Select syntax

Continuous Query

In summary, a CQL statement is like a SQL one, so the continuous query consists of a select, a from, a where, a goup and a having part.

We use the following example to explain basic details of CQL-Query.

```
SELECT auction, AVG(price) AS aprice
FROM bid [SIZE 60 MINUTES ADVANCE 1 MINUTE TIME]
WHERE auction > 10
GROUP BY auction
HAVING aprice<100.0
```

Select

```
SELECT auction, AVG(price) AS aprice...
```

From

```
... FROM bid [SIZE 60 MINUTES ADVANCE 1 MINUTE TIME]...
```

The most different parts between usual SQL and CQL is the FROM part, because you have the possibility to definie windows. CQL defines them by squared brackets.

There are 3 different windows you can define: an unbounded window, a time-based window and a tuple-based window.

An **unbounded window** is defined by the keyword UNBOUNDED. It sets the end timestamp to infinite. Example:

```
SELECT * FROM bid [UNBOUNDED]
```

Since an unbounded window does not limit the validity of a stream element (in fact it is not really a window), the declaration of UNBOUNDED is optional. You get the same result without a window declaration. You can use the UNBOUNDED keyword to highlight that no window is defined.

A **time-based window** is defined by the size of the window as time span and an optional advance parameter. The latter defines after what time span the window should move (if no unit is declared, it has the same unit as size). Additional, a partition attribute can be defined. Syntax:

```
SELECT * FROM <source> [SIZE <size> <unit> TIME]

SELECT * FROM <source> [SIZE <size> <unit> ADVANCE <advance size> TIME]

SELECT * FROM <source> [SIZE <size> <unit> ADVANCE <advance size> <unit> TIME]

SELECT * FROM <source> [SIZE <size> <unit> TIME PARTITION BY bid.auction]

SELECT * FROM <source> [SIZE <size> <unit> ADVANCE <advance size> TIME PARTITION BY <partition attribute>]

SELECT * FROM <source> [SIZE <size> <unit> ADVANCE <advance size> <unit> TIME PARTITION BY <partition
attribute>]
```

For valid values for <unit> see TimeWindow.

A **tuple-based window** is defined by the size of the window as number of tuples and an optional advance parameter. The latter defines after how many tuples the window should move. Additional, a partition attribute can be defined. Syntax:

```
SELECT * FROM <source> [SIZE <size> TUPLE]

SELECT * FROM <source> [SIZE <size> ADVANCE <advance size> TUPLE]

SELECT * FROM <source> [SIZE <size> TUPLE PARTITION BY bid.auction]

SELECT * FROM <source> [SIZE <size> ADVANCE <advance size> TUPLE PARTITION BY <partition attribute>]
```

Futher information about windows can be found here

Where

```
... WHERE auction > 10 ...
```

Group By and Having

```
... GROUP BY auction
HAVING aprice<100.0
```

Stream To

If you want to stream your results into a sink, you first have to create a sink.

```
STREAM TO writeout SELECT * FROM nexmark:person WHERE...
```

This example would push all data that is produced by "SELECT * FROM nexmark:person WHERE..." into the sink named writeout, which is a file-writer in our case (see above).

Examples

Here are some language examples what can be used in the select-part of a CQL-Statement

```
#PARSER CQL
#TRANSCFG Standard
#DOREWRITE false
#QUERY
DROP STREAM bid IF EXISTS
#QUERY
ATTACH STREAM bid (timestamp STARTTIMESTAMP, auction INTEGER, bidder INTEGER, datetime LONG, price DOUBLE)
CHANNEL localhost : 65442
#QUERY
DROP STREAM person IF EXISTS
#QUERY
ATTACH STREAM person (timestamp STARTTIMESTAMP,id INTEGER,name STRING,email STRING,creditcard STRING,city
STRING,state STRING) CHANNEL localhost : 65440
/// SIMPLE PROJECTS
#QUERY
SELECT * FROM bid
#QUERY
SELECT bid.* FROM bid
#QUERY
SELECT price FROM bid
#QUERY
SELECT bidder, price FROM bid
#QUERY
SELECT timestamp, auction, bidder, datetime, price FROM bid
```

```
/// PROJECTS WITH RENAMED SOURCE BUT NO USE IN PROJECT
#QUERY
SELECT * FROM bid AS b
#QUERY
SELECT bid.* FROM bid AS b
#QUERY
SELECT price FROM bid AS b
#QUERY
SELECT bidder, price FROM bid AS b
#QUERY
SELECT timestamp, auction, bidder, datetime, price FROM bid AS b
/// PROJECTS WITH RENAMED SOURCE WITH USE IN PROJECT
#QUERY
SELECT * FROM bid AS b
#QUERY
SELECT b.* FROM bid AS b
#QUERY
SELECT b.price FROM bid AS b
#QUERY
SELECT b.bidder, b.price FROM bid AS b
#QUERY
SELECT b.timestamp, b.auction, b.bidder, b.datetime, b.price FROM bid AS b
/// PROJECTS WITH RENAMED ATTRIBUTES
#QUERY
SELECT price AS p FROM bid
#QUERY
SELECT price AS p, bidder FROM bid
#QUERY
SELECT price AS p, bidder AS b FROM bid
/// PROJECTS WITH RENAMED ATTRIBUTES AND SOURCES
#QUERY
SELECT price AS p FROM bid AS b
#QUERY
SELECT b.price AS p FROM bid AS b
#QUERY
SELECT b.price AS p, b.bidder FROM bid AS b
#QUERY
SELECT b.price AS p, b.bidder AS b FROM bid AS b
/// PROJECTS WITH CONSTANTS, FUNCTIONS AND EXPRESSIONS
#QUERY
SELECT bidder + price AS d FROM bid
#QUERY
SELECT 123.4 * price AS d FROM bid
#QUERY
SELECT 123.4 AS d FROM bid
#QUERY
SELECT 123.4 AS d, price FROM bid
#QUERY
SELECT DolToEur(price) AS d FROM bid
#QUERY
SELECT DolToEur(price) AS d, price FROM bid
#QUERY
SELECT DolToEur(price) * auction AS d FROM bid
#QUERY
SELECT DolToEur(price) * price AS d FROM bid
#QUERY
SELECT DolToEur(price) * auction AS d, price FROM bid
#QUERY
SELECT DolToEur(123.4) AS d FROM bid
#QUERY
SELECT 'test' AS s FROM bid
#QUERY
SELECT 'test' AS s, 123.4 AS d FROM bid
#QUERY
SELECT 'test' AS s, 123.4 AS d, price FROM bid
/// PROJECTS AND SELECTS WITH RENAMED SOURCE WITH USE IN SELECT
#QUERY
SELECT * FROM bid AS b WHERE b.bidder > 10
#QUERY
SELECT b.* FROM bid AS b WHERE b.bidder > 10
#QUERY
```

```
SELECT b.price FROM bid AS b WHERE b.price < 150.0
#QUERY
SELECT b.bidder, b.price FROM bid AS b WHERE b.bidder =  1
#QUERY
SELECT b.timestamp, b.auction, b.bidder, b.datetime, b.price FROM bid AS b WHERE b.price > 100.0
/// PROJECTS WITH RENAMED ATTRIBUTES
#QUERY
SELECT price AS p FROM bid WHERE p < 100
#QUERY
SELECT price AS p, bidder FROM bid WHERE p > 100
#QUERY
SELECT price AS p, bidder AS b FROM bid WHERE b=1 AND p <100
/// AGGREGATES ARE NOT HANDLED LIKE FUNCTIONS AND MAY HAVE A GROUPING ETC.
#QUERY
SELECT AVG(price) AS aprice FROM bid
#QUERY
SELECT AVG(price) AS aprice FROM bid GROUP BY auction
#QUERY
SELECT auction, AVG(price) AS aprice FROM bid GROUP BY auction
#QUERY
SELECT auction, AVG(price) AS aprice FROM bid GROUP BY auction HAVING aprice<100.0
/// JOINS AND SELFJOINS
#QUERY
SELECT auction, bidder FROM bid, person WHERE bid.bidder = person.id
#QUERY
SELECT auction FROM bid AS b, person AS p WHERE b.bidder = p.id
#QUERY
SELECT auction AS a, bidder AS b FROM bid, person WHERE a = b
#QUERY
SELECT auction, bidder FROM bid, person WHERE bid.bidder = person.id
#QUERY
SELECT left.* FROM bid AS left, bid AS right WHERE left.bidder = right.bidder
#QUERY
SELECT a.auction AS aid FROM bid AS a, bid AS b WHERE aid=b.auction
```

And here are examples, based on nexmark, that are more complex

```
#PARSER CQL
#TRANSCFG Standard
#DROPALLQUERIES
#DROPALLSOURCES
#QUERY
CREATE STREAM nexmark:person (timestamp STARTTIMESTAMP, id INTEGER, name STRING, email STRING, creditcard
STRING, city STRING, state STRING) CHANNEL localhost : 65440
#QUERY
CREATE STREAM nexmark:bid (timestamp STARTTIMESTAMP, auction INTEGER, bidder INTEGER, datetime LONG, price
DOUBLE) CHANNEL localhost : 65442
#QUERY
CREATE STREAM nexmark:auction (timestamp STARTTIMESTAMP, id INTEGER, itemname STRING, description STRING,
initialbid INTEGER, reserve INTEGER, expires LONG, seller INTEGER, category INTEGER) CHANNEL localhost : 65441
#QUERY
CREATE STREAM nexmark:category (id INTEGER, name STRING, description STRING, parentid INTEGER) CHANNEL
localhost : 65443

#PARSER CQL
#TRANSCFG Standard
#DROPALLQUERIES
/// Query 1: Currency Conversion
#QNAME Nexmark:Q1
#ADDQUERY
SELECT auction, DolToEur(price) AS euro, bidder, datetime
FROM nexmark:bid [UNBOUNDED];

///Query 2: Selection
#QNAME Nexmark:Q2
#ADDQUERY
SELECT auction, price
FROM nexmark:bid
WHERE auction=7 OR auction=20 OR auction=21 OR auction=59 OR auction=87;
```

```
///Query 3: Local Item Suggestion
#QNAME Nexmark:Q3
#ADDQUERY
SELECT p.name, p.city, p.state, a.id
FROM nexmark:auction [UNBOUNDED] AS a, nexmark:person [UNBOUNDED] AS p
WHERE a.seller=p.id AND (p.state='Oregon' OR p.state='Idaho' OR p.state='California') AND a.category = 10;


///Query 4: Average Price for a Category
#QNAME Nexmark:Q4
#ADDQUERY
SELECT AVG(q.final)
FROM nexmark:category [UNBOUNDED] AS c,
    (SELECT MAX(b.price) AS final, a.category
     FROM nexmark:auction [UNBOUNDED] AS a, nexmark:bid [UNBOUNDED] AS b
     WHERE a.id = b.auction AND b.datetime < a.expires AND  a.expires < Now()
     GROUP BY a.id, a.category) AS q
WHERE q.category = c.id
GROUP BY c.id;

///Query 5: Hot Items
#QNAME Nexmark:Q5
#ADDQUERY
SELECT b2.auction
FROM   (SELECT b1.auction, COUNT(auction) AS num
        FROM nexmark:bid [SIZE 60 MINUTES ADVANCE 1 MINUTE TIME] AS b1
        GROUP BY b1.auction
        ) AS b2
WHERE num >= ALL   (SELECT count(auction) AS c
                    FROM nexmark:bid [SIZE 60 MINUTES ADVANCE 1 MINUTE TIME] AS b2
                    GROUP bY b2.auction)

///Query 6: Average Selling Price by Seller
#QNAME Nexmark:Q6
#ADDQUERY
SELECT AVG(Q.final) AS s, Q.seller
FROM (
SELECT MAX(B.price) AS final, A.seller
      FROM nexmark:auction [UNBOUNDED] AS A , nexmark:bid [UNBOUNDED]  AS B
      WHERE A.id=B.auction AND B.datetime < A.expires AND A.expires < ${NOW}
      GROUP BY A.id, A.seller) [SIZE 10 TUPLE PARTITION BY A.seller] AS Q
GROUP BY Q.seller;

///Query 7: Monitor New Users
#QNAME Nexmark:Q7
#ADDQUERY
SELECT p.id, p.name, a.reserve
FROM nexmark:person [SIZE 12 HOURS ADVANCE 1 TIME] AS p, nexmark:auction [SIZE 12 HOURS ADVANCE 1 TIME] AS a
WHERE p.id = a.seller;
```