



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

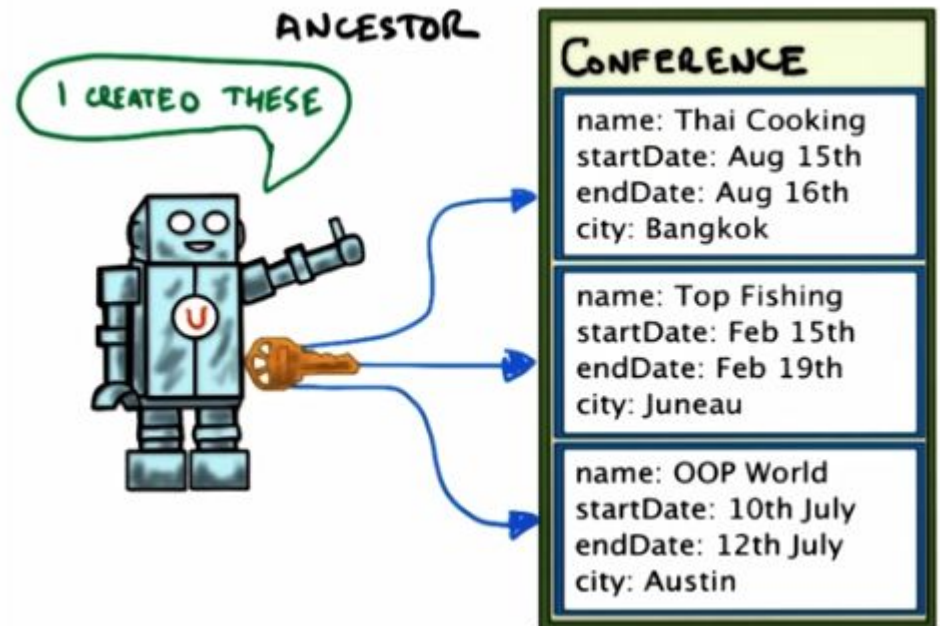


DATA RELATIONSHIPS

□ Ancestor

- assigned at creation
- can never be changed

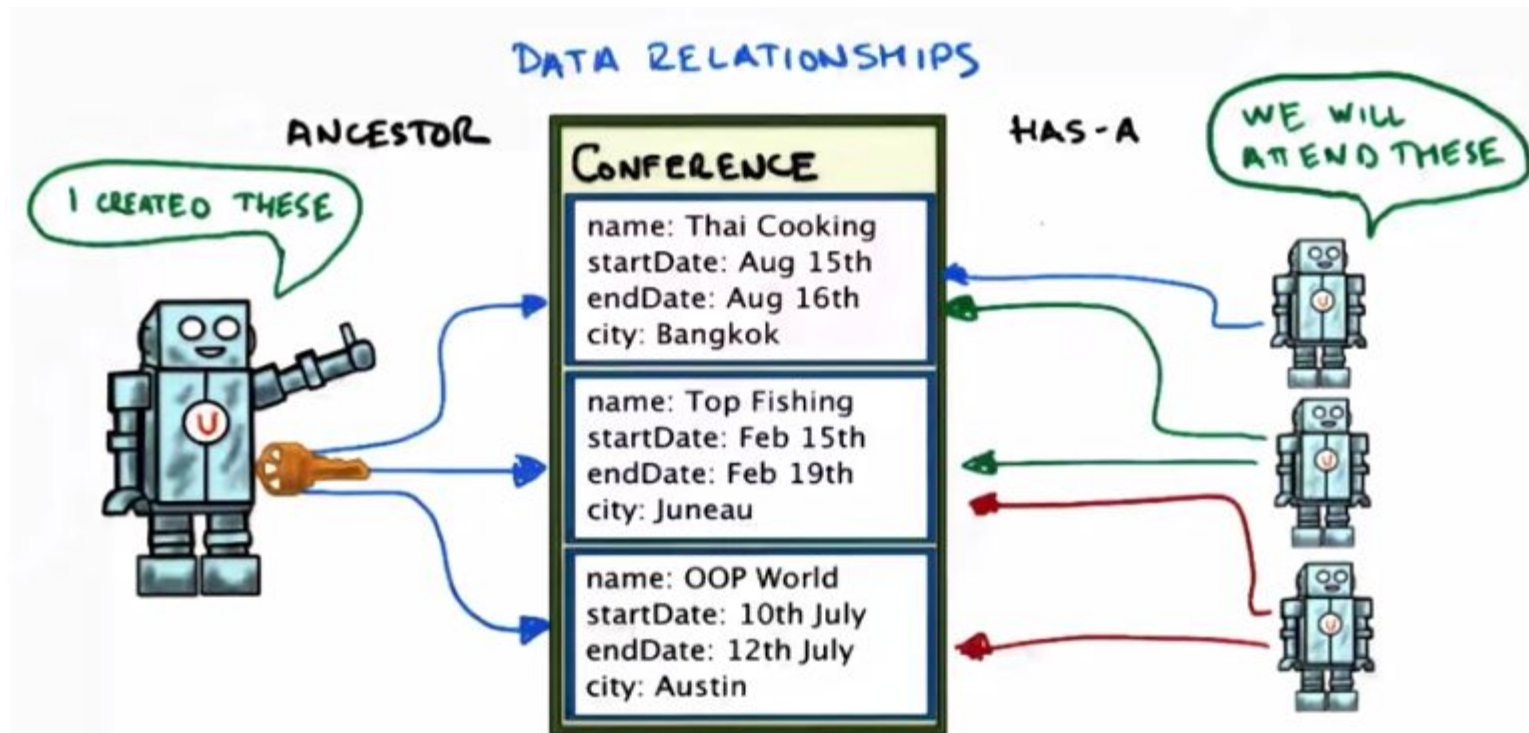
- в нашому застосуванні будуть конференції і користувачі, що створюють конференції
- хто буде ancestor?
 - **USER**



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



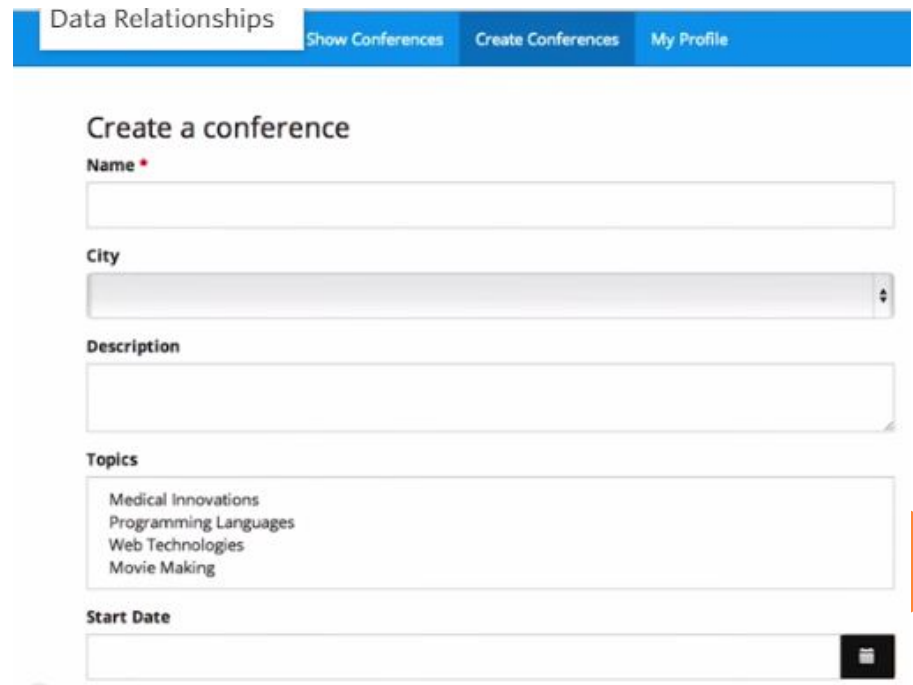
КОНФЕРЕНЦІЇ

- 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



The screenshot shows a web application interface for creating a conference. At the top, there is a blue navigation bar with the text "Data Relationships" and three buttons: "Show Conferences", "Create Conferences", and "My Profile". Below the navigation bar, the main heading is "Create a conference". The form consists of several fields: "Name" (a text input field with a red asterisk indicating it is required), "City" (a dropdown menu), "Description" (a text area), "Topics" (a list of topics: "Medical Innovations", "Programming Languages", "Web Technologies", and "Movie Making"), and "Start Date" (a date input field with a calendar icon on the right).

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 клас
 - `factory().register(Conference.class);`



CONFERENCEAPI

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



CONFERENCEAPI

□ Дopiшeмo кoд:

- Спoчaткy oтpимaємo `Key<Profile>` кoриcтyвaчa
 - `Key<Profile> profileKey = Key.create(Profile.class,userId);`
- Пiсля цьoгo видiлiмo ключ для нaшoї кoнфeрeнцiї
- Синтaкcис мaє вигляд
 - `Key<T> key = factory().allocatedId(Entity.Class);`
 - `factory()` cтaтичний мeтoд в `OfyService`
- Aлe в нaшoмy випaдкy цe нe зoвciм вiрнo тaк як нaш клac `Conference` мaє бaтькa `Profile`
- в тaкoмy випaдкy cинтaкcис мaє 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
 - запусіть 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 кожній 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 самотійно 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

CONFERENCE
name: Big Horror topics: Movie Making startDate: Jul 11th endDate: Jul 12th city: Halloween
name: Running topics: Health and Nutrition startDate: Nov 14th endDate: Nov 16th city: Halloween

RETRIEVE ALL CONFERENCES
WHERE CITY = HALLOWEEN
AND
HAS MOVIE MAKING IN TOPIC
SORT RESULT BY NAME



FILTER BY PROPERTY

- EQUALITY FILTER: $=$
- MEMBER OF FILTER: \in
- INEQUALITY FILTER:
 - \neq NOT EQUAL TO
 - $<$ LESS THAN
 - \leq LESS THAN OR EQUAL TO
 - $>$ GREATER THAN
 - \geq GREATER THAN OR EQUAL TO



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

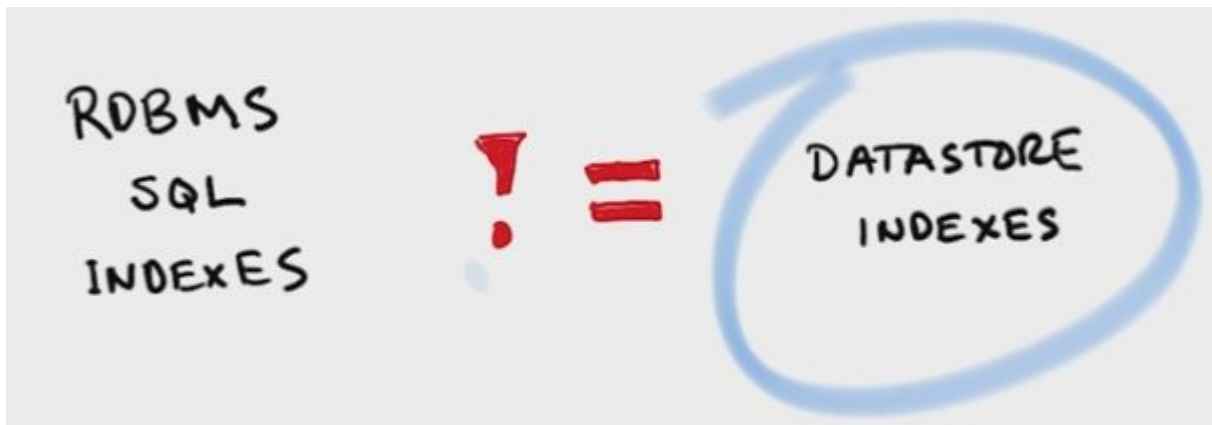


FILTER BY PROPERTY

- Давайте додамо метод в Арі, що буде повертати всі конференції, що створені в якомусь конкретному місті (наприклад London) і має конкретний топик
- `@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

- Індокси 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

glibovet-conferenc... ▾

APIs & auth

- APIs
- Credentials
- Consent screen
- Push

Monitoring

Source Code

Compute

Networking

Storage

- Cloud Storage
- Cloud Datastore

Dashboard

- Query
- Indexes
- Cloud SQL

Big Data

- Support
- Need help?
- Privacy & terms ↗

KIND All Kinds ▾

All Entity statistics

Last updated: 1 day 12 hr ago

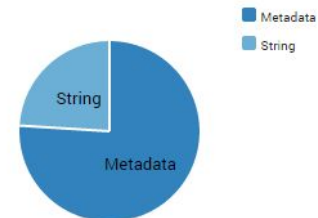
STORAGE SPACE BY ENTITY KIND



	ENTITIES	BUILT-IN INDEXES	COMPOSITE INDEXES
Total size:	202 B	136 B	-
Entry count:	1	1	-

Breakdown by Property Type

TYPE	SIZE	INDEX SIZE
String	82 B	-
Metadata	256 B	-



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 використовуємо 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 запусимо 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;
```
- `}`



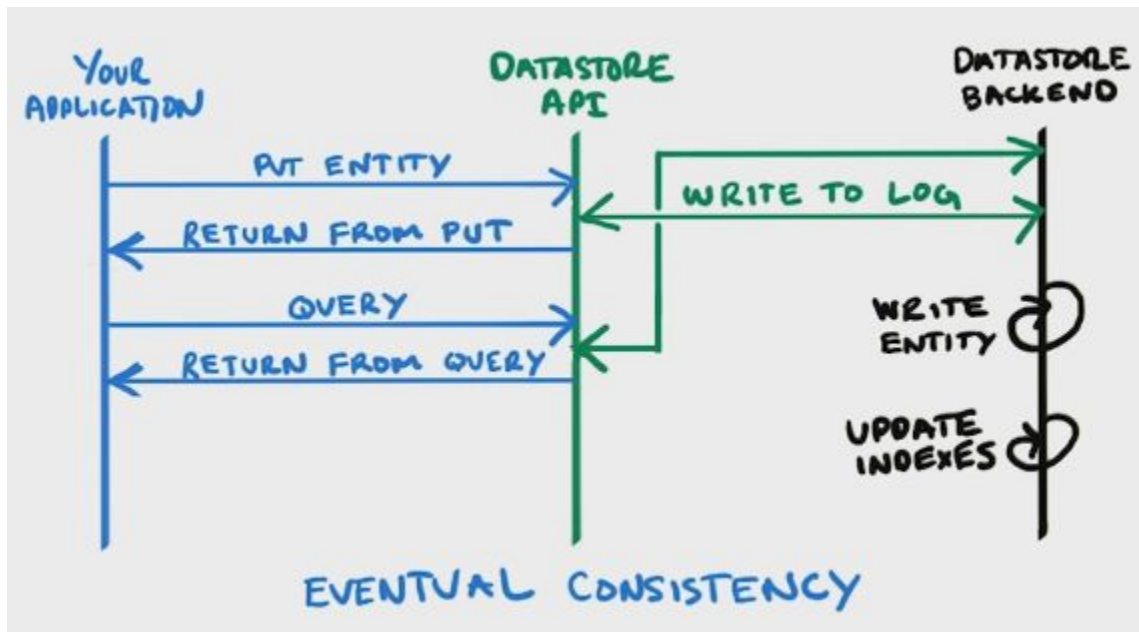
DATASTORE COMMIT PROCESS

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



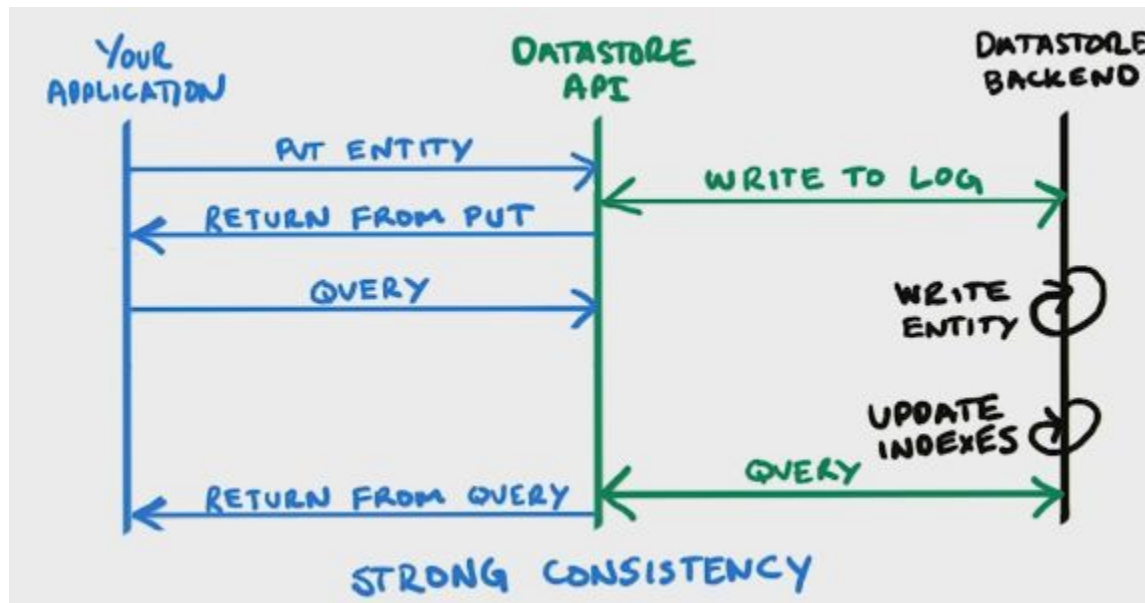
DATASTORE COMMIT PROCESS

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



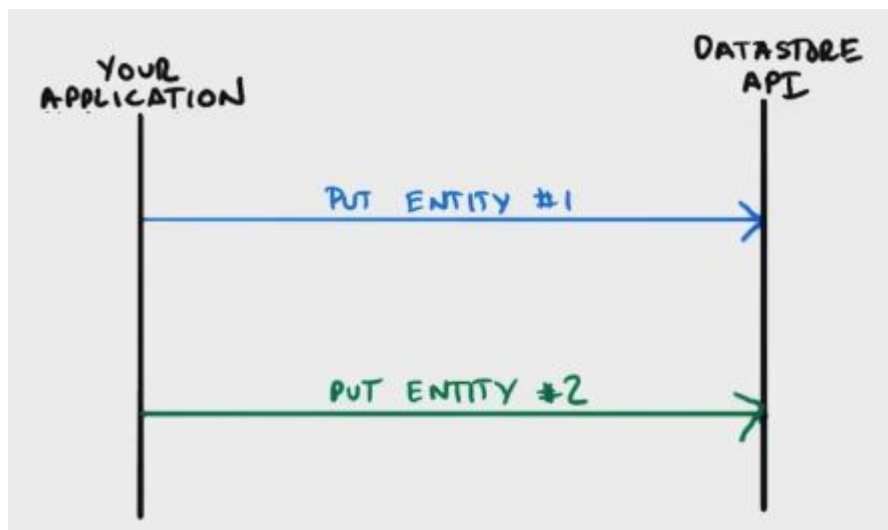
ТРАНЗАКЦІЇ

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

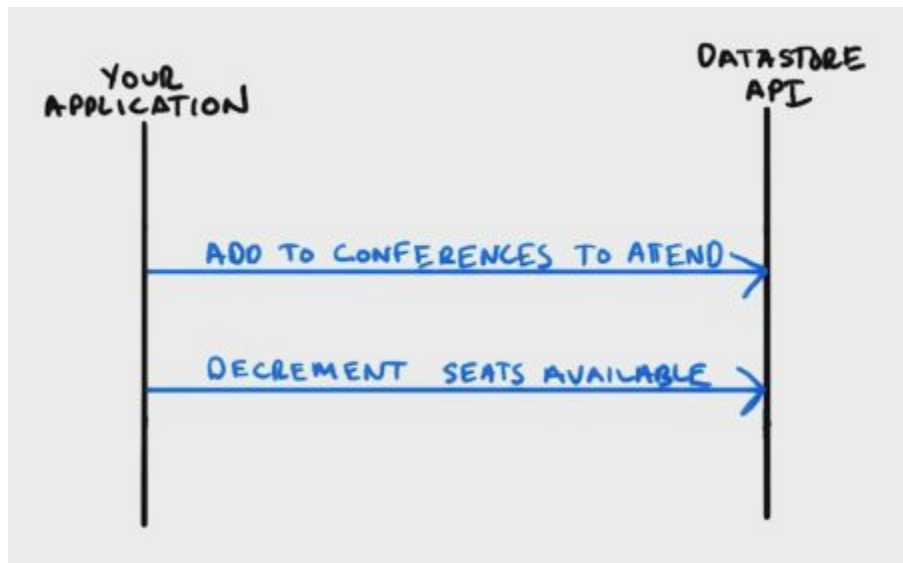


ТРАНЗАКЦІЇ

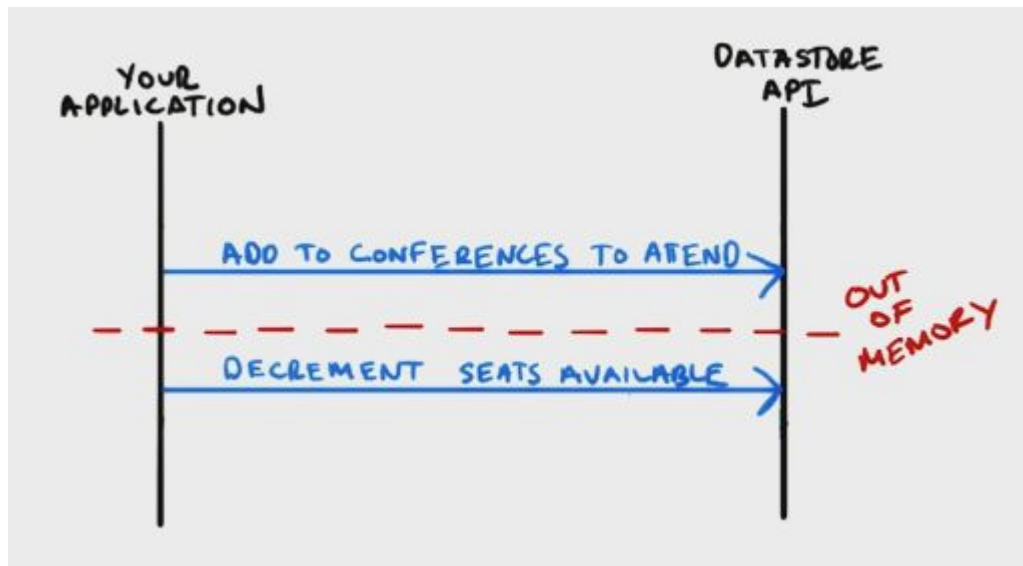
- В стандартному випадку наші запити виглядають наступним чином
- Зазвичай цього достатньо, але бувають випадки ...
- Як ви думаєте, що це за випадки?



ТРАНЗАКЦІЇ

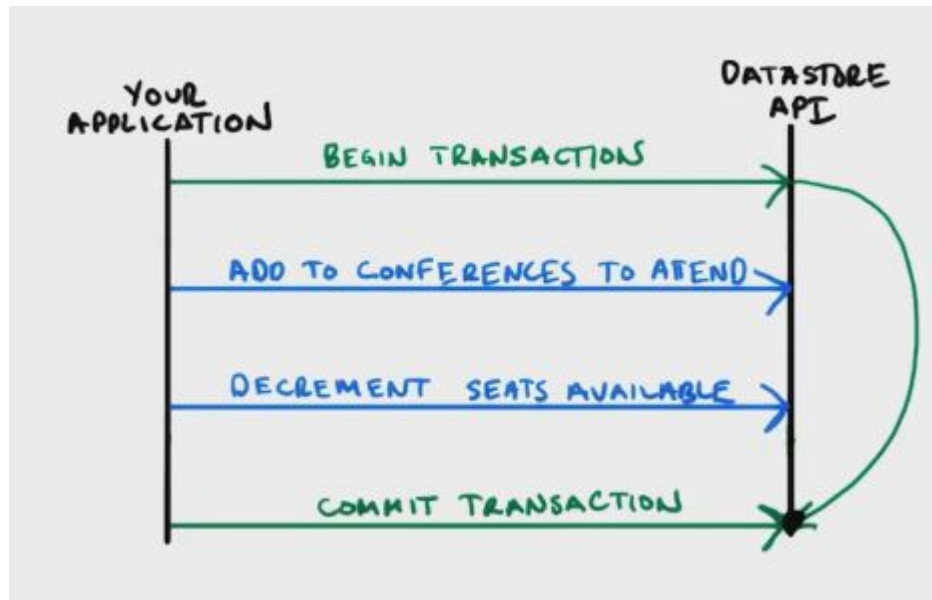


ТРАНЗАКЦІЇ

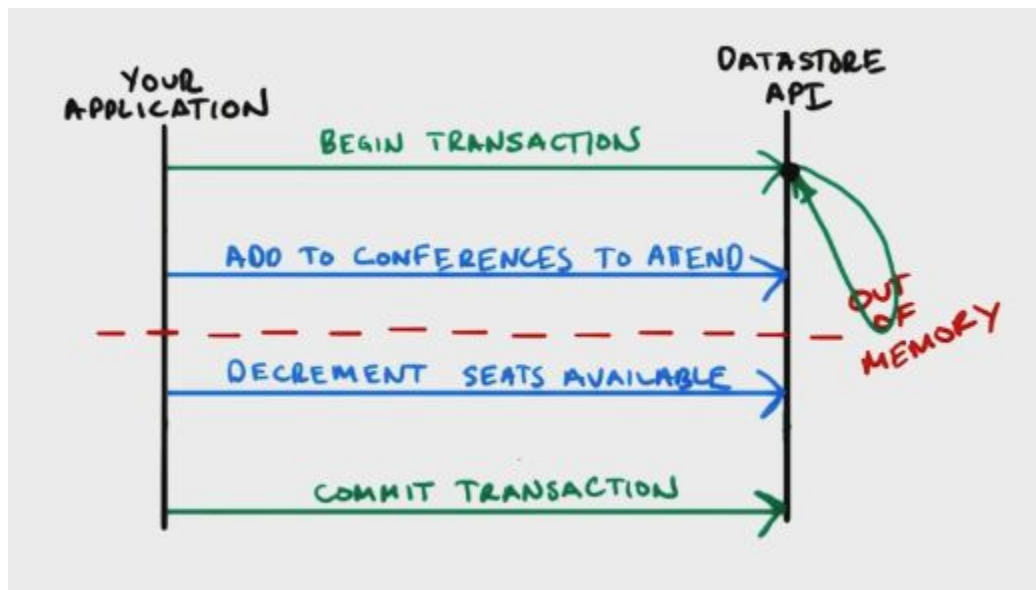


ТРАНЗАКЦІЇ

- Дана проблема може бути вирішена за допомогою транзакцій



ТРАНЗАКЦІЇ



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 concurrency



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 транзакції
 - 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

