Indexing tables

An index provides a means to access data using attributes other than the partition key for fast, efficient lookup of data matching a given condition. The index indexes column values in a separate, hidden table from the one that contains the values being indexed. A number of techniques exist for guarding against the undesirable scenario where data might be incorrectly retrieved during a query involving indexes on the basis of stale values in the index.

Indexes can be used for collections, collection columns, static columns, and any other columns except counter columns.

When to use an index

Using a secondary index

Create indexes on a column after defining a table. Secondary indexes are used to query a table using a column that is not normally query-able.

```
use cycling;
CREATE TABLE rank_by_year_and_name (
  race_year int,
  race_name text,
  cyclist_name text,
  rank int,
  PRIMARY KEY ((race_year, race_name), rank)
);

INSERT INTO rank_by_year_and_name (race_year, race_name, cyclist_name, rank) VALUES
(2000, '1', 'A', 1);
```

Both race_year and race_name must be specified as these columns comprise the partition key.

```
SELECT * FROM rank_by_year_and_name WHERE race_year=2000 AND race_name='1';

 race_year | race_name | rank | cyclist_name
-----------+-----------+------+--------------
      2000 |         1 |    1 |            A

(1 rows)
```

A logical query to try is a listing of the rankings for a particular year. Because the table has a composite partition key, this query will fail if only the first column is used in the conditional operator.

```
SELECT * FROM rank_by_year_and_name WHERE race_year=2000;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"

SELECT * FROM rank_by_year_and_name WHERE race_name='1';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"
```

An index is created for the race year, and the query will succeed.

```
CREATE INDEX ryear ON rank_by_year_and_name (race_year);

SELECT * FROM rank_by_year_and_name WHERE race_year=2000;

 race_year | race_name | rank | cyclist_name
-----------+-----------+------+--------------
      2000 |         1 |    1 |            A

(1 rows)
```

A clustering column can also be used to create an index. An index is created on rank, and used in a query.

```
CREATE INDEX rrank ON rank_by_year_and_name (rank);

SELECT * FROM rank_by_year_and_name WHERE rank = 1;

 race_year | race_name | rank | cyclist_name
-----------+-----------+------+--------------
      2000 |         1 |    1 |            A

(1 rows)
```

Using multiple indexes

Indexes can be created on multiple columns and used in queries. The general rule about cardinality applies to all columns indexed. In a real-world situation, certain columns might not be good choices, depending on their cardinality.

```
CREATE TABLE cyclist_alt_stats ( id UUID PRIMARY KEY, lastname text, birthday timest
amp, nationality text, weight text, height text );

CREATE INDEX birthday_idx ON cycling.cyclist_alt_stats ( birthday );
CREATE INDEX nationality_idx ON cycling.cyclist_alt_stats ( nationality );
```

```
INSERT INTO cyclist_alt_stats (id, lastname, birthday, nationality, weight, height)
VALUES (41d01b63-244e-435d-bd6d-5dc8a0addb8c, 'J', '1982-01-29', 'Russia', '80', '18
0');

SELECT * FROM cyclist_alt_stats WHERE birthday = '1982-01-29' AND nationality = 'Rus
sia';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute
 this query as it might involve data filtering and thus may have unpredictable perfo
rmance. If you want to execute this query despite the performance unpredictability,
use ALLOW FILTERING"
```

The indexes have been created on appropriate low cardinality columns, but the query still fails. Why? The answer lies with the partition key, which has not been defined. When you attempt a potentially expensive query, such as searching a range of rows, the database requires the ALLOW FILTERING directive. The error is not due to multiple indexes, but the lack of a partition key definition in the query.

```
SELECT * FROM cyclist_alt_stats WHERE birthday = '1982-01-29' AND nationality = 'Rus
sia' ALLOW FILTERING;

 id                                   | birthday                        | height | l
astname | nationality | weight
--------------------------------------+---------------------------------+--------+--
--------+-------------+--------
 41d01b63-244e-435d-bd6d-5dc8a0addb8c | 1982-01-29 00:00:00.000000+0000 |    180 |
      J |      Russia |     80

(1 rows)
```

Indexing a collection