Datastore Entity Groups and Transactions

Google App Engine

Google Cloud Platform

Agenda

1. Consistency Model
2. Ancestor Queries & Entity Groups
3. Transactions
4. Lab & Exercise

Eventual vs. Strong Consistency

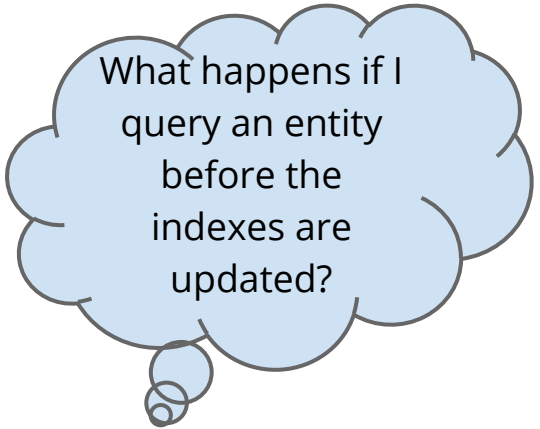
Google Cloud Platform

Every Query Needs an Index

- Every query to the datastore uses an index:
  - An automatically-generated single property index or
  - A custom index

When Are Indexes Updated?

- Every entity update has multiple writes
- Commit phase
  - Writes data in log
- Write phase
  - Writes data to datastore
  - Update indexes (might take longer than writing to the datastore)
- If commit phase succeeds, write phase is guaranteed to succeed
- **But … might not happen immediately**



Consistency

- When you run a query, what do you get?

```
Conference entity:
        startDate:  April  15
        confName: "App Engine for All"
```

1. Change startDate property to May 15
2. Save the entity in the datastore
3. Query for all conferences with start date in May

Consistency

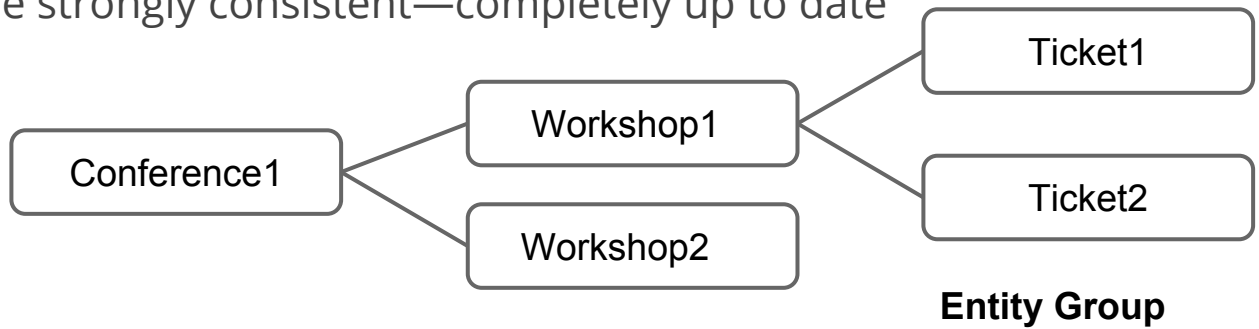
- When you run a query, what do you get?

Conference entity:
startDate: May 15
confName: "App Engine for All"

1. Change startDate property to May 15
2. Save the entity in the datastore
3. Query for all conferences with start date in May

Consistency: Strongly Consistent

- How do you guarantee to get the latest result ?
- Queries always return results from the **INDEX**
- get() forces the index to update before returning the results
- To get latest updates, use ancestor queries
- Ancestor queries force the index to update
- Results only include descendants of a specific entity.
- Results are strongly consistent—completely up to date



Consistency

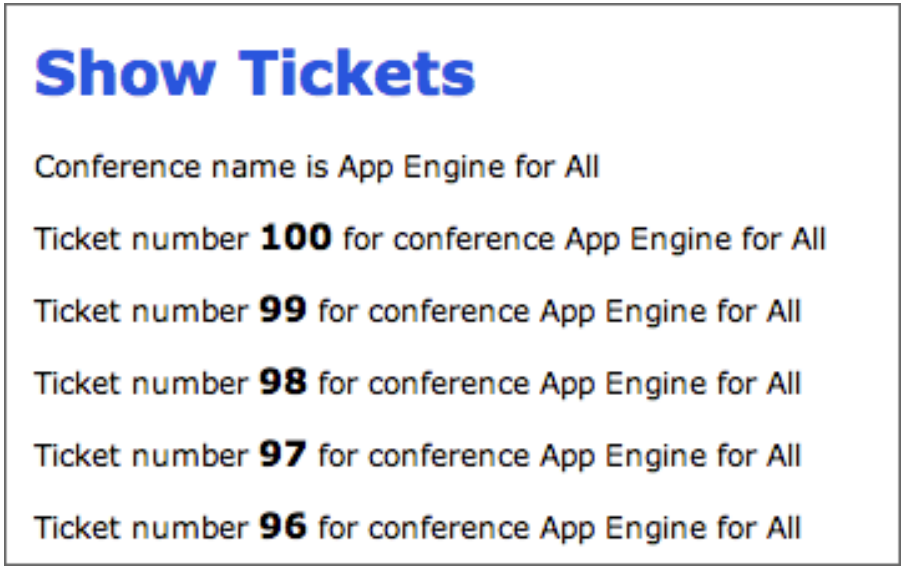
- **Eventually Consistent**
  - Queries without an ancestor filter get results from the last index update

- **Strongly Consistent**
  - Queries using an ancestor filter forces applicable index updates to complete

Consistency: Strong Consistent Example

1. Create 100 tickets for a conference
2. Query for the conference's tickets

Use an Ancestor Filter for Strong Consistency



Consistency: Eventual Consistency Example

1. Create 100 tickets for a conference
2. Query for the conference's tickets

Query without an ancestor filter: Results are eventually consistent

Ancestor Queries

Simple Query

Java

```
Query query = new Query("Person");
Query.Filter filter = new FilterPredicate("name", FilterOperator.EQUAL, "John");
query.setFilter(filter);
PreparedQuery results = datastore.prepare(query);
```

Python

```
q = Person.all()
q.filter("name =", "John")

// GqlQuery
q = db.GqlQuery("SELECT * FROM Person WHERE name = :1", "John")
```

Ancestor Query

Java

To add an ancestor filter to a query:

```
Query query = new Query("Kind");
query.setAncestor(parentKey);
```

Query for Workshop entities that are descendants of a particular conference:

```
Query query = new Query("Workshop");
query.setAncestor(conferenceKey);
```

Ancestor Query

Python

To add an ancestor filter to a query:

```
query = Kind.all()
query.ancestor(parent)
```

Query for Workshop entities that are descendants of a particular conference:

```
query = Workshop.all()
query.ancestor(conference)
```

Query For Descendents

- Use **ancestor filter** to query for descendants of an entity

Entity Groups

- What are entity group used for?
- Entity groups are used in
  - Ancestor queries
  - Transactions

Transactions

Transaction

Transaction: **a set of operations** performed on a data store, that preserves **ACID** characteristics.

- **Atomicity**
  Each transaction is "All or Nothing"
- **Consistency**
  Each transaction brings the datastore from one valid state to another
- **Isolation**
  Concurrent execution of transactions does not break consistency
- **Durability**
  Committed results of transaction persist after hardware failures

Transaction

- Isolation and Consistency
- Transaction requirement:
  - *Concurrent execution of transactions must not break consistency*
- Datastore uses:
  - snapshot isolation (serializable isolation)
  - optimistic concurrency

Transaction: Snapshot Isolation

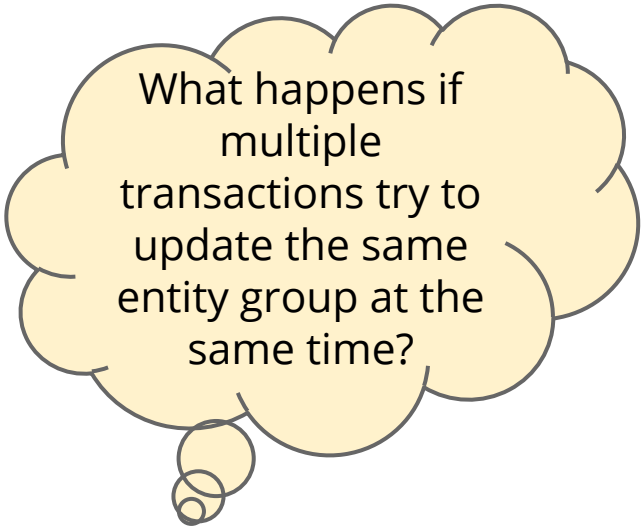
- Snapshot Isolation
- All reads in a transaction reflect the state of the Datastore at the time the transaction *started*
- If an entity is modified or deleted in the transaction, a query or get returns the *original* version of the entity, or nothing if the entity did not exist then

Transaction: Optimistic Concurrency

Optimistic Concurrency:

A transaction commits its changes only if:

- the values updated by the transaction have not changed since the snapshot was taken



- First transaction to commit changes succeeds
- All others fail
- Others can try again to apply changes to the updated data

Transaction: Code Sample

```
DatastoreService datastore = DatastoreServiceFactory.getDatastoreService()
Transaction txn = datastore.beginTransaction();
try {
  Key ekey = KeyFactory.createKey("Employee", "Joe");
  Entity employee = datastore.get(eKey);

  /*... reading and writing on employee ...*

  datastore.put(employee);
  txn.commit();

} finally {
    if (txn.isActive()) {
        txn.rollback();
    }
}
```

Transaction: Code Sample

- Put the actions to perform in the transaction in a function
- @db.transactional handles concurrency retry

```
from google.appengine.ext import db

@db.transactional
def my_txn():
    x = MyModel(a=3)
    x.put()
    y = MyModel(a=7)
    y.put()

my_txn()
```

Transaction: Python

- If a function can run transactionally or non-transactionally, call it with run_in_transaction

```
from google.appengine.ext import db

def my_txn():
    x = MyModel(a=3)
    x.put()
    y = MyModel(a=7)
    y.put()

db.run_in_transaction(my_txn)
```

Transaction: Entity Operation

- What Entity Operations can a Transaction Do?

- Operations on an entity group:
  - Create entities
  - Update entities
  - Delete entities

- Single entity group transactions:
  update a single entity group

- Cross-entity group transactions:
  update up to **25** entity groups

Queries inside a transaction return results from the state of the datastore before the transaction started

Transaction: Limit

- Limit to the number of entity groups
  - 1 for single entity group transaction
  - 25 for cross-entity group transaction
- Limit to number of updates per entity group per second
  - Usually between 1 and 5 updates per second
- Duration limits
  - Max duration of 60 seconds

Transaction: X-Group

- Is this a valid X-Group Transaction?

	Action	**Kind**	**Entity Name**	**Parent Name**
1.	Create and save entity	HotelChain	SouthWinds	
2.	Create and save entity	Venue	Venue1	SouthWinds
3.	Create and save entity	Cafe	Cafe1	Venue1
4.	Create and save entity	HotelChain	NorthWinds	
5.	Create and save entity	Venue	Venue2	NorthWinds
6.	Create and save entity	Cafe	Cafe2	Venue2
7.	Create and save entity	Cafe	Cafe3	Venue2
8.	Create and save entity	Menu	MondayMenu	Cafe3

Transaction: X-Group

- Is this a valid X-Group Transaction?

	Action	Kind	Entity Name	Parent Name
1.	Create and save entity	HotelChain	SouthWinds	
2.	Create and save entity	Venue	Venue1	SouthWinds
3.	Create and save entity	Cafe	Cafe1	Venue1
4.	Create and save entity	HotelChain	NorthWinds	
5.	Create and save entity	Venue	Venue2	NorthWinds
6.	Create and save entity	Cafe	Cafe2	Venue2
7.	Create and save entity	Cafe	Cafe3	Venue2
8.	Create and save entity	Menu	MondayMenu	Cafe3

Transaction: X-Group

- Is this a valid X-Group Transaction?

	Action	Kind	Entity Name	Parent Name
1.	Create and save entity	HotelChain	SouthWinds	
2.	Create and save entity	Venue	Venue1	SouthWinds
3.	Create and save entity	Cafe	Cafe1	Venue1
	ity	HotelChain	NorthWinds	
	ity	Venue	Venue2	NorthWinds
	ity	Cafe	Cafe2	Venue2
7.	Create and save entity	Cafe	Cafe3	Venue2
8.	Create and save entity	Menu	MondayMenu	Cafe3

This is a valid transaction.
It modifies 2 entity groups

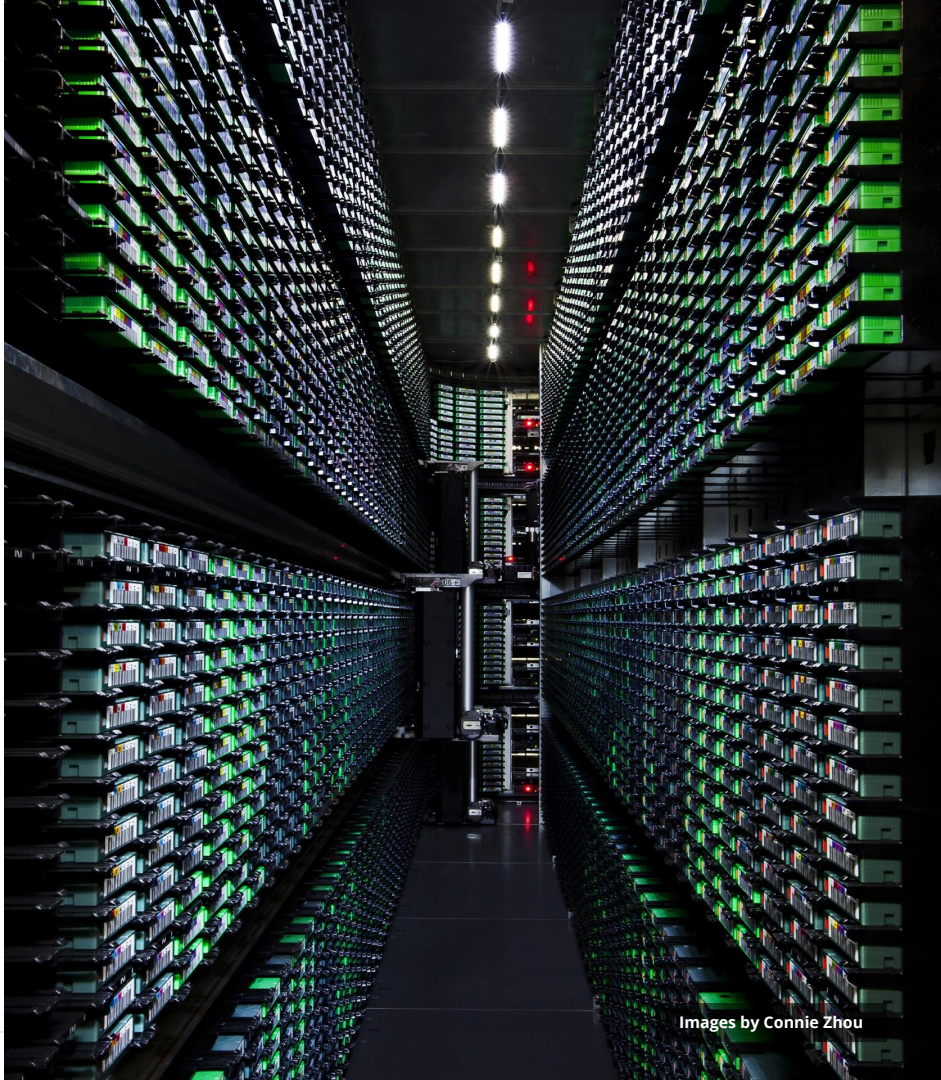
Transaction: Best Practices

- Best Practices
  - A transaction should happen quickly
    - minimize chances of external changes that conflict with the transaction
  - Prepare data outside the transaction
    - Prepare keys outside the transaction
    - Use the keys to fetch entities inside the transaction

Codelab

- You will use an enhanced version of the App Engine application introduced in the Introduction to Datastore hands-on exercise

- You will uncomment certain sections of the code that illustrate datastore consistency, entity groups and ancestor queries



Resources

- **Developer Documentation**
  - Datastore Queries (Java) (Python)
  - Transactions (Java) (Python)

- **App Engine Datastore Articles**
  - Mastering the Datastore series
  - Transaction Isolation
  - Handling Datastore Errors
  - Understanding Write Costs
  - How Entities and Indexes are Stored

Resources

- **Google I/O Sessions**
  - Under the Covers of the Google App Engine Datastore, Ryan Barret, Google I/O 2008
  - Building Scalable, complex Apps on App Engine, Brett Slatkin, Google I/O 2009
  - Next gen queries, Alfred Fuller, Google I/O 2010


cloud.google.com
Images by Connie Zhou