CASSANDRA

Cassandra = Dynamo + Bigtable

Data Model

- Same as Bigtable
- Composite Columns
  - Secondary indexes

26

26

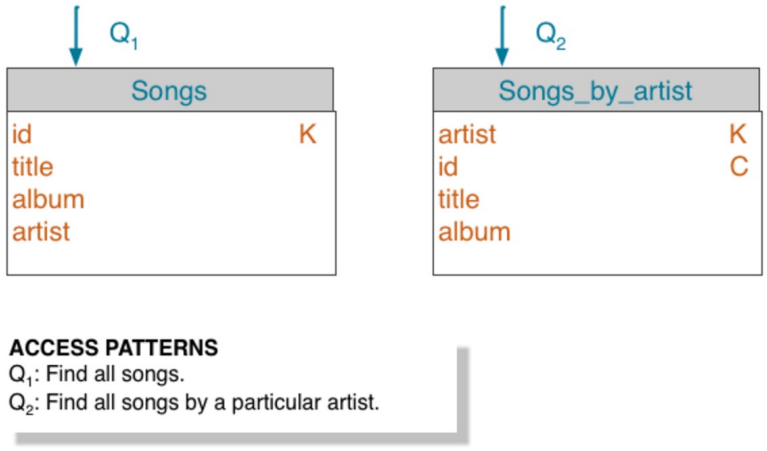
Data Modeling in Cassandra

- "Data modeling in Cassandra uses a *query-driven approach*, in which specific queries are the key to organizing the data. Cassandra's database design is based on the requirement for fast reads and writes, so the better the schema design, the faster data is written and retrieved. Queries are the result of selecting data from a table; schema is the definition of how data in the table is arranged."

27

27

Data Modeling in Cassandra



28

28

Data Modeling in Cassandra

- "Notice that the key to designing the table is not the relationship of the table to other tables, as it is in relational database modeling. Data in Cassandra is often arranged as one query per table, and data is repeated amongst many tables, a process known as *denormalization*. The relationship of the entities is important, because the order in which data is stored in Cassandra can greatly affect the ease and speed of data retrieval."

29

29

30

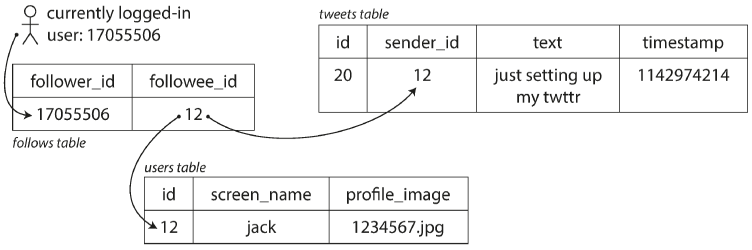
30

Twitter Operations

- Operations:
  - new tweet (4.6K/sec, 12K/sec peak)
  - timeline (300K/sec)
- Problem: Fan-Out

31

Twitter: Relational Database Approach

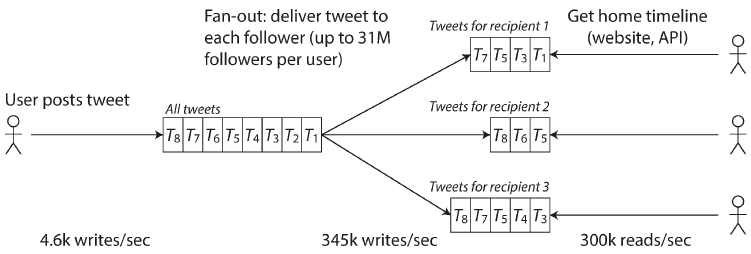



```
SELECT tweets.*, users.* FROM tweets
  JOIN users   ON tweets.sender_id    = users.id
  JOIN follows ON follows.followee_id = users.id
  WHERE follows.follower_id = current_user
```

32

Twitter: NoSQL Database Approach

- Push tweet timeline for each follower (75 avge)
- Faster reads in exchange for more writing



33

Composite Column

```
CREATE TABLE tweets (
  tweet_id uuid PRIMARY KEY,
  author varchar,
  body varchar
 );

CREATE TABLE timeline (
  user_id varchar,
  tweet_id uuid,
  author varchar,
  body varchar,
  PRIMARY KEY (user_id, tweet_id)
);
```

Partition Key

Tweets Table

tweet_id	author	body
1742	gwashington	I chopped down the cherry tree
1765	phenry	Give me liberty or give me death
1778	jadams	A government of laws, not men

Denormalized!

Timeline Table

Partition key | Remaining key

user_id	tweet_id	author	body
gmason	1765	phenry	Give me liberty or give me death
gmason	1742	gwashington	I chopped down the cherry tree
ahamilton	1797	jadams	A government of laws, not men
ahamilton	1742	gwashington	I chopped down the cherry tree

Timeline Physical Layout

Clustered by tweet_id

gmason	[1765, author]: phenry	[1765, body]: Give me liberty or give me death	[1742, author]: gwashington	[1742, body]: I chopped down the cherry tree
ahamilton	[1797, author]: jadams	[1797, body]: A government of laws not men	[1742, author]: gwashington	[1742, body]: I chopped down the cherry tree

Playlist Example

```
CREATE TABLE songs (
  id uuid PRIMARY KEY,
  title text,
  album text,
  artist text,
  data blob
 );

CREATE TABLE playlists (
  id uuid,
  song_order int,
  song_id uuid,
  title text,
  album text,
  artist text,
  PRIMARY KEY  (id, song_order ) );
```

36

36

Queries

- Query all playlists

```
SELECT * FROM playlists
```

- Filter on secondary column (index required)

```
CREATE INDEX ON playlists( artist );
SELECT album, title FROM playlists
   WHERE artist = 'Fu Manchu';
```

- Use clustering from composite primary key

```
SELECT * FROM playlists WHERE id =
62c36092-82a1-3a00-93d1-46196ee77204
   ORDER BY song_order DESC LIMIT 50;
```

37

37

Collection Columns

- Represent one-to-many relationships

- Set
- List
- Map

38

38

Sets

```
ALTER TABLE playlists ADD tags set<text>;

UPDATE playlists SET tags = tags + {'2007'}
  WHERE id =
     62c36092-82a1-3a00-93d1-46196ee77204
  AND song_order = 2;

SELECT album, tags FROM playlists
  WHERE tags CONTAINS 'blues';
```

39

39

Lists

```
ALTER TABLE playlists ADD reviews list<text>;

UPDATE playlists
  SET reviews = reviews + [ 'best lyrics' ]
  WHERE id =
     62c36092-82a1-3a00-93d1-46196ee77204
  AND song_order = 4;
```

40

40

Maps

```
ALTER TABLE playlists
  ADD venue map<timestamp, text>;

INSERT INTO playlists (id, song_order, venue)
  VALUES
    (62c36092-82a1-3a00-93d1-46196ee77204, 4,
     { '2013-9-22 22:00'  : 'The Fillmore',
       '2013-10-1 21:00' : 'The Apple Barrel'});

SELECT artist, venue FROM playlists WHERE venue
CONTAINS 'The Fillmore';
SELECT album, venue FROM playlists WHERE venue
CONTAINS KEY '2013-09-22 22:00:00-0700';
```

41

41

Collections vs Composite Columns

- Collections
  - For small amounts of data
    - Telephone numbers, tags, etc
  - Limited to 64K
- Composite Columns
  - For unlimited growth potential
  - Use compound primary key

42

42

User-Defined Types

```
CREATE KEYSPACE mykeyspace WITH REPLICATION = {
  'class' : 'NetworkTopologyStrategy',
  'datacenter1' : 1 };

CREATE TYPE mykeyspace.address (
  street text,
  city text,
  zip_code int,
  phones set<text>
);

CREATE TYPE mykeyspace.fullname (
  firstname text,
  lastname text
);
```

43

43

User-Defined Types

```
CREATE TABLE mykeyspace.users (

  id uuid PRIMARY KEY,

  name frozen <fullname>,

  direct_reports set<frozen <fullname>>,
      // a collection set

  addresses map<text, frozen <address>>
      // a collection map
);
```

44

44

User-Defined Types

```
INSERT INTO mykeyspace.users (id, name)
VALUES (
  62c36092-82a1-3a00-93d1-46196ee77204,
  {firstname: 'Marie-Claude',
   lastname: 'Josset'});

UPDATE mykeyspace.users
  SET addresses = addresses +
  {'home': { street: '191 Rue St. Charles',
             city: 'Paris',
             zip_code: 75015,
             phones: {'33 6 78 90 12 34'}}}
  WHERE id=62c36092-82a1-3a00-93d1-46196ee77204;
```

45

45

User-Defined Types

```
SELECT name FROM mykeyspace.users
  WHERE
  id=62c36092-82a1-3a00-93d1-46196ee77204;

SELECT name.lastname FROM mykeyspace.users
  WHERE
  id=62c36092-82a1-3a00-93d1-46196ee77204;
```


46