



**DataStax Astra** — Built on Apache Cassandra and designed from the ground up to run anywhere, on any cloud, in any datacenter, and in every possible combination. DataStax delivers the ultimate hybrid and multi-cloud database.

Search



Search other guides

## Using CQL

Starting cqlsh on Linux and Mac OS X

&gt; Creating and updating a keyspace

&gt; Creating a table

&gt; Creating advanced data types in tables

&gt; Creating functions

&gt; Inserting and updating data

&gt; Batching data insertion and updates

### Querying tables

#### Retrieval and sorting results

Retrieval using collections

Retrieval using JSON

Retrieval using the IN keyword

Retrieval by scanning a partition

Retrieval using standard aggregate functions

Retrieval using a user-defined function (UDF)

Retrieval using user-defined aggregate (UDA) functions

&gt; Querying a system table

&gt; Indexing

&gt; Altering a table

Altering a user-defined type

&gt; Removing a keyspace, schema, or data

&gt; Securing a table

&gt; Tracing consistency changes

Displaying rows from an unordered partitioner with the TOKEN function

Determining time-to-live (TTL) for a column

Determining the date/time of a write

&gt; Legacy tables

## Retrieval and sorting results

Querying tables to select data is the reason data is stored in databases. Similar to SQL, CQL can SELECT data using simple or complex qualifiers. At its simplest, a query selects all data in a table. At its most complex, a query delineates which data to retrieve and display and even calculate new values based on user-defined functions.

### Setting up the example table

Create a table that will sort data into more than one partition.

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

Insert the data:

```
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2015, 'Tour of Japan - Stage 4 - Minami > Shinshu', 'Benjamin PRADES', 1);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2015, 'Tour of Japan - Stage 4 - Minami > Shinshu', 'Adam PHELAN', 2);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2015, 'Tour of Japan - Stage 4 - Minami > Shinshu', 'Thomas LEBAS', 3);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2015, 'Giro d'Italia - Stage 11 - Forli > Imola', 'Ilnur ZAKARIN', 1);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2015, 'Giro d'Italia - Stage 11 - Forli > Imola', 'Carlos BETANCUR', 2);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2014, '4th Tour of Beijing', 'Phillippe GILBERT', 1);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2014, '4th Tour of Beijing', 'Daniel MARTIN', 2);
INSERT INTO cycling.rank_by_year_and_name (race_year, race_name, cyclist_name, rank)
VALUES (2014, '4th Tour of Beijing', 'Johan Esteban CHAVES', 3);
```

### Procedure

- Use a simple SELECT query to display all data from a table.

```
cqlsh> SELECT * FROM cycling.cyclist_category;
```

race_year	race_name	rank	cyclist_name
2014	4th Tour of Beijing	1	Phillippe GILBERT
2014	4th Tour of Beijing	2	Daniel MARTIN
2014	4th Tour of Beijing	3	Johan Esteban CHAVES

2015   Giro d'Italia - Stage 11 - Forli > Imola   1   Ilnur ZAKARIN
2015   Giro d'Italia - Stage 11 - Forli > Imola   2   Carlos BETANCUR
2015   Tour of Japan - Stage 4 - Minami > Shinshu   1   Benjamin PRADES
2015   Tour of Japan - Stage 4 - Minami > Shinshu   2   Adam PHELAN
2015   Tour of Japan - Stage 4 - Minami > Shinshu   3   Thomas LEVAS

- The example below illustrates how to create a query that uses `category` as a filter.

```
cqlsh> SELECT * FROM cycling.cyclist_category WHERE category = 'SPRINT';
```

category	id	lastname	points
SPRINT	1	TERRI	34
SPRINT	2	JIM	120

Note that Cassandra will reject this query if `category` is not a partition key or clustering column. Queries require a sequential retrieval across the entire `cyclist_category` table. In a distributed database like Cassandra, this is a crucial concept to grasp; scanning all data across all nodes is prohibitively slow and thus blocked from execution. The use of partition key and clustering columns in a `WHERE` clause must result in the selection of a contiguous set of rows.

A query based on `lastname` can result in satisfactory results if the `lastname` column is indexed.

- You can also pick the columns to display instead of choosing all data.

```
cqlsh> SELECT category, points, lastname FROM cycling.cyclist_category;
```

category	points	lastname
SPRINT	34	TERRI
SPRINT	120	JIM
GC	1234	TERRI
GC	2234	JIM

- For a large table, limit the number of rows retrieved using `LIMIT`. The default limit is 10,000 rows. To sample data, pick a smaller number. To retrieve more than 10,000 rows set `LIMIT` to a large value.

```
cqlsh> SELECT * From cycling.cyclist_name LIMIT 3;
```

id	firstname	lastname
e7ae5cf3-d358-4d99-b900-85902fda9bb0	Alex	FRAME
5b6962dd-3f90-4c93-8f61-eabfa4a803e2	Marianne	VOS
220844bf-4860-49d6-9a4b-6b5d3a79cbfb	Paolo	TIRALONGO

- You can fine-tune the display order using the `ORDER BY` clause. The partition key must be defined in the `WHERE` clause and the `ORDER BY` clause defines the clustering column to use for ordering.

```
cqlsh> CREATE TABLE cycling.cyclist_cat_pts ( category text, points int, id UUID, lastname text, PRIMARY KEY (category, id);
```

```
cqlsh> SELECT * FROM cycling.cyclist_cat_pts WHERE category = 'GC' ORDER BY points ASC;
```

category	points	id	lastname
GC	780	829aa84a-4bba-411f-a4fb-38167a987cda	SUTHERLAND
GC	1269	220844bf-4860-49d6-9a4b-6b5d3a79cbfb	TIRALONGO

- Tuples are retrieved in their entirety. This example uses AS to change the header of the tuple name.

```
cqlsh> SELECT race_name, point_id, lat_long AS CITY_LATITUDE_LONGITUDE FROM cycling.route;
```

race_name	point_id	city_latitude_longitude
47th Tour du Pays de Vaud	1	('Onnens', (46.8444, 6.6667))
47th Tour du Pays de Vaud	2	('Champagne', (46.833, 6.65))
47th Tour du Pays de Vaud	3	('Novalle', (46.833, 6.6))
47th Tour du Pays de Vaud	4	('Vuiteboeuf', (46.8, 6.55))
47th Tour du Pays de Vaud	5	('Baulmes', (46.7833, 6.5333))
47th Tour du Pays de Vaud	6	('Les Clées', (46.7222, 6.5222))

Contact Us

+1 (650) 389-6000

info@datastax.com



© 2020 DataStax | Privacy policy | Terms of use | Updated: 28 February 2020

DataStax, Titan, and TitanDB are registered trademark of DataStax, Inc. and its subsidiaries in the United States and/or other countries.

Apache, Apache Cassandra, Cassandra, Apache Tomcat, Tomcat, Apache Lucene, Apache Solr, Apache Hadoop, Hadoop, Apache Spark, Spark, Apache TinkerPop, TinkerPop, Apache Kafka and Kafka are either registered trademarks or trademarks of the Apache Software Foundation or its subsidiaries in Canada, the United States and/or other countries.