
# Cassandra

Advanced Topics on NoSQL databases

A4 - S8

**ESILV**
nicolas.travers (at) devinci.fr

# Chapter 1

# Create a database on Cassandra

## 1.1 Keyspace

Before querying it, we need to create a database (Keyspace). To begin with:

```
CREATE KEYSPACE IF NOT EXISTS school
  WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor': 3 };
```

Here we create a "`school`" database for which the replication factor is set to 3, in order to manage falt tolerance. To select the database:

- in `cqlsh` mode / shell only:

```
USE school;
```

- In `DevCenter`, the combo box on top, you can choose "school".

## 1.2 Column Family

Then, we need to create "tables" which are called *Column Family* on Cassandra. Here is the schema definition:

```
CREATE TABLE IF NOT EXISTS Lesson (
  idLesson INT, Title VARCHAR, Responsible INT, Level VARCHAR, Quota INT, Coeff INT,
  PRIMARY KEY ( idLesson )
);
CREATE INDEX fk_Lesson_Responsible ON Lesson ( Responsible ) ;
```

```
CREATE TABLE IF NOT EXISTS Teacher (
  idTeacher INT, Lastname VARCHAR, Firstname VARCHAR, status VARCHAR,
  PRIMARY KEY ( idTeacher )
);
```

To verify the schemas:

- In cqlsh mode / shell:

```
DESC school;
```

- In `DevCenter` top-right corner, select your table, you will see your table's schema and indexes.

Then we can see the information from column families, types and storage information.

## 1.3 Import data

To finish with, we need to insert data. For this, use standard INSERT queries (can combine JSON):

```
INSERT INTO Lesson (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES (1,'Introduction to Databases',1,'M1',30,3);
INSERT INTO Lesson (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES (2,'Skylines',4,'M1',30,2);
INSERT INTO Lesson (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES (3,'Service production and distribution',5,'M1',30,2);
INSERT INTO Lesson (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES (4,'Advanced databases',1,'M2',30,5);
INSERT INTO Lesson (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES (5,'Systems Architecture',6,'M2',8,1);
INSERT INTO Lesson (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES (6,'IT Business / Introduction',7,'M2',20,3);
INSERT INTO Lesson (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES (7,'IT Business / Strategy and Management',8,'M2',10,1);
```

```
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (1,'Travers','Nicolas','Temporary');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (2,'Mourier','Pascale','Permanent');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (3,'Boisson','Francois','Temporary');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (4,'Mathieu','Eric','Permanent');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (5,'Chu','Chengbin','Permanent');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (6,'Boutin','Philippe','Permanent');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (7,'Escribe','Julien','Temporary');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (8,'Znaty','David','Temporary');
INSERT INTO Teacher (idTeacher,Lastname,Firstname,status) VALUES (9,'Abal-Kassim','Cheik Ahamed','Temporary');
```

To show column family's content:

```
SELECT * FROM Lesson;
```

# Chapter 2

# Querying Cassandra

New we have a database, we can query it. However, since we use a query language very close to SQL, we will try to use it in this way. This practice work will be a little bit disturbing. The goal is to show how to use Cassandra in the proper way, not in the way to use traditional databases.

## 2.1 CQL : Cassandra Query Language

The Cassandra query Language CQL is available here: `https://cassandra.apache.org/doc/latest/cql/dml.html#select`. It is inspired from SQL. In the following, express the following queries in CQL:

*2.1.1* List of lessons,
**Correction :**

```
SELECT * FROM Lesson;
```

*2.1.2* List of lessons titles,
**Correction :**

```
SELECT Title FROM Lesson;
```

*2.1.3* Family name of the $4^{th}$ teacher,
**Correction :**

```
SELECT lastname FROM Teacher WHERE idTeacher=4;
```

*2.1.4* Titles of lessons for which the teacher '1' is responsible for,
**Correction :**

```
SELECT Title FROM Lesson WHERE Responsible=1 ;
```

*2.1.5* Titles from lessons which the quota is over 30,
**Correction :**

```
SELECT Title FROM Lesson WHERE Quot>30 ;
```

Here is the given error code:`InvalidRequest: code=2200 [Invalid query] message="No secondary indexes on the restricted c`
No index has been provided on Quota. We cannot query it, otherwise it will broadcast all the nodes.

*2.1.6* Lessons' title for a level 'M1' and the responsible is '1',
Use 'ALLOW FILTERING' at the end of the query.
**Correction :** Since there is two distinct predicates in the query, Cassandra cannot insure an optimized evaluation. So, you need to force the execution with 'ALLOW FILTERING'. Otherwise:`InvalidRequest: code=2200 [Invalid query]`
Due to very few data, this query is efficient. It is not always the case.

```
SELECT Title FROM Lesson WHERE Responsible=1 AND Level = 'M1' ALLOW FILTERING;
```

*2.1.7* Lessons' title where the responsible id is below 5,
**Correction :** Since a hash is applied to the responsible id (on the index), the index itself does not know how to evaluate it efficiently. So a ALLOW FILTERING is necessary (and broadcast used).

```
SELECT Title FROM Lesson WHERE responsible < 5;
```

*2.1.8* Lessons' title where the token of identifiers are below 0. Use function 'TOKEN()' of rows' identifier.
**Correction :**

```
SELECT Title FROM Lesson WHERE token(idLesson) < 0;
```

The result itself is not really understandable. If fact, the token function is the resulting number where data with the value '0' must be placed on the network. So, there is no meaning to query it. It can be usefull to see if data can be placed on a same node, or data grouped together.

*2.1.9* Count the number of lessons where responsible is equal to '1',
**Correction :**

```
SELECT COUNT(*) FROM Lesson WHERE responsible=1;
```

## 2.2 Joins & Denormalization

Now, we wish to join tables together. We will try to see how to bring a solution.

*2.2.1* Give lessons' title given by a teacher which status is 'temporary',
**Correction :** Of course, we wish to do it in the standard way:

```
SELECT Title FROM Lesson, Teacher
WHERE Responsible = idTeacher and status='Temporary';

SELECT Title FROM Lesson
WHERE Responsible IN (SELECT idTeacher FROM Teacher WHERE status='Temporary');
```

But CQL do not allow it. See in the following how to answer it.

*2.2.2* There is no way to do a join un CQL (since it is a distributed database, a wide broadcast is necessary and to avoid absolutely). We will then denormalize schemas in order to provide a new table with data join together. But do we integrate Lessons in Teachers or the other way?
We will start with Teacher embedded in Lessons, in a column family called: LESSONTEACHER.

Three possibilities are available to do it: *SET, LIST, MAP, SubType*. We have to choose it in order to query the status of the teacher (Temporary).
**Correction :**

```
CREATE TABLE IF NOT EXISTS LessonTeacher (
        idLesson INT, Title VARCHAR,
        Responsible map<text, text>,
        Level VARCHAR, Quota INT, Coeff INT,
        PRIMARY KEY ( idLesson )
) ;
```

*2.2.3* You can then insert those rows:

```
INSERT INTO LessonTeacher (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES
 (1,'Introduction to Databases',{'idTeacher':'1','lastname':'Travers','firstname':'Nicolas','status':'Temporary'},'M1',30,3);
INSERT INTO LessonTeacher (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES
 (2,'Skylines',{'idTeacher':'4','lastname':'Mathieu','firstname':'Eric','status':'Permanent'},'M1',30,2);
INSERT INTO LessonTeacher (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES
 (3,'Service production and distribution',{'idTeacher':'5','lastname':'Chu','firstname':'Chengbin','status':'Permanent'},'M1',30,2);
INSERT INTO LessonTeacher (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES
 (4,'Advanced databases',{'idTeacher':'1','lastname':'Travers','firstname':'Nicolas','status':'Temporary'},'M2',30,5);
INSERT INTO LessonTeacher (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES
 (5,'Systems Architecture',{'idTeacher':'6','lastname':'Boutin','firstname':'Philippe','status':'Permanent'},'M2',8,1);
INSERT INTO LessonTeacher (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES
 (6,'IT Business / Introduction',{'idTeacher':'7','lastname':'Escribe','firstname':'Julien','status':'Temporary'},'M2',20,3);
INSERT INTO LessonTeacher (idLesson,Title,Responsible,Level,Quota,Coeff) VALUES
 (7,'IT Business / Strategy and Management',{'idTeacher':'8','lastname':'Znaty','firstname':'David','status':'Temporary'},'M2',10,1);
```

*2.2.4* Create an index on attribute 'Level' of column family LessonTeacher;
**Correction :**

```
CREATE INDEX IF NOT EXISTS LessonTeacher_Level ON LessonTeacher ( Level ) ;
```

*2.2.5* Give lastnames from Responsibles from lessons of 'M1' level,
**Correction :** The 'map' embedded structure is not adapted for projection. Thus we need to project all the 'responsible' attribute.

```
SELECT Responsible FROM LessonTeacher WHERE Level = 'M1';
```

*2.2.6* Give lessons of 'Temporary' lessons' responsible. Without 'ALLOW FILTERING'.
**Correction :**

```
//with allow filtering
 SELECT * FROM LessonTeacher WHERE Responsible['status'] = 'Temporary' ALLOW FILTERING;
SELECT Title FROM LessonTeacher WHERE Responsible CONTAINS 'Temporary' ALLOW FILTERING;

//index to avoid allow filtering
CREATE INDEX IF NOT EXISTS LessonTeacher_status ON LessonTeacher ( Responsible ) ;

//The only way to query an index on a map is CONTAINS
SELECT Title FROM LessonTeacher WHERE Responsible CONTAINS 'Temporary';

//In order to use the index and insure that the status is filtered out (in the cassandra server)
 SELECT * FROM LessonTeacher WHERE Responsible CONTAINS 'Temporary'
AND Responsible['status'] = 'Temporary' ALLOW FILTERING;
```

*2.2.7* Now, lets change nesting in the other way by embedding the lessons in the teacher (which is responsible of those lessons). In order to add a set of values (all the lessons), we can use: SET, MAP or LIST. Use the MAP structure, Create a column family 'TeacherLessons' with a sub type 'Lesson'.
**Correction :**

```
CREATE TYPE IF NOT EXISTS LessonType (idLesson INT, Title VARCHAR, Level VARCHAR, Quota INT, Coeff INT);

CREATE TABLE IF NOT EXISTS TeacherLessons (
        idTeacher INT, lastname VARCHAR, firstname VARCHAR, status VARCHAR,
        lessons map<int, frozen<LessonType>>,
        PRIMARY KEY ( idTeacher )
) ;
```

*2.2.8* Insert the corresponding data:

```
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (1,'Travers','Nicolas','Temporary',
 {1:{idLesson:1,Title:'Introduction to Databases',Level:'M1',Quota:30,Coeff:3},
 4:{idLesson:4,Title:'Advanced databases',Level:'M2',Quota:30,coeff:5}});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (2,'Mourier','Pascale','Permanent', {});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (3,'Boisson','Francois','Temporary', {});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (4,'Mathieu','Eric','Permanent',
 {4:{idLesson:2,Title:'Skylines',Level:'M1',Quota:30,coeff:2}});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (5,'Chu','Chengbin','Permanent', {});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (6,'Boutin','Philippe','Permanent',
 {5:{idLesson:5,Title:'Systems Architecture',Level:'M2',Quota:8,coeff:1}});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (7,'Escribe','Julien','Temporary',
 {6:{idLesson:6,Title:'IT Business / Introduction',Level:'M2',Quota:20,coeff:3}});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (8,'Znaty','David','Temporary',
 {7:{idLesson:7,Title:'IT Business / Strategy and Management',Level:'M2',Quota:10,coeff:1}});
INSERT INTO TeacherLessons (idTeacher,lastname,firstname,status,lessons) VALUES (9,'Abal-Kassim','Cheik Ahamed','Temporary', {});
```

*2.2.9* Create an index on the status of this new column family,
**Correction :**

```
CREATE INDEX IF NOT EXISTS TeacherLessons_status ON TeacherLessons ( status ) ;
```

*2.2.10* Give lessons' titles of temporary teachers,
**Correction :** The sub type allow to provide projection, but not map. So we need to project lessons. Ici non plus il n'est pas possible de projeter sur l'intitulé du (des) Lesson.

```
SELECT Lessons FROM TeacherLessons WHERE status = 'Temporary';
```

# Chapter 3

# Advanced features

## 3.1 Indexing

*3.1.1* The following query needs an 'ALLOW FILTERING' to be executed. Give a solution to handle it properly,

```
SELECT lastname, firstname FROM Teacher WHERE status='Temporary';
```

**Correction :** We simply need to create an index on 'status'.

```
CREATE INDEX Teacher_status ON Teacher ( status ) ;
```

*3.1.2* Create a new index on LESSON.LEVEL. Execute the following query and look at the produced *trace*[1]. Which index has been chosen for this query?

```
SELECT Title FROM Lesson WHERE Responsible=1 AND Level = 'M1' ALLOW FILTERING;
```

**Correction :** The long trace provides all the information of the execution. We can identify query (*parsing*), instanciation (*statement*), transaction, index choice (*Candidate indexes* - mean cardinality), execution, files (*sstables*), merge results...
The chosen index is: `Scanning with Lesson.fk_lesson_teacher` It is due to more selective index (if equal, the first one is chosen).

```
Execute CQL3 query,1477946885779000,172.17.0.2,COORDINATOR,0,
Parsing SELECT Title FROM Lesson WHERE Responsible=1 AND Level = 'M1' ALLOW FILTERING,1477946885779000,172.17.0.2,COORDINATOR,131,SharedPool-Worker-1
Preparing statement,1477946885779000,172.17.0.2,COORDINATOR,258,SharedPool-Worker-1
Computing ranges to query,1477946885780000,172.17.0.2,BOTH,903,SharedPool-Worker-1
Submitting range requests on 257 ranges with a concurrency of 257 (0.00234375 rows per range expected),1477946885780000,172.17.0.2,BOTH,1071,SharedPool-Worker-1
Submitted 1 concurrent range requests covering 257 ranges,1477946885804000,172.17.0.2,BOTH,25188,SharedPool-Worker-1
Executing indexed scan for (min(-9223372036854775808), min(-9223372036854775808)],1477946885805000,172.17.0.2,BOTH,26197,SharedPool-Worker-2
Candidate index mean cardinalities are CompositesIndexOnRegular{columnDefs=[ColumnDefinition{name=Responsible, type=org.apache.cassandra.db.marshal.Int32Type, kind=REGULAR, componentIndex=0, indexName=fk_l
Executing single-partition query on Lesson.fk_lesson_teacher,1477946885805000,172.17.0.2,BOTH,26400,SharedPool-Worker-2
Acquiring sstable references,1477946885805000,172.17.0.2,BOTH,26411,SharedPool-Worker-2
Merging memtable tombstones,1477946885805000,172.17.0.2,BOTH,26435,SharedPool-Worker-2
Partition index with 0 entries found for sstable 1,1477946885805000,172.17.0.2,BOTH,26609,SharedPool-Worker-2
Seeking to partition beginning in data file,1477946885805000,172.17.0.2,BOTH,26626,SharedPool-Worker-2
Skipped 0/1 non-slice-intersecting sstables, included 0 due to tombstones,1477946885805000,172.17.0.2,BOTH,26747,SharedPool-Worker-2
Merging data from memtables and 1 sstables,1477946885805000,172.17.0.2,BOTH,26761,SharedPool-Worker-2
Read 2 live and 0 tombstone cells,1477946885805000,172.17.0.2,BOTH,26788,SharedPool-Worker-2
Executing single-partition query on Lesson,1477946885806000,172.17.0.2,BOTH,26859,SharedPool-Worker-2
Acquiring sstable references,1477946885806000,172.17.0.2,BOTH,26871,SharedPool-Worker-2
Merging memtable tombstones,1477946885806000,172.17.0.2,BOTH,26887,SharedPool-Worker-2
Key cache hit for sstable 1,1477946885806000,172.17.0.2,BOTH,26916,SharedPool-Worker-2
Seeking to partition beginning in data file,1477946885806000,172.17.0.2,BOTH,26925,SharedPool-Worker-2
Skipped 0/1 non-slice-intersecting sstables, included 0 due to tombstones,1477946885806000,172.17.0.2,BOTH,26998,SharedPool-Worker-2
Merging data from memtables and 1 sstables,1477946885806000,172.17.0.2,BOTH,27010,SharedPool-Worker-2
Read 1 live and 0 tombstone cells,1477946885806000,172.17.0.2,BOTH,27036,SharedPool-Worker-2
Executing single-partition query on Lesson,1477946885806000,172.17.0.2,BOTH,27174,SharedPool-Worker-2
Acquiring sstable references,1477946885806000,172.17.0.2,BOTH,27186,SharedPool-Worker-2
Merging memtable tombstones,1477946885806000,172.17.0.2,BOTH,27199,SharedPool-Worker-2
Key cache hit for sstable 1,1477946885806000,172.17.0.2,BOTH,27222,SharedPool-Worker-2
Seeking to partition beginning in data file,1477946885806000,172.17.0.2,BOTH,27230,SharedPool-Worker-2
Skipped 0/1 non-slice-intersecting sstables, included 0 due to tombstones,1477946885806000,172.17.0.2,BOTH,27247,SharedPool-Worker-2
Merging data from memtables and 1 sstables,1477946885806000,172.17.0.2,BOTH,27256,SharedPool-Worker-2
Read 1 live and 0 tombstone cells,1477946885806000,172.17.0.2,BOTH,27286,SharedPool-Worker-2
Scanned 1 rows and matched 1,1477946885806000,172.17.0.2,BOTH,27317,SharedPool-Worker-2
Request complete,1477946885810041,172.17.0.2,COORDINATOR,31041,
```

*3.1.3* Suppose that the following query is the mostly used one on our database. We need to optimize it.

```
SELECT idLesson, Title FROM Lesson WHERE Responsible=X;
```

However there is only an index which leads to mutiple network queries (one for each Lesson of this teacher). Create a column family 'LESSON2' which will enhanced this query by querying a single node.
**Correction :** To acheive this, we need to change the clustering strategy which is done in PRIMARY KEY(idLesson).
For this:

[1] DevCenter : tab 'QUERY TRACE' next to the result tab - Shell/cqlsh : TRACING ON, before the query

(a) Add "Responsible" to the primary key
(b) Responsible becomes the group key
(c) Don't forget to create a unique index on idLesson for uniqueness

⇒

```
CREATE TABLE Lesson2 (
        idLesson INT, Title VARCHAR, Responsible INT,
        Quota INT, Coeff INT, level VARCHAR,
        PRIMARY KEY ( (Responsible), idLesson )
) ;
```

Take the tuples from Lesson and put them into the Lesson2 table.

*3.1.4* Execute the following query on both column families ("Lesson" and "Lesson2"). Look at the produced trace.

```
SELECT idLesson, Title FROM Lesson WHERE Responsible=1;
```

Correction : Lesson : **Executing read on ecole.cours using index fk_lesson_teacher**. Data are also kept from different files 'sstable'
Lesson2 : **Executing single-partition query on Lesson2**. Only one sstable is reached. There are also few time for this query.

## 3.2 Updates

*3.2.1* Update column family TeacherLessons by adding a new Lesson for teacher number '1'.

```
{'idLesson':10,'Title':'Cassandra','Level':'M2','coeff':1}
```

Correction :

```
UPDATE TeacherLessons SET Lessons = Lessons +
     {10 : {idLesson:10,title:'Cassandra',level:'M2',quota:10,coeff:1}}
    WHERE idTeacher = 1;
```

*3.2.2* Update column family TeacherLessons by replacing lesson '7' for teacher number '8' by this:

```
{'idLesson':7,'Title':'IT Business / Block Chain Management','Level':'M2','coeff':1}
```

Correction :

```
UPDATE TeacherLessons SET Lessons[7] =
     {idLesson:7,Title:'Cassandra',Level:'M2',Quota:10,coeff:1}
     WHERE idTeacher = 8;
```

*3.2.3* Delete lessons from teachers where the status is 'temporary'.
Correction :

```
DELETE Lesson from TeacherLessons WHERE status='Temporary';
```

This deleting query cannot work since the primary key is mandatory for each deletion.

## 3.3 User Define Aggregate functions

To agregate values from different rows, we need to create User Define Aggregate functions[2] (UDA). To achieve this, we need first to activate this functionnality with parameter '*enable_user_defined_functions*'.

For this:

- Edit file '`cassandra.yaml`' (config Cassandra folder)
- Find '`user_define`'
- Modify the parameter with '`true`' value (a space is mandatory between ':' and 'true')
- Save the config file
- restart the Cassandra server

Then, we can create the UDA:

*3.3.1* Create the *state* function:

```
CREATE OR REPLACE FUNCTION avgState ( state tuple<int,bigint>, val int )
  CALLED ON NULL INPUT RETURNS tuple<int,bigint> LANGUAGE java
  AS 'if (val !=null) { state.setInt(0, state.getInt(0)+1);
      state.setLong(1, state.getLong(1)+val.intValue()); }
      return state;';
```

*3.3.2* Create *final* function:

```
CREATE OR REPLACE FUNCTION avgFinal ( state tuple<int,bigint> )
  CALLED ON NULL INPUT RETURNS double LANGUAGE java
  AS 'double r = 0;
      if (state.getInt(0) == 0) return null;
      r = state.getLong(1);
      r/= state.getInt(0);
      return Double.valueOf(r);';
```

*3.3.3* Create the *UDA* function

```
CREATE AGGREGATE IF NOT EXISTS average ( int )
  SFUNC avgState STYPE tuple<int,bigint>
  FINALFUNC avgFinal INITCOND (0,0);
```

*3.3.4* Compute the average *Quota* on 'LessonTeacher' column family of level 'M1',
**Correction :**

```
SELECT average(Quota) FROM LessonTeacher WHERE Level='M1'
```

*3.3.5* Idem with 'Temporary' reponsibles,
**Correction :**

```
SELECT average(Quota) FROM LessonTeacher WHERE Responsible CONTAINS 'Temporary';
```

### 3.3.1 Bonus

*3.3.1* Create a new UDA to produce an equivalence to "*GROUP BY + COUNT*" on textual attributes, like for:

```
SELECT countGroup(level) from LessonTeacher;
```

[2] According to your report on Cassandra, this feature has to be used for `HARD` queries

The type of the 'state' parameter must be '*map<text, int>*'.

Correction :

```
CREATE OR REPLACE FUNCTION countGroupState ( state map<text, int>, val1 text)
        CALLED ON NULL INPUT RETURNS map<text, int> LANGUAGE java
        AS 'Integer count = (Integer)state.get(val1);
                if(count == null)
                        count = 0;
                count ++;
                state.put(val1, count);
                return state;';

CREATE OR REPLACE AGGREGATE countGroup ( text)
        SFUNC countGroupState STYPE map<text, int>
        INITCOND {};
```