6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

The abilities and some possible extensions of the Continuous Query Language

Antal Buza

Institute of Informatics, College of Dunaujvaros
e-mail: buza@mail.poliod.hu

Abstract

The CQL, Continuous Query Language is an expressive SQL-based declarative language for registering continuous queries against streams and updatable relations. CQL is suitable for data stream queries. There are situations when the queries operate on relational databases *and* on the data streams simultaneously. The execution of CQL query takes a long time (it may be several hours, days or even more). It is not unambiguous which semantic is suitable for the user when the database is updated during the execution of CQL query. For example, when we wish keep an eye on changing the value of an account while the official rates are updated, then CQL system must calculate with the retroactive effect of this update. Another semantic is reasoned when the prices are changed while we use CQL query for observing the trade of a supermarket. In this case the effect of the update is valid from the moment of update. In this paper we give a short description of CQL, characterisation of update-problems, and we give possible suggestions for the semantically extension of CQL.
Additional interesting question is the explanation of the consistent state. In classical database theory it is an usually requirement that the "normal" state of a database is the consistent state. The consistency is not a permanent state, during the updates it may be damaged for a short time. Investigations were performed to find out what the effect of the inconsistent state on the currently operating CQL queries is.

Categories and Subject Descriptors: H.2.3 [Database Management]: Languages – query languages; H.2.4 [Database Management]:Systems – query processing

Key Words and Phrases: Database, data stream, query language, continuous query

1. Introduction

In the section 2 we can find a short overview about Continuous Query Language. The section 2 is based on [3] and [6]. In the section 3 we focus on the suggested extensions of CQL. The extensions follow the effects of updates of databases realised during the long execution of CQL queries. According to the real situations the effects of the updates of databases are categorised in three types: the "retroactive effect", "from the update", and the "strong" effect of update.

2. Introduction the continuous queries and the CQL

In this section we would like shortly introduce to the readers the continuous queries and the Continuous Query Language i.e. CQL. We hope that this section is useful for readers who are not familiar to CQL before. This section is based on [3] and [6]. If you have knowledge about CQL you can jump to the section 3.

2.1. The continuous query

We consider that the *continuous query* which is issued once and then logically run continuously over the database (in contrast to traditional *one-time* queries which are run once to completion over the current data sets) and/or over the data stream.

2.2. CQL - the Continuous Query Language

CQL is an expressive SQL-based declarative language for registering continuous queries against streams and databases.

2.3. Illustrative CQL examples

Instead of the full description of CQL we cite some illustrative examples to demonstrate the abilities of CQL. Consider the domain of network traffic management for a large network. The network traffic management applications process typically rapid, unpredictable and continuous data streams. In the following examples we observe the traffic generated streams PT_c and PT_b (packet traces collected from the costumer and backbone links, respectively). For simplicity, we assume that the packet header comprises the fields: *saddr* – IP address of packet sender, *daddr* – IP address of packet destination, *id* – packet identification number, *length* – length of the packet, *timestamp* – time when the packet header was recorded.

The first CQL example computes the load on the backbone link averaged over one minute periods and notifies the network operator if the load exceeds a threshold T.

```
SELECT notifyoperator(sum(length)) FROM PT_b
       GROUP BY getminute(timestamp)
       HAVING sum(length)> T
```

In the example the `notifyoperator` and the `getminute` are self-explanatory functions. Similar functionality might be achievable using triggers in conventional DBMS, but something conventional triggers are certainly not designed for.
The next example illustrates the finding of the fraction of traffic on the backbone link coming from the customer network.

```
(SELECT count(*) FROM PT_c AS C, PT_b AS B
       WHERE C.saddr=B.saddr AND C.daddr=B.daddr
              AND C.id=B.id)
/
(SELECT count(*) FROM PT_b)
```

This is an example of an ad-hoc continuous query. (Since unbounded intermediate storage could potentially be required for joining two continuous data streams, we must use some kind of restrictions: time window or other, because of the answer might be an approximate answer only.)
The third example monitors the top 5% source-to-destination pairs in terms of traffic on the backbone link:

```
WITH load AS
       (SELECT saddr, daddr, SUM(length) AS traffic
         FROM PT_b
         GROUP BY saddr, daddr)
       SELECT saddr, daddr, traffic
         FROM load AS L_1
         WHERE (SELECT COUNT(*) FROM load AS L_2
                  WHERE L_2.traffic < L_1.traffic) >
                  (SELECT 0.95*COUNT(*) FROM load)
         ORDER BY traffic
```

As it was written in the abstract the continuous queries operate either on relational databases or on the data streams separated or on both simultaneously.

3. The update of databases produces different effects on answer strategy of the CQL queries

In this section we present some different situations produced by update of databases and their needed effects on the answer strategy of the CQL queries. We give

some suggested expansions of CQL for producing the useful answer according to different situations.

Remind that the continuous query runs continuously for a long time. During this long running time the database might be updated. How must the CQL query for the event of update reflect?

3.1. The retroactive effect of the update

Let us see an example in the bank environment. We store the accounts in the relation (RDBMS), several updates come through the stream(s) and we store the rates in the relation "*rates*". The rates may be changed during the execution of the CQL query. In this case, when we use a continuous query for the actual sum of the accounts in € (or in $, or in any common value), we can calculate with the retroactive effect of the update of rates relation. For example, we have an account 400£, and 1£=N€, when the rate is changing, (the new rate is 1£=M€), as a consequence the value of our account expressed via € will change too, of course.

Figure 1. illustrates a situation like this. We store the accounts in relation `ACC`, the rates are stored in relation `RAT`, the update records of the `ACC` relation comes through the `UPDS` data stream. The continuous query was started at time t_0, and the relation `RAT` was updated at time t_{upd}.

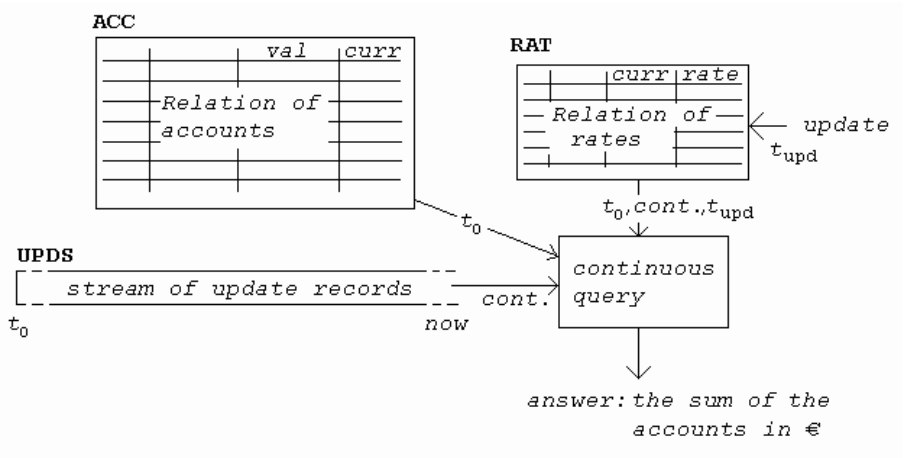

Figure 1: The effect of the update of `RAT` relation is retroactive for the answer of CQL query

The suitable suggested extension of the CQL query is as follows: this query summarizes the actual accounts (stored in `ACC` relation) expressed via Euro. The query is sensitive to the update of the rates (`RAT`) relation.

```
SELECT SUM(value*rate)
        FROM (STREAM(ACC) CONTINUE UPDS)
                NATURAL JOIN
                UPDATE_RETRO(RAT)
```

In the example keyword `STREAM` produces stream from relation `ACC`, keyword `CONTINUE` is an „ordered" `UNION` (e.g. in clausal `FROM` it is one stream, first part of the stream consists of the rows of `ACC` relation at time t_0, followed by the stream records coming through the `UPDS` data stream). Keyword `UPDATE_RETRO` indicates that when the relation `RAT` is updated the continuous query will be virtually restarted (e.g. it rereads all relations and processes the stream from now). The virtually restarted state of continuous query is illustrated in Figure 2.

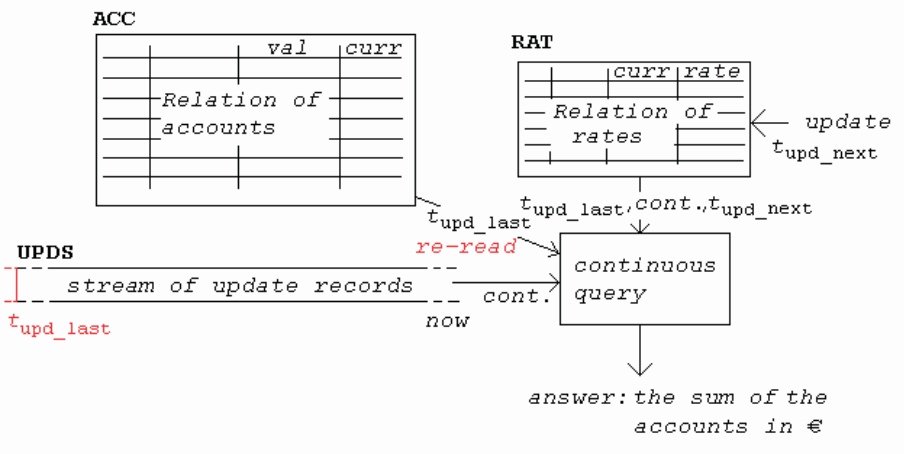

Figure 2: The virtually restart of the `UPD_RETRO` type CQL query

In Figure 2. the t_{upd_last} symbolizes the last update time of the `RAT` relation, the t_{upd_next} symbolizes the next update time of the `RAT` relation (in the future), $t_{upd\_last} < \text{now} < t_{upd\_next}$. The virtually restart means that the query in the moment of time t_{upd_last} rereads the relation `ACC` and it uses only the new part (produced after moment of time t_{upd_last}) of the `UPDS` stream. The query reads the `RAT` relation logically permanently. This is the theoretical operation of the continuous query. In the practice when for example we have enough memory for the store of the relation `RAT`, then the query don't read continuously or repeatedly this relation, one does only when the relation was updated.

3.2. The effect of the update "from now"

Another reasoned semantic of update is when the update produces an effect only from the actual time (from the moment of the update) and in the future of

course. For example in the trade systems when the prices are changed and we use the continuous query for determining the actual (daily, weekly, monthly,...) income, we can calculate the effect of update only from now (from the moment of update) and in the future (till the next update).

Let us see an example for cases like this. Figure 3. shows the environment of the continuous query.

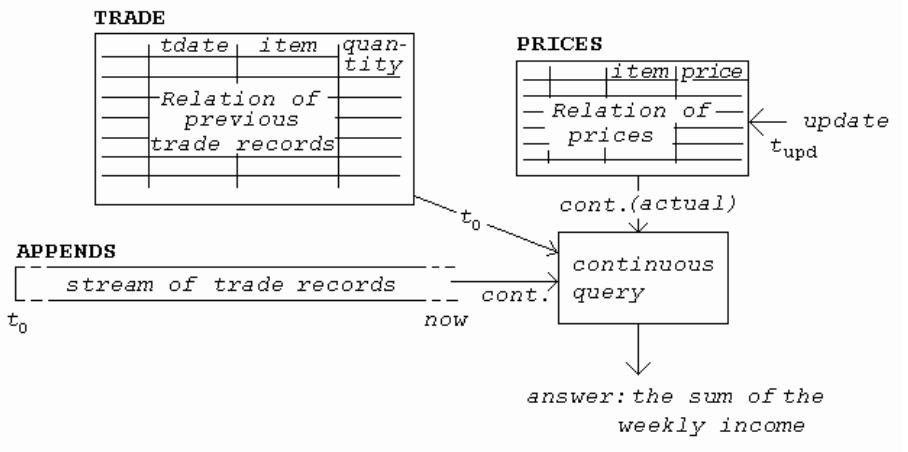

Figure 3: The environment of the continuous query
when the effect of the update belongs to the type "from now"

In Figure 3. the relation `TRADE` is the relation of the previous trade records, in the relation `PRICES` we store the actual prices and through the data stream `APPENDS` the new trade records come continuously. We can calculate the sum of the weekly income using the following continuous query:

```
SELECT SUM(quantity*price)
       FROM (STREAM (SELECT * FROM trade
                     WHERE week(tdate)=week(date))
             CONTINUE appends)
             NATURAL JOIN
             ACTUAL(prices)
```

Where the keywords `STREAM` and `CONTINUE` are the same as in the previous example, the keyword `ACTUAL` indicates that the query uses the actual state of the relation (it is changeable during the execution of the continuous query).

This query calculates the weekly income of a supermarket, taking into account the changes of the prices of the sailed items. The effects of the updates of the prices valid from the time of the updates till the next updates.
The query execution system rereads the relation of prices repeatedly or when it

detects the update of this relation, respectively. (It depends on the size of the updated relation and the size of the usable memory.)

3.3. Strongly update-sensitive queries

The third strategy is that when the query is strongly sensitive for the update. The query uses the original (i.e. contains of relation at the moment of starting of the CQL query) contains of the relations. The execution of the continuous query terminates when the content of those relation is changing. The query may be restarted manually or automatically and it produces the new answer totally independent of the previous answer.

To specify of that relation update of which causes the strongly sensitivity we suggest the following form:

```
ORIGINAL(relation)
```

The keyword `ORIGINAL` indicates that the continuous query uses the original content of the relation during the executions. The update of this relation must cause the termination of the execution of the continuous query. As we already discussed the query may be restarted automatically or manually.

The three semantics ("updatre_retro", "actual", and "original") are not mutually exclusive, it means that all versions may be used in the same query too.

3.4. About the accuracy of the answer and the inconsistency state of the database

There are several reasons which cause inaccuracies in the answer. The inaccuracies cause not necessary that the usefulness of answer would be decreased. The approximate answer may be suitable too.

It may happen that we lose a part of the data stream. From the time of terminate of the query execution till the restart of the query execution we can or cannot store and use the records from the data stream. It is one of the several other reasons why the answer should be only approximate.

During the execution of the continuous query we must take a short break (in case of `UPDATE_RETRO` the time-requirement of the virtually restart, in case of `ACTUAL` the time-requirement of reread and in case of `ORIGINAL` the time-requirement of restart). During the short break the partial lost of the stream may occur, especially when the data stream is coming very fast. In cases when the losing a part of the data stream would cause any essential problem we must avoid the problem by saving and processing of that part of the stream(s).

Another interesting area is the inconsistency. In the classical database theory it is an usually requirement that the "normal" state of a database is the consistent state. The consistency is not a permanent state, during the updates it may be damaged for a short time. The effect of the inconsistent state on the currently operating CQL queries is the inaccuracy.

4. Conclusions

The continuous data streams are native data occurrences. Perhaps ones are more native than the relations in lot of cases. Thus the using of the data streams may have number of advantages. Therefore the expansion of CQL, since being suitable for really situations might be useful. This is the reason for the paper suggested a few expansions of CQL.

References

[1] Arvind Arasu and Shivnath Babu and Jennifer Widom: *An abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations*. Invited paper in the DBPL (9th International Conference on Data Base Programming Languages) workshop, September 2003.

[2] Arvind Arasu et al.: *STREAM: The Stanford Stream Data Manager*. IEEE Data Engineering Bulletin, Vol. 26 No. 1, March 2003.

[3] Arvind Arasu and Shivnath Babu and Jennifer Widom: *The CQL Continuous Query Languag: Semantic Fundations and Query Execution*. Proceedings of the 9th International Conference on Data Base Programming Languages (DBPL) September 2003.

[4] Rajeev Motwani et al.: *Query Processing, Resource Management, and Approximation in a Data Stream Management System*. In Proc. of the 2003 Conf. on Innovative Data Systems Research (CIDR), January 2003

[5] Shivnath Babu, Lakshminarayanan Subramanian and Jennifer Widom: *A Data Stream Management System for Network Traffic Management*. In Proc. of the Workshop on Network-Related Data Management (NRDM 2001), May 2001

[6] Shivnath Babu and Jennifer Widom: *Continuous Queries over Data Streams*. SIGMOD Record, Sept. 2001

[7] Utkarsh Srivastava, Shivnath Babu and Jennifer Widom: *Monitoring Stream Properties for Continuous Query Processing*. In Proc. of the 2003 Workshop on Management and Processing of Data Streams (MPDS 2003), June 2003

Postal address

Antal Buza
Institute of Informatics
College of Dunaujvaros
Tancsics utca 1/a
2400 Dunaujvaros
Hungary