partition key

mechanism for determining where a row will be stored

May be composed of multiple columns

partition key is used to determine the token used by cassandra to store and retrieve data by in the cluster

primary key

mechanism for uniquely identifying a row of data

First column must always be the partition key

Simple primary keys are limited to one row per partition

Composite primary keys may have multiple rows per partition

Where partition key contains multiple columns as part of defining a primary key, surround partition key columns in parentheses.

If primary key has multi column partition key and no clustering keys, the primary key ends up being surrounded by double parentheses

create table foo (

id varchar,

name varchar,

primary key ((id, name))

);

Filtering

Using a clustering key to limit the number of rows returned from a query.

Has a performance penalty where partition column isn’t also specified as a restricting variable

How are primary keys specified?

Using the partition key

Using the partition key plus additional columns

composite primary keys

Primary key containing the partition key plus one or more clustering keys

Allows multiple rows to be stored together in a single partition

static columns

Static columns

Useful where a clustering key is specified

create table foo (

id varchar,

name int STATIC,

module int.

primary key (id, name)

)

To insert the static data in a partition

insert into foo (id, name) values (‘nodejs’, ‘Intro to NodeJS’);

If only partition-level data is entered, the static columns will return null if queried for.

To insert clustered data, just reference the partition key, then the clustered data. You don’t need to re-enter the static data.

insert into foo (id, module) values (‘nodejs’, 1);

insert into foo (id, module) values (‘nodejs’, 2);

If static level data is subsequently changed, it will be represented in all queries for that partition, regardless of the row

if you select just static data from a partition, you’ll get duplicate data, one row for each row in the partition.

To limit duplicates, use distinct

select distinct id, name from foo;

Static columns are also useful if you want to store latest info about a cluster across all rows

## Time series data

Examples:

click-stream data from a website

Data from sensors in an industrial setting

Commit logs

TimeUUID datatype

Version 1 UUID consisting of

number of 100ns intervals since UUID epoch

MAC address of machine generating UUID

clock seqno designed to prevent dups

Guaranteed unique and sortable by date and time embedded within

It can be useful to include timeuuid as a clustering key

Clustering keys can be referenced in the table definition to indicate an order in which data should be stored in the partition

create table course\_page\_views (

course\_id varchar,

view\_id timeuuid,

last\_viewed timeuuid static,

primary key (course\_id, view\_id)

) with clustering order by (view\_id desc);

Data can still be retrieved in ascending view\_id order…

select \* from source\_page\_views where course\_id = ‘foo’ order by view\_id asc;

Timeuuid functions

now()

returns timeuuid value representing the current date and time

useful in inserts

dateOf()

toDate()

toTimestamp

unixTimestampOf()

Selections are supported based on range of TimeUUID values

maxTimeuuid()

minTimeuuid()

select \* from foo where pk = ‘123’ and view\_id >= maxTimeuuid(‘2016-01-01 00:00+0000’)

and view\_id <= minTimeuuid(‘2016-02-01’);

Clustering key benefits

A benefit of having a Timeuuid as a clustering key is that it provides a guarantee that writes won’t clobber other writes as part of write-repair or hinted-handoff

Data is physically ordered on disk based on clustering keys, regardless of when they arrive

Delayed writes will be inserted according to the semantics of the clustering key

Great if you need inserts to be in order

TimeUUID’s can be specified manually, for example, when a client must generate it and have a handle to the inserted value.

Within a cluster ordered by time, you can retrieve the latest value simply by limiting to 1

select last\_viewed from course\_page\_views where course\_id = ‘123’ LIMIT 1

To retrieve the latest across partitions

## Bucketting Time Series Data

A Cassandra partition can only store 2 billion cells in a partition

Cell is a column in a row

All data in a partition must fit on a single cluster node

Bucketing is useful where you can’t use a TTL to minimize the amount of data stored in a partition

Involves specifying a partition key containing time values, for example, year and month.

This forces a TTL on each partition physically.

Ensures a single partition will never hold more than a single month

The challenge of this is that it may force you to query across two partitions if you require data from two months.

Try to understand the anticipated query requirements so that any IN clauses don’t span too many partitions, possibly causing a bottleneck.

Spanning