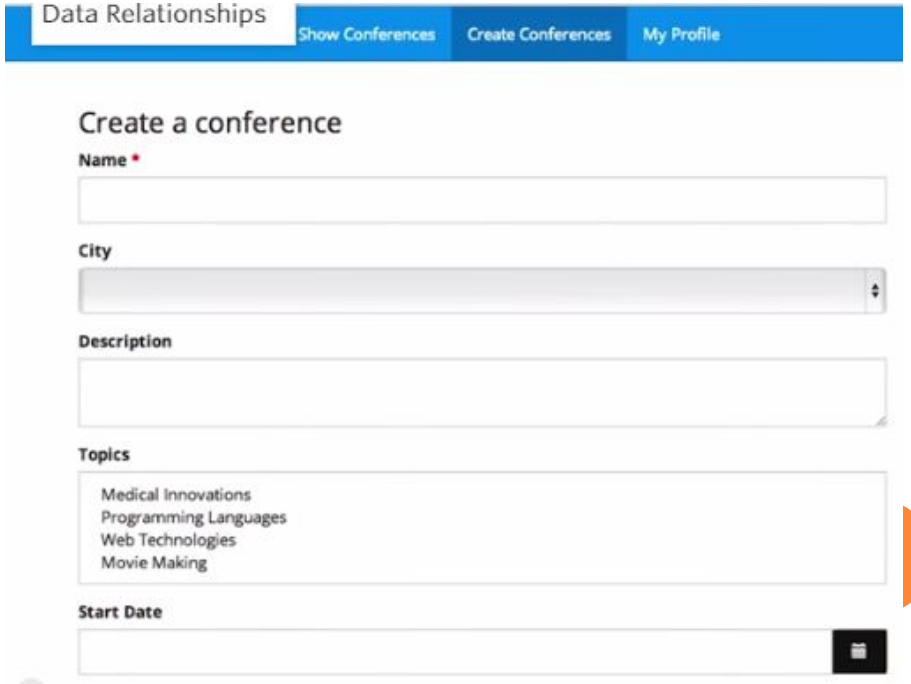


Конференції

- Зараз ми спробуємо розібратися з нашими конференціями
  - нам потрібно навчитися писати запити і використовувати фільтри
  - розібратися з індексами
  - розібратися з транзакціями

Створення конференції

- Перше з чого ми почнемо, це з створення конференції
- Десь так воно має виглядати в вас зараз
- Але зараз конференція не створиться, бо ми ще не реалізували API



Створення конференції

- Ми хочемо створити Entity конференція таким чином, що б кожна конференція мала свого Parent (користувача, що створює конференцію)
- Як ви думаєте навіщо це робити?
  - знайти всі конференції користувача
  - при видаленні користувача видалити всі його конференції

Створення конференції

- Давайте подивимося на два нові класи:
  - Conference
    - звернемо увагу на нові анотації та методи
  - ConferenceForm
    - дуже простий клас, що використовується для веб інтерфейсу
    - ми вже використовували схожий для Profile
  - ці класи в додатку до лекції
  - ви маєте покласти ці файли в вірні пакети

Анотації

- Які анотації нові ми побачили?
- Що значить @Index?
- Що значить @Parent?

OfyService

- Тепер нам необхідно внести зміни в OfyService
- Ми маємо зареєструвати Conference класс
  - factory().register(Conference.class);

ConferenceApi

- Тепер нам необхідно додати певні методи в ConferenceApi
- Ми додамо методи
  - getProfileFromUser
  - createConference
  - їхні заглушки також в додатковому коді, подивимося на них

ConferenceApi

- Допишемо код:
  - Спочатку отримаємо Key<Profile> користувача
    - Key<Profile> profileKey = Key.create(Profile.**class,userId);**
  - Після цього виділимо ключ для нашої конференції
  - Синтаксис має вигляд
    - Key<T> key = factory().allocatedId(Entity.Class);
      - factory() статичний метод в OfyService
  - Але в нашому випадку це не зовсім вірно так як наш клас Conference має батька Profile
  - в такому випадку синтаксис має вигляд:
    - Key<T> key = factory().allocatedId(parentKey,Entity.Class);

ConferenceApi

- Раніше ми вже навчилися зберігати сутності
- Але в нашому випадку необхідно одразу зберігати дві сутності Profile і Conference
- Тоді ми можемо скористатися методом entities()
  - ofy().save().entities(entity1,entity2,…).now();

ConferenceApi

- Тепер ми можемо забрати id для Conference маючи ключ
  - key. getId();
- Далі ми заберемо існуючий профайл або створимо новий для користувача з значеннями за замовчанням
- Створимо нову конференцію
- і в кінці збережемо разом конференцію і профайл

Типи запитів

- В нашому застосуванні мають бути наступні фільтри:
  - всі конференції
    - це запит типу Query by Kind
  - всі конференції створені користувачем
    - Query by Kind filtered by Ancestor
  - всі конференції на які користувач записався
    - Query by Kind filtered by Property
  - також в нас буде фільтр за
    - топіком
    - початком
    - кількістю відвідувачів

Запити

- Якщо ми знаємо ключ запит простий
  - Entity entity = ofy().load().key(key).now();
- Якщо нам потрібні всі Entity певного типу ми маємо зробити наступну річ:
- Спершу ми створюємо запит:
  - Query query = ofy().load().type(Kind.class);
  - Також ми може відсортувати результат за певною властивістю
  - Query query = ofy().load().type(Kind.class).order(“name”);
- Потім ми забираємо результат запиту:
  - List<Kind> results = query.list();

ConferenceApi

- Тепер ми маємо додати в ConferenceApi метод queryConferences
  - код є в додатках
  - давайте подивимося на нього
  - зверніть увагу на імпорт, ви маєте приєднати вірну бібліотеку для Query
  - пізніше ми виправимо даний метод
  - запустіть проект і протестуйте API, вам мають повернути всі конференції
  - також в вас має почати працювати вкладка Show Conferences в застосуванні

Ancestor Queries

- Як ви пам’ятаєте в кожної конференції є її батько (людина, що створила дану конференцію)
- Тепер ми спробуємо забрати всі конференції, що створила одна людина
- Запит має наступний вигляд:
  - Query query = ofy().load().type(Entity.class).ancestor(key);
  - key of the parent

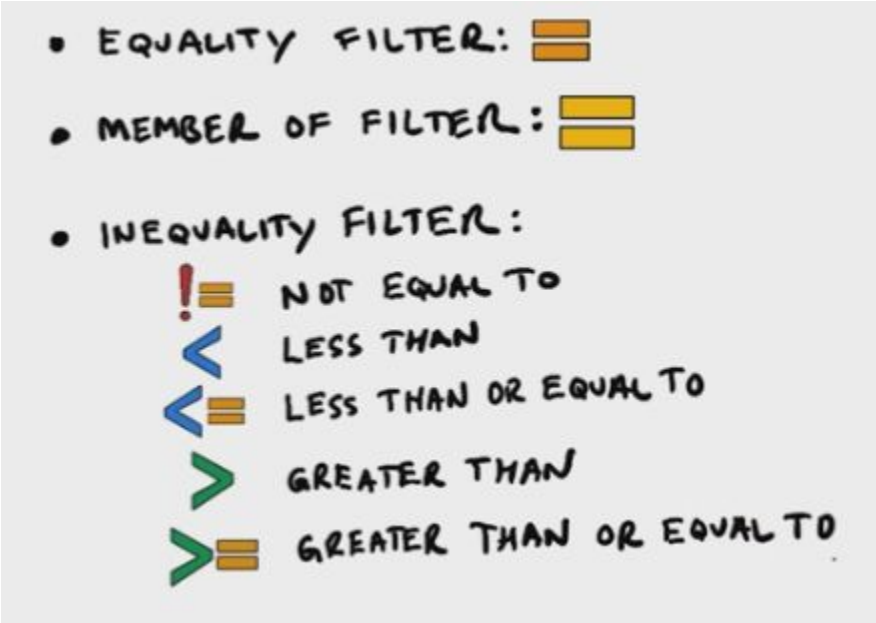
Ancestor Queries

- Ви маєте самостійно додати метод getConferencesCreated() в ConferenceApi
- даний метод має повертати всі конференції створені конкретним користувачем (користувачем, що залогінився)
  - метод має бути POST
  - user має бути залогінений

Filter by Property

- Тепер ми розглянемо самий цікавий запит, запит за параметрами

Filter by Property



Filter by Property

- Такі запити мають вигляд:
  - Query query = ofy().load().type(Kind.class).filter("property operator", "value");
  - Приклад:
  - Query query = ofy().load().type(Conference.class).filter(“city =", “London");
- Після того як Query об’єкт створений ви не можете його змінити …

Filter by Property

- Але ми можемо додавати фільтри до запиту.
- Ми можемо розбити створення Query
  - Query query = ofy().load().type(Conference.class);
  - query = query.filter(“city =", “London");
  - … і так багато фільтрів

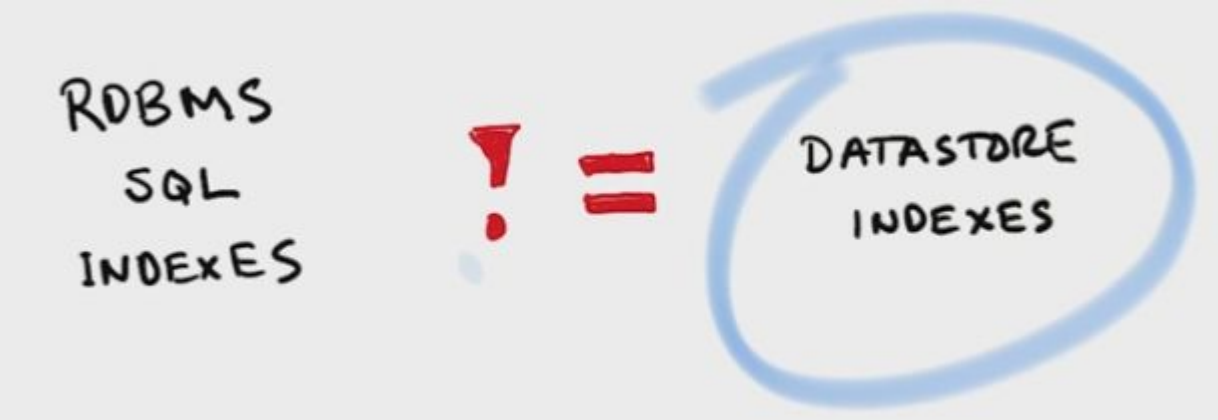
Filter by Property

- Давайте додамо метод в Api, що буде повертати всі конференції, що створені в якомусь конкретному місті (наприклад London) і має конкретний топік

```
@ApiMethod(
        name = "getConferencesFiltered",
        path = "getConferencesFiltered",
        httpMethod = HttpMethod.POST
    )
    public List<Conference> getConferencesFiltered(){

    Query query = ofy().load().type(Conference.class);
    query = query.filter("city =", "London");
    query = query.filter("topics =", "Web Technologies");
        return query.list();
    }
```

Datastore Indexes



Datastore Indexes

- Індекси в реляційних базах даних пришвидшують запити по полях
- В Datastore, якщо ви хочете робити запит по полю то поле має мати індекс

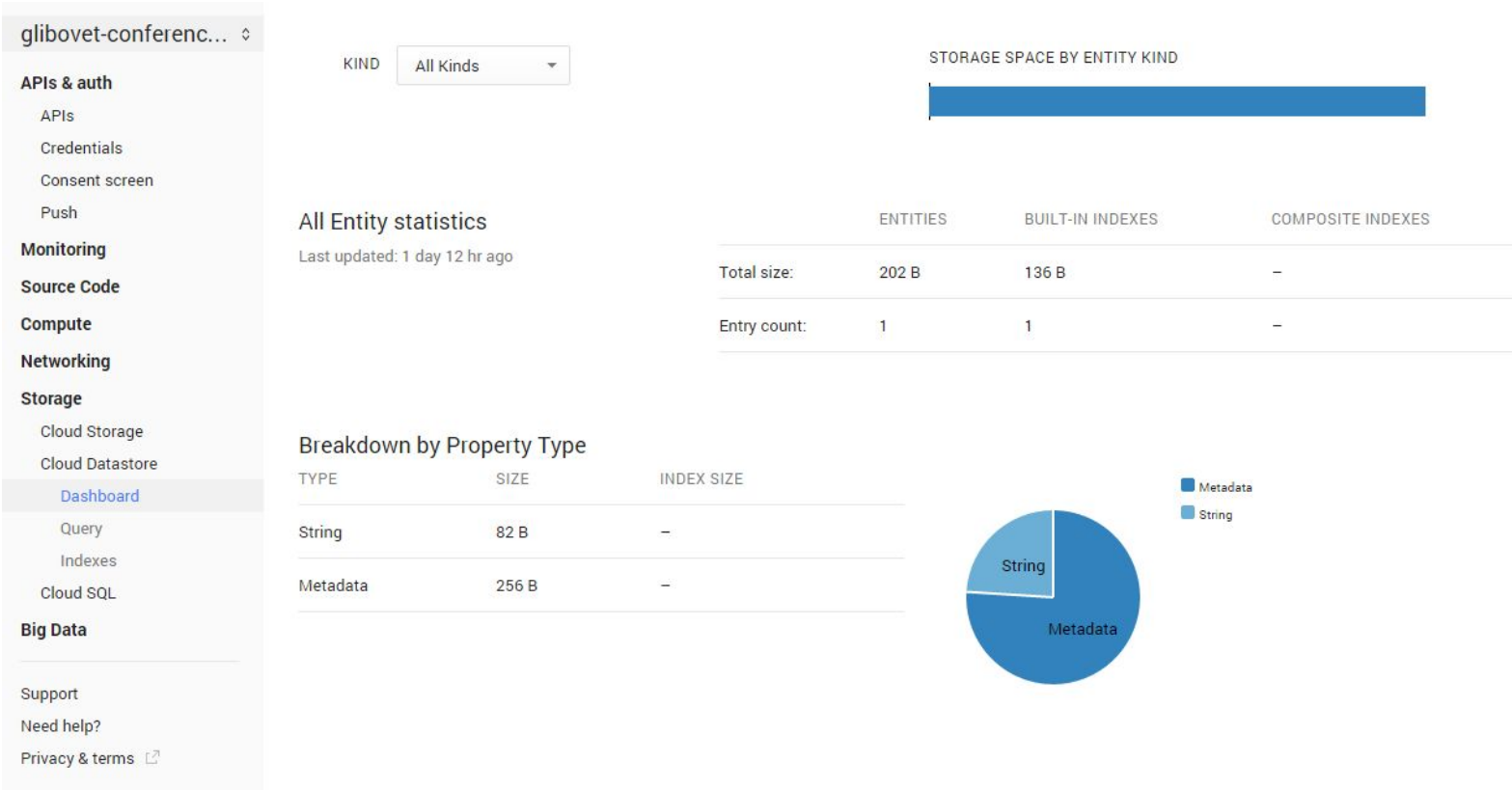
Datastore Indexes

- Коли ви зберігаєте значення, Datastore зберігає відповідний ключ Entity

Datastore Indexes

- Коли вам потрібно знайти всі Entity які мають поле City = London
- Datastore подивиться запис
  - Conference/city/Paris і знайде всі відповідні Entity ключі

Size of INDEX tables



Datastore Indexes

- В великому застосуванні індекси можуть займати більше місця ніж самі дані
- Тому вам дуже важливо визначитися за якими полями ви хочете вміти робити запити
- За замовчанням всі поля індексуються, навіть якщо ви забули анотацію @Index
- Але коли ми використовуємо Objectify відбувається навпаки, якщо поле не має анотації воно не індексується
  - Тому якщо ми вирішили, що поле має індексуватися ми обов'язково надаємо анотацію @Index
  - Якщо ми все таки хочемо, що б всі поля індексувалися, ми надаємо анотацію @Index класу і використовувати анотацію @UnIndex для полів, що треба виключити

Composite Indexes

- Ми розглянули звичайні індекси, а що ви скажете про наступний запит
  - Retrieve all Conferences
    - filter by CITY and TOPIC
    - sort by NAME
- Звичайні індекси не можуть бути комбіновані для відповіді на такий запит
- Поясніть мені чому?

Composite Indexes

- Для відповіді на попередній запит нам потрібен індекс який буде містити різні варіанти поєднань
- Це називається **composite indexes**
- Такі індекси можна створити наступним чином:
  - додати INDEX до INDEX файлу
  - або
  - запустити застосування локально, зробити запит і система автоматично створить index файл, який ви потім зможете завантажити в хмару

Composite Indexes

- Без композитного індексу в вас не будуть працювати запити.
- Якщо в вас все вірно налаштовано, то коли ви будете запускати своє локальне застосування, при першому ж складному запиті буде створюватися композитний індекс
- Цей індекс зберігається в файл
  - target/conference-1.0/WEB-INF/appengine-generated/datastore-indexes-auto.xml
- Цей файл буде залитий разом з вашим застосуванням в хмару і після цього в хмарі зможуть виконуватися ваші запити

Composite Indexes

- Якщо раптом в вас не створюється індекс, ви можете його вручну додати в цей файл
- Він має наступний вигляд
- <!-- Indices written at Wed, 7 Jan 2015 14:10:28 EET -->

- <datastore-indexes>
  - <datastore-index kind="Conference" ancestor="false" source = "auto">
    - <property name="city" direction="asc"/>
    - <property name="maxAttendees" direction="asc"/>
  - </datastore-index>
  - <datastore-index kind="Conference" ancestor="false" source = "auto">
    - <property name="city" direction="asc"/>
    - <property name="maxAttendees" direction="asc"/>
    - <property name="name" direction="asc"/>
  - </datastore-index>
  - <datastore-index kind="Conference" ancestor="false" source = "auto">
    - <property name="city" direction="asc"/>
    - <property name="name" direction="asc"/>
  - </datastore-index>
  - <datastore-index kind="Conference" ancestor="false" source = "auto">
    - <property name="maxAttendees" direction="asc"/>
    - <property name="name" direction="asc"/>
  - </datastore-index>
- </datastore-indexes>

Query Restrictions

- Фільтр нерівності можна використовувати лише один раз
  - цей вираз не вірний
    - startdate > 15th June && maxattendees <1000
- Властивість, що приймає участь в нерівності має бути відсортована першою
  - цей вираз не вірний
    - maxattendees < 1000 SORT BY NAME

Query Restrictions

- Давайте допишемо один фільтр і запустимо.
- Як ви думаєте, який буде результат?

```
@ApiMethod(
        name = "getConferencesFiltered",
        path = "getConferencesFiltered",
        httpMethod = HttpMethod.POST
    )
    public List<Conference> getConferencesFiltered(){

    Query query = ofy().load().type(Conference.class).order("name");
    query = query.filter("city =", "London");
    query = query.filter("topics =", "Web Technologies");
    query = query.filter("month =", 1);
    query = query.filter("maxAttendees >",10);
       return query.list();
    }
```

Query Restrictions

- Виправимо
- @ApiMethod(
- name = "getConferencesFiltered",
- path = "getConferencesFiltered",
- httpMethod = HttpMethod.***POST***
-)
- **public List<Conference> getConferencesFiltered(){**
  - Query query = *ofy().load().type(Conference.**class);***
  - query = query.filter("maxAttendees >",10) query = query.filter("city =", "London");
  - query = query.filter("topics =", "Web Technologies");
  - query = query.filter("month =", 1) .order("maxAttendees").order("name");
  - **return query.list();**
- }

Фільтр

- Ви пам’ятаєте, що в нашому застосуванні є фільтр.
- Ми з вами написали метод
  - **public List<Conference> queryConferences() {**
    - Query query = *ofy().load().type(Conference.**class).order("name");***
    - **return query.list();**
  - }
- Але він повертає всі конференції відсортовані за назвою
- Якщо ми зараз запустимо застосування і спробуємо додавати фільтри, вони не будуть працювати
- Нам потрібно доробити наш метод

Фільтр

- В нас в додатках є файл ConferenceQueryForm скопіюємо його в відповідний пакет
- Дуже цікавий клас, я раджу його розібрати
- Тепер ми можемо приймати в метод ConferenceQueryForm
- і замість існуючого коду вставити
  - **public List<Conference> queryConferences(ConferenceQueryForm conferenceQueryForm) {**
    - **return conferenceQueryForm.getQuery().list();**
  - }
- тепер ваші фільтри мають почати працювати

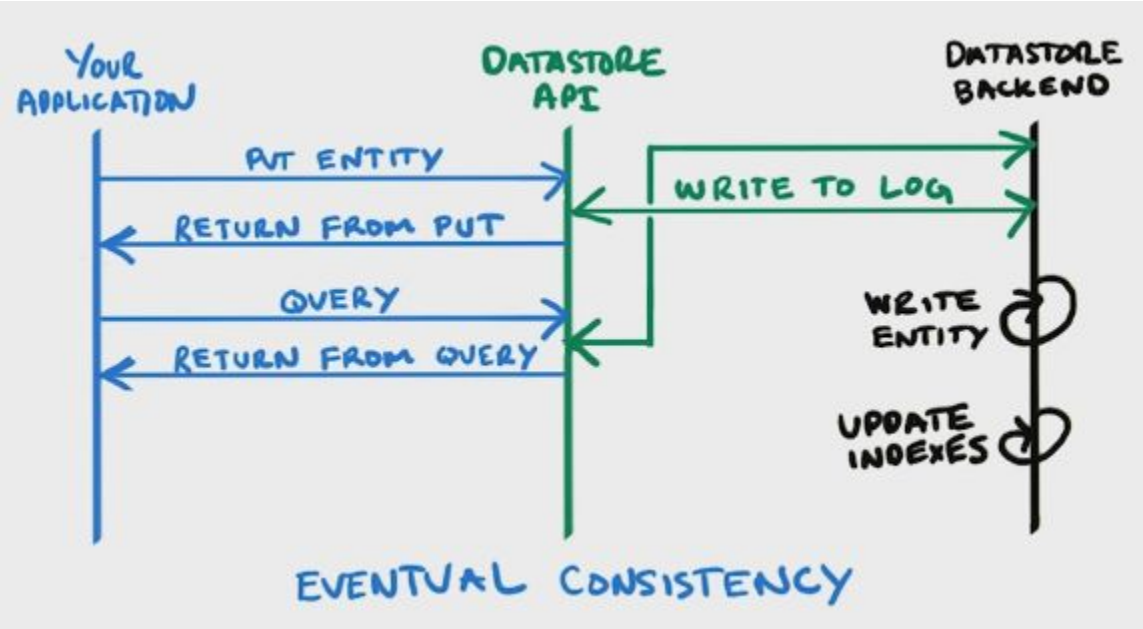
Фільтр

- Трохи оптимізації
- public List queryConferences(ConferenceQueryForm conferenceQueryForm) {
  - Iterable<Conference> conferenceIterable = conferenceQueryForm.getQuery();
  - List<Conference> result = **new ArrayList<>(0);**
  - List<Key<Profile>> organizersKeyList = **new ArrayList<>(0);**
  - **for (Conference conference : conferenceIterable) {**
    - organizersKeyList.add(Key.*create(Profile.**class,** **conference.getOrganizerUserId()));***
    - result.add(conference);
  - }
  - // To avoid separate datastore gets for each Conference, pre-fetch the Profiles.
  - *ofy().load().keys(organizersKeyList);*
  - **return result;**
- }

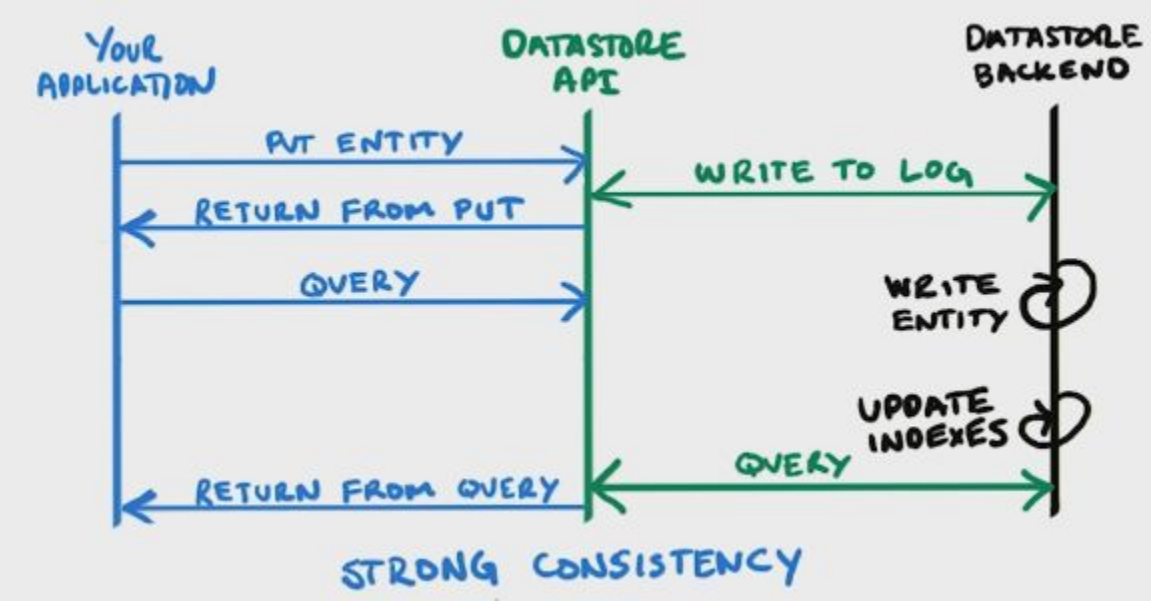
Datastore Commit Process

- Datastore has two consistency models
  - eventual consistency
  - strong consistency

Datastore Commit Process
Eventual Consistency



Datastore Commit Process
Strong Consistency



Eventual vs Strong

- Яка стратегія краще підходить для ?
  - блог
  - ATM money

Eventual vs Strong

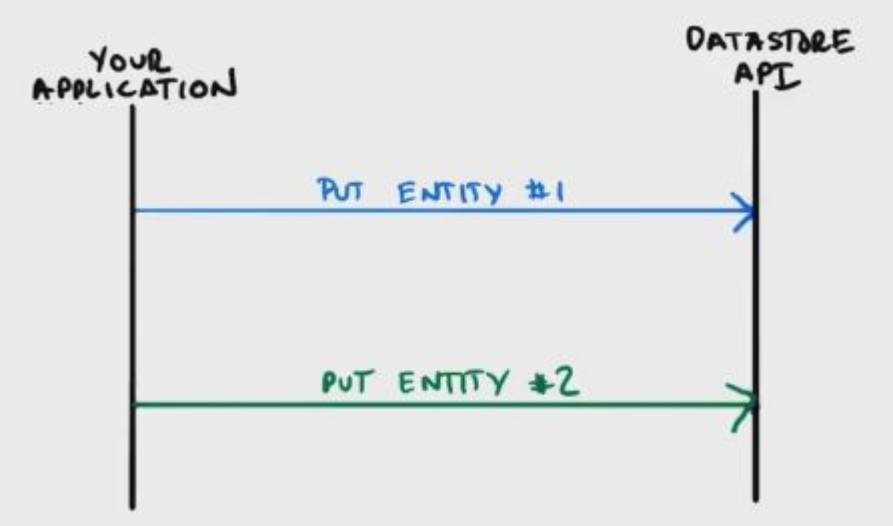
- За замовчанням використовується Eventual consistency
- Але якщо ви робите запит
  - Ancestor relationship
  - Filter by ancestor
  - всі сини будуть видобуті використовуючи Strong consistency

Транзакції

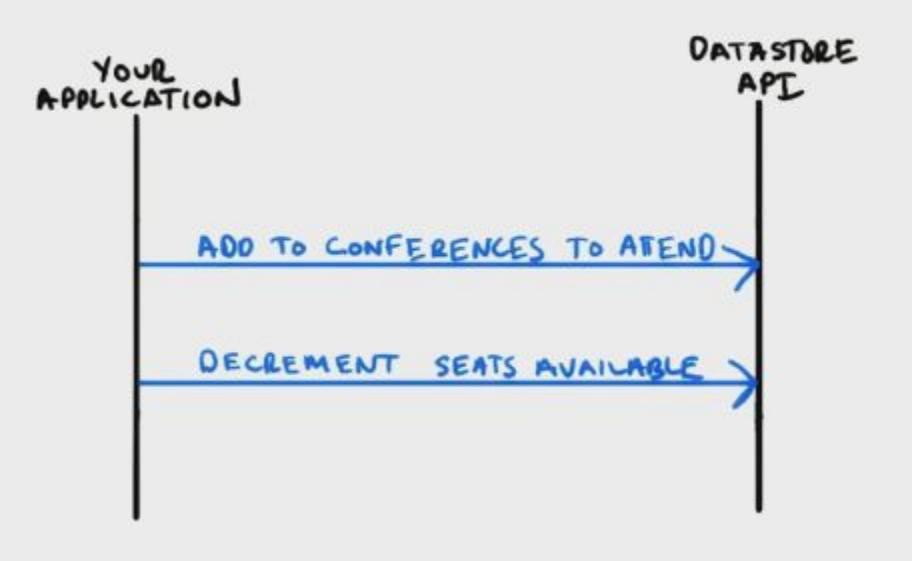
- В нашому випадку коли користувач реєструється на конференцію ми зв’язуємо конференцію і користувача

Транзакції

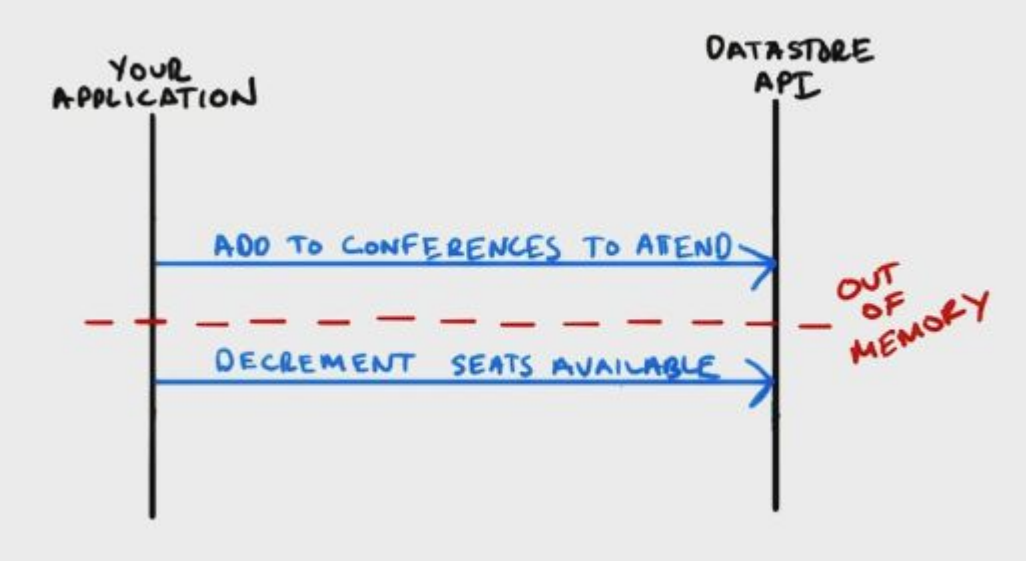
- В стандартному випадку наші запити виглядають наступним чином
- Зазвичай цього достатньо, але бувають випадки …
- Як ви думаєте, що це за випадки?



Транзакції

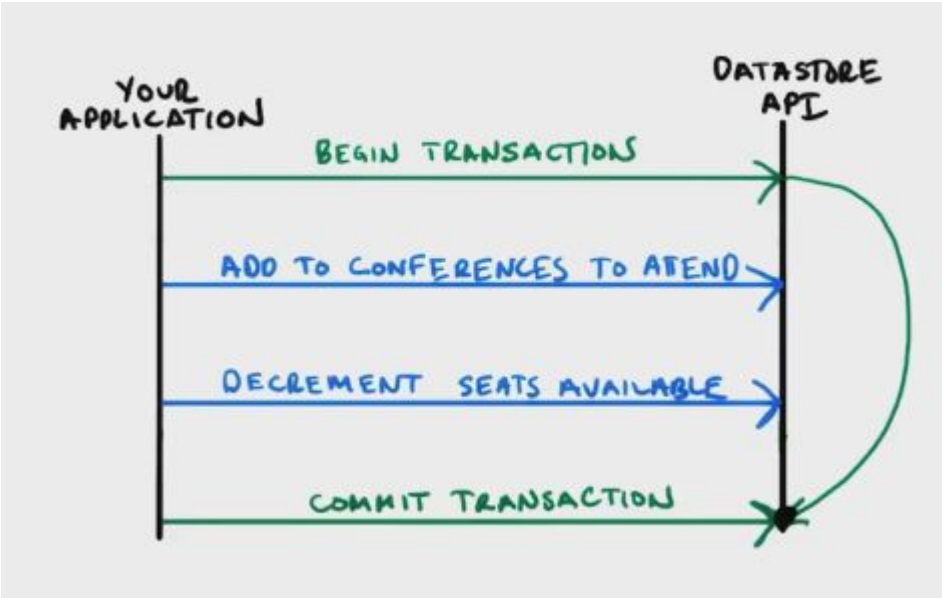

YOUR APPLICATION
DATASTORE API
ADD TO CONFERENCES TO ATEND
DECREMENT SEATS AVAILABLE

Транзакції

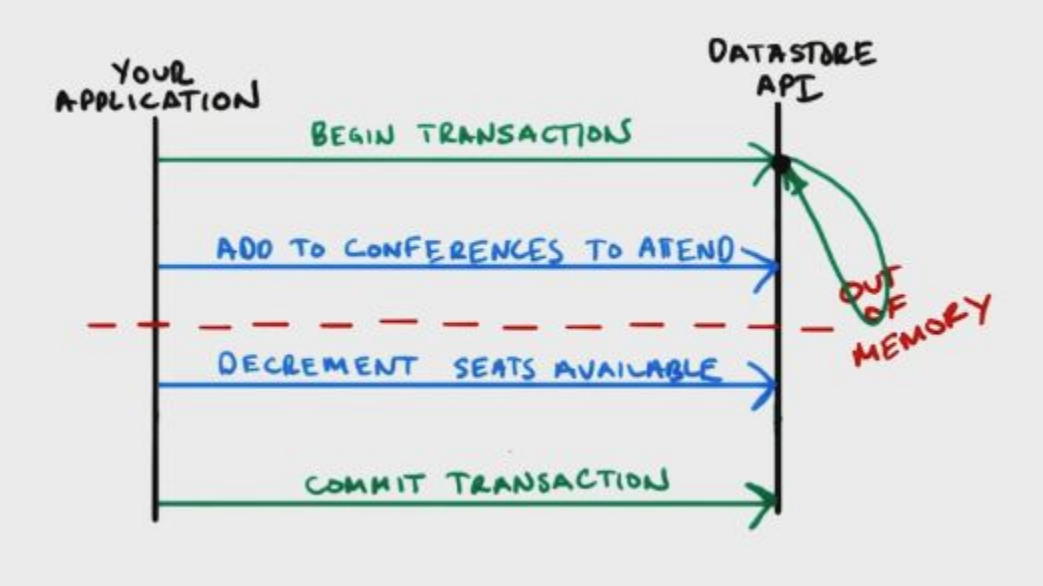

YOUR APPLICATION
DATASTORE API
ADD TO CONFERENCES TO ATTEND
OUT OF MEMORY
DECREMENT SEATS AVAILABLE

Транзакції

- Дана проблема може бути вирішена за допомогою транзакцій



Транзакції


YOUR APPLICATION
DATASTORE API
BEGIN TRANSACTION
ADD TO CONFERENCES TO ATTEND
OUT OF MEMORY
DECREMENT SEATS AVAILABLE
COMMIT TRANSACTION

Транзакції

- Давайте впровадимо транзакції в наше застосування
- В нас поки, що не реалізована функція реєстрації на конференцію
- Давайте зробимо це

Реєстрація

- При реєстрації на конференцію ми маємо зробити наступні речі
  - зареєструвати користувача на конференцію
  - зменшити кількість вільних місць
  - вирішити, що робити в конфліктній ситуації
    - вільне місце залишилося одне а претендентів декілька
    - вони одночасно роблять запит

Реєстрація

- Спочатку ми змінимо профайл користувача і додамо поле, що буде відповідати за конференції на які підписався користувач
  - private List <String> conferenceKeysToAttend = new ArrayList<> (0);

- Додамо методи
  - один має повертати копію списку конференцій на які записаний користувач
  - інший має додавати конференцію до списку
  - видалити конференцію з списку

Реєстрація

- Тепер нам необхідно додати методи в ConferenceApi
- Їх заглушки знаходяться в додатку registerForConference-skeleton-and-other-additions.txt

  - @Named – ви маєте імпортувати **import javax.inject.Named;**
  - **import com.google.api.server.spi.response.NotFoundException;**

Реєстрація

- Майже весь код готовий, ви маєте реалізувати лише один метод registerForConference
- Але що б його написати ви маєте розібратися з транзакціями

Реєстрація

- Запуск транзакції
- <T> result = ofy().transact(new Work <T> {
  - public <T> run () {
    - // do stuff
    - // do more stuff
    - return <T>;
  - }
- });

Transaction rules

- Snapshot isolation
- Optimistic concurency

Snapshot isolation

- Всі запити на читання мають повертати значення які мало сховище до початку транзакції
- Updates не будуть видимі ззовні
  - все або нічого

Optimistic concurency

- Commit може бути тільки успішним (всі дії успішні)
- Значення, що отримують нове значення не мають бути змінені з моменту початку транзакціїї
  - до початку транзакції
    - a = 2
  - в ході вашої транзакції ви змінили a =6
  - коли ви пробуєте записати транзакцію, якщо a в базі не 2 ваша транзакція не відбудеться
- Одна транзакція може модифікувати максимум 5 Ancestor Groups
- Має бути завершена за 60 секунд

Конференції на які ми записані

- Ми не реалізували ще одну річ
- Ми не показуємо конференції на які зареєструвався користувач
- Треба дописати і цей метод
  - @ApiMethod(
  - name = "getConferencesToAttend",
  - path = "getConferencesToAttend",
  - httpMethod = HttpMethod.***GET***
  -)
  - **public Collection<Conference> getConferencesToAttend(final User user)**
  - **throws UnauthorizedException, NotFoundException {**
  - **…**
  - **}**

Відписка

- Також ми маємо дати можливість користувачу відписатися від конференції
  - Допишіть метод самостійно
  - Заглушка метода присутня в додатках

unregisterFromConference

- /**
  - * Unregister from the specified Conference.   *
  - * @param user An user who invokes this method, null when the user is not signed in.
  - * @param websafeConferenceKey The String representation of the Conference Key to unregister from.
  - * @return Boolean true when success, otherwise false.
  - * @throws UnauthorizedException when the user is not signed in.
  - * @throws NotFoundException when there is no Conference with the given conferenceId.
- */
- @ApiMethod(
  - name = "unregisterFromConference",
  - path = "conference/{websafeConferenceKey}/registration",
  - httpMethod = HttpMethod.DELETE)
- public WrappedBoolean unregisterFromConference(
  - final User user,
  - @Named("websafeConferenceKey") final String websafeConferenceKey
-) throws UnauthorizedException, NotFoundException, ForbiddenException, ConflictException {
- }

- Дякую за увагу