Weed3_Cassandra_CRUD_Operations

Write Operations in Cassandra

- Receiving node is the coordinator for the operation
- Writes directed to all partition replicas
- Acknowledgement expected from consistency number of nodes
- No reading before writing (by default)
- At node level
  - Writes are done in memory and later flushed on disk (SSTables)
  - Every flush => new SSTable
  - All disk writes are sequential ones. Data will be reconciled through Compaction
- Cassandra attaches a timestamp to every write

INSERT

- Insert operations require full primary key
- Cassandra does not perform read before write: an INSERT can behave as an UPSERT
  - If we INSERT data in an existing entry, data will be UPDATED
- Inserts require a value for all primary key columns, but not other columns
  - Only specified columns will be inserted/updated
- You can INSERT/UPDATE data with Time-To-Live

```
INSERT INTO groups (groupid, username, group_name, age)
VALUES (12, 'aland@gmail.com', 'baking', 32);

INSERT INTO grouops(groupid, username)
VALUES (45, 'moirad@yahoo.com');

INSERT INTO groups(groupid, username, group_name)
VALUES (25, 'john@gmail.com', 'vegan cooking') USING TTL 10;
```

UPDATE

`UPDATE groups SET group_name = 'grilling' WHERE groupid = 45;`

```
UPDATE groups SET age = 55, group_name = 'coffee' WHERE groupid = 20 and
username = 'bella@yahoo.com';
```

Lightweight transactions (LWT)

Are at least forut imes slower than the normal INSERT/UPDATE in Cassandra. Use them sparingly in your application

```
UPDATE groups SET AGE = 62 WHERE groupid=12 and username='aland@gmail.com'
IF EXISTS; // TRUE, age will be updated.
```

Summary

- Cassandra does not perform a read before writes by default, therefore INSERT and UPDATE operations behave similarly;
- Lightweight Transactions can be used in order to enforce a read before write;
- Cluster-level writes are sent to all partition replicas - Only number of nodes accvording to consistency are needed.

Read Operations in Cassandra

- Receiving node is coordinator for the operation
- Reads are directed only to number of replicas requipred for consistency
- Inconsistencies between contacted nodes are repaired during Read process

READ/SELECT Rules in Cassandra

- Start your query using the Partition Key
    - Limit reads to specific nodes containing your data
- Follow the order of your Primary Key columns

`Primary Key(PartitionKey, ClusteringKey1, ClusteringKey2`

Good commands:

```
SELECT * FROM table WHERE PartitionKey = ;

SELECT * FROM table WHERE PartitionKey IN (,); //list of values

SELECT * FROM table WHERE PartitionKey = AND ClusteringKey1 = ;

SELECT * FROM table WHERE PartitionKey = AND Clusteringkey1 = AND
ClusteringKey2 = ;
```

Bad commands

`SELECT * FROM groups;` // noy okay performance wise

`SELECT * FROM groups where clustering key1 = ;` // will not work

`SELECT * FROM groups WHERE regular_column = ;` // will not work

ALWAYS START YOUR QUERIES WITH YOUR PARTITION KEYS

DELETE

- You can delete by record, cell, range and partition

```
DELETE FROM groups WHERE groupip=12 AND username = 'elaine@yahoo.com';

DELETE age FROM groups WHERE groupid=12 AND username = 'aland@gmail.com';

DELETE FROM groups where groupid=12;
```

- Delete operations in Cassandra ahve a great impact on performance
- Deleting data in distributed systems is trickier than in relation databases
- Especially in peer-to-peer ones like Cassandra
- Reads and writes can be directed to any of partition's replicas, so there's no primary node for a write/read
- Cassandra marks all deleted items with a "tombstone" containing the time of the delete operation
- Tombstones
  - A special value to indicate data ahs been deleted + time of delete
  - Tombstones prevent deleted data from being returned during reads
  - Tombstones are deleted at `gc_grace_seconds` during compaction process

Summary

- Cluster-level reads are only sent to replcias according to consistency settings
- Reads should follow the Primary Key columns order for best performance
- Delete operations can be done at record, cell, range, and partition levels

---

Examples:

Let us insert a row into the table movies.

On cqlsh run the below command.

```
INSERT into movies(
movie_id, movie_name, year_of_release)
VALUES (1,'Toy Story',1995);
```

Verify that the data is saved.

`select * from movies;`

In the previous exercise you have inserted some data into the table named movies.
Let us query the data in the movies table.

```
select movie_name from movies where movie_id = 1;
```

The movie_id for Scream is 4. It was released in 1996 and not 1995.
Here is how you modify it.

```
UPDATE movies
SET year_of_release = 1996
WHERE movie_id = 4;
```

Delete the movie with the movie_id 5.

```
DELETE from movies
WHERE movie_id = 5;
```