MonetDB/DataCell: leveraging the column-store database technology for efficient and scalable stream processing
Liarou, E.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 1

Introduction

1.1 From the Theory of Forms to the Knowledge Boom

Knowledge is a concept that the ancient philosophers studied more than 2000 years ago. Plato and Aristotle, already in 400 BC, tried to understand and define what is knowledge and how it is created and acquired. In the *Theory of Ideas* (or *Forms*), Plato argues that the knowledge is already created and given to us from a universal metaphysical level. In this way, he claims that we learn in this life by remembering and trying to imitate the principles that our soul already encloses from the world of *Ideas*. On the other hand, Aristotle, the most important student of Plato for twenty years, supported that the observation and the study of particular phenomena will lead us to the real knowledge.

Over the years, the definition of knowledge constituted an ongoing debate among philosophers and the triptych of the *true justified belief* has been challenged by modern epistemologists several times (Gettier, 1963). However, scientists, through the steps they follow in research, help us to realize how we come to the genesis of scientific knowledge. In that sense, we could say that scientific research agrees with the empirical aristolelean philosophy, since it depends on the observation, the measurement and the study of evidence. The collection of data, its process and evaluation constitute critical steps that transform the pure data into information, and in turn the latter one into scientific knowledge.

Even without a globally agreed definition of what knowledge is, it is a universal conviction that knowledge constitutes a very powerful and valuable *good*. 
Apart from the science and technology, connected to knowledge by an endless two-way bond, almost all the aspects of everyday life depend on knowledge and can be improved by using the existing know-how, saving us from constantly reinventing the wheel.

Today, tons of information surrounds us where we can acquire from many and different sources, such as books, mass media, social networks, etc. In particular, the World Wide Web consists of a bottomless source of new information, readily available at our fingertips. The amount of data being generated every day is still growing exponentially; it seems that for the first time in history there is more information than we can even process and consume. However, information alone does not directly bring us closer to the philosopher’s true knowledge. To this end, it becomes a matter of major importance to find ways to manage, analyze, selectively discard and exploit all this data we collect and turn it into (useful) knowledge.

1.2 Data Management

The father of history, Herodotus, aptly predicates that “Of all men’s miseries the bitterest is to know so much and to have control over nothing”. This quote was not randomly said by Herodotus, the person who first realized the importance of collecting, confirming, writing, organizing and delivering to the next generations historic material that was taking place at his time. To have control over our knowledge thesaurus is an important issue, and it becomes even more difficult the more information we have to access and the more we need to combine multiple data sets.

Taking a closer look at the technological achievements of the last century, we see that they drastically affected the creation of knowledge. In the mid of 20th century, the technological evolution and most importantly transistor’s invention, brought us closer to the information technology revolution. The reason for that was twofold; firstly it allowed the miniaturization of all modern electronics that brought on the digital information age and secondly it triggered the creation of cheaper and more powerful computational units that were able to store and process the generated data. A few decades later, the microprocessor made feasible the generation and process of such large amount of data that one hundred years ago it was hard, even impossible, to manipulate in a manual way.

To this end, data management very soon became the main concern of information technology. The big firms and organizations that were continuously generating data on a daily basis, kept asking for new technologies that would
allow them to process, analyze, visualize and manage their data in a more efficient way. Furthermore, fundamental sciences such as astronomy and biology came across even more demanding issues, since the data they started producing and want to discover patterns exceeds by far the needs of all the other fields. The geneticist Richard Lewontin, in his book titled *Biology as Ideology: The Doctrine of DNA*, characteristically states that the knowledge itself is not powerful enough, but it further empowers only those who have or can acquire the power to use it.

### 1.2.1 Database Management Systems

The necessity for new information technologies became very soon a clear research target for the computer science community and the first data management prototypes came already around the 1960s. These systems were mostly customized and used only in large organizations, who could afford the extremely high costs. Back then, a database was designed to be the system that would be responsible to store, organize and access enormous quantities of digital data in an automatic and efficient way.

One of the first Database Management Systems (DBMSs), called IMS, was built by IBM back in 1968 for NASA’s Apollo space program. Since then, we meet the database technology almost in every aspect of our electronic life. Shopping at a store, borrowing a book from the library, making bank transactions, or requesting student transcripts are only some of the examples that imply the existence of a database. A DBMS typically consists of the appropriate software that provides the insertion of new data in the database, the modification and deletion of existing data and more importantly the efficient search and retrieval of data that qualifies the requester’s constrains.

A milestone in the database research was the *relational model*, originally formulated and proposed by Edgar Codd in the 1970s (Codd, 1970). There the data is organized into a set of *tables*, which are *related* to each other in many and different ways. Each table follows a predefined schema and each record (tuple) stored in a table must also respect the same schema in order to be valid. One of the nice properties of the relational model is that we can add and access data, without reorganizing the tables every time we do so. A table can have many records and each record can have many fields (attributes). In the relational model, we distinguish the tuples of a table using a unique key, called *primary key*. Another type of keys are the *foreign keys*, used to create links between tables. In this way, the navigation among tables and the retrieval of those entries that qualify the user’s request was dramatically improved, com-
pared to other previously used models, e.g., hierarchical and network database model. Consider for example the case of a university database; there we could have several different tables such as the “students”, the “courses” and the “professors” tables. For each student we may keep a record, marked by a unique student ID, and other attributes that better describe the student, e.g., name, date of birth and address. Moreover, we may keep track of which courses he has successfully passed and the evaluation he has received, by linking the two different tables, i.e., the tables “students” and “courses”. Also, each course is linked to the “professors” table to indicate who is teaching it in each semester. By organizing our data in that structured way, we can easily navigate through the tables and retrieve any combinative query, e.g., give me all the professors that teach a course with success rate greater than 70% and average student grades 8 out of 10, for at least 5 years in a row.

The success of the relational model, mainly comes from the fact that it works in a declarative way. Relational databases are extremely easy to customize to fit almost any kind of data. The user is able to access and manipulate his data without being involved in technical decisions that have to do with designing how the data will be stored and how the requests are going to be executed. Through SQL, a declarative query language, the user obtains full control over the data and is able to describe in an abstract way what kind of information he is interested in, keeping his hands clean of any internal low-level system specifications. On the contrary, the DBMS is in charge to decide autonomously what is the best way to organize and physically store the data, and designs the appropriate strategy for getting the user’s queries answered.

Apart from the powerfulness and the flexibility that the database systems provide, another reason that contributed to their wide use is coming for the fact that they are generic enough and able to handle multiple users at the same time. A DBMS has the appropriate mechanisms to always ensure data integrity, despite multiple concurrent users or different application programs are accessing the same database. The ACID properties are the main rules at the database cookbook, that guarantee safe transaction executions. In short, the first rule, called atomicity, implies that once a transaction starts it should be fully completed otherwise it should become in a status as if it never happened. All transactions must maintain the consistency of the database. Two concurrent transactions must be isolated and not interfere with each other while happening. Finally, once a transaction is completed the DBMS guarantees that its impact to the database will be durable from here on. Working under these rules and balancing with mastery between reliability and performance, the database systems very soon convinced the big firms and organizations that they are trustworthy
and skilled to manage their valuable data.

Database management systems were created to provide persistent data storage and an efficient and reliable answering mechanism. It is the suggested complete software solution, when the application scenario prerequisites that the data is a priori known and relatively static. A DBMS typically stores, organizes, indexes and prepares the stored data to the best of its knowledge and it becomes ready to accept and immediately answer the potential queries that will be posed in the future. Once a request comes, the database system syntactically and semantically analyzes it, and based on a predefined set of rules, as well as previously acquired query processing experience (e.g., statistics, indices), it decides what is the best query plan to use for deriving the matching answers. The execution engine precisely follows the designed query plan, and evaluates the query over the data that is currently stored in the database.

Database systems constitute an alive evolving research field for the past 50 years. Their quick commercial exploitation, challenged their initial capabilities and brought out their potential weak points. Many research subfields have been created to fill in the gaps and strengthen their features; some focus at the core level of query processing and optimization, and others cope with higher level topics such as language interfaces, distributed and parallel processing, privacy and security issues or research related to web applications. The diverse market needs motivate the expansion of different database architectures. Both a small business and an astronomical data center may use a database system, but their fundamentally different requirements drove researchers and developers to design different database architectures and solutions. Many open source prototypes, as such PostgreSQL, MySQL and MonetDB not only survived through the years but also keep leading the database research. Moreover the big players of the commercial arena, such as Oracle, IBM and Microsoft, are continuously investing in the ongoing database technology evolution.

Half a century after the first prototypes, database systems are still the center of attention of information technology. It seems that there are still more to research, since new data sources challenge their capabilities and performance every day.

1.3 Data Stream Management

Plenty of application scenarios fit in the traditional database processing scheme. However, a new type of applications, called data streams, that came a few decades after the establishment of the DBMSs, could not be satisfied by that
model. In the data stream scenario, we have to deal with the *continual* generation and processing of an infinite flood of data (stream). Queries on the other hand, appear to be persistent, namely once they are submitted they remain active forever or for at least a long period of time. These two fundamental differences on the queries and data lifetime, became enough to make very soon clear that database systems were not skilled to handle such applications. This way, the computer scientists started looking for new system architectures that could fulfill the new requirements.

A potentially large application domain stimulates the creation of data stream management systems (DSMSs). Sensors, organized in wireless networks, that continuously measure physical, biological or chemical input, nicely fit in the data stream model. The sensors produce streams of data that continuously should be analyzed in real-time to keep track of environmental conditions and detect anomalies in case they happened. Smoke detectors, health-care monitors and traffic controllers are only some simple examples that fall in that application scenario. Furthermore, sensor networks take control over smart building design, or can be used to wildlife tracking systems to give rich information to animal biologists.

In the same line, network monitoring systems continuously need to analyze the network traffic to catch potential problems, such as unusual activity, delays, server crashes and bottlenecks. They derive information enclosed in IP packets, while they are passing through the network, and generate the appropriate alerts when they diagnose a problematic behavior.

Financial trading applications is another scenario that meets the data stream requirements. The idea is the same also here, continuous fast updated information coming from different sources, should be analyzed and combined to accomplish profitable transactions. The list of applications that inspired the creation of data stream systems is long. For example, consider that the World Wide Web provides a plethora of streaming opportunities through web feeds. Users are able to subscribe to interesting sources of information and they are automatically notified when new data is available.

**DSMS vs. DBMS**

Let us now see in more detail what are the main fundamental differences between the database and the data stream application scenario.

- **Continuous query processing.**
  
  In a stream application, we need mechanisms to support *long-standing/*
1.3. DATA STREAM MANAGEMENT

Continuous queries over data that is continuously updated from the environment. The queries are issued once and then they stay active for a long time, monitoring the incoming data. On the contrary in a database scenario, the user poses a query and receives the corresponding answers only once. If he wants to check for potentially different answers later, he should re-submit the same query to the database. The database traditionally, evaluates that queries all over again, without taking into account the previous evaluation.

- **Data lifetime.**
In the database scenario, the data is characterized as persistent. Updates over the data are expected but their rate is less frequent than the incoming rate of queries. On the other side, in a stream application, we should be prepared to handle an infinite sequence of data in real-time. Typically once the data comes it is analyzed against waiting queries and then it is forgotten.

- **Pull vs. Push model**
Taking into account the way that a DBMS treats data and queries, we could say that it follows a pure pull-based model, since each time a new query arrives, the engine pulls the data from the disk to search for answers. On the contrary, a typical DSMS works in a push-based way, pushing the incoming streams to meet the interested waiting queries.

- **Real-time processing**
In data stream applications, it is very important to achieve real-time processing. Delays may affect answer’s validity and also could produce system bottlenecks, since more data will be continuously collected. A data stream engine should be alert and process the incoming data in real-time.

- **Workload fluctuations**
Data stream arrival may vary dramatically. There are application scenarios with low data input rates, such as sensors that update their measurements every one minute, or other cases where we have to deal with extremely high input stream rates. For example, most recent generation of satellites provides ground reception rates of 300 Mbit/sec and 800 Mbit/sec. Environment and workload changes call for adaptive processing strategies at the query evaluation level to achieve the best query response time. In databases we have to deal with workload variation too, but in
terms of queries. In that case, it may become mandatory to update our indices over the stored data.

- **Window processing**

As we have already mentioned a stream is an infinite sequence of data. Given that hardware and software have limitations, we also need to limit the maximum amount of data we can gather and process within a given time budget. The initial stream processing models were very simple; they were producing answers by considering only one incoming tuple at a time. The window processing model came as an intermediate solution between single tuple and database processing. In this case, the system produces answers considering a number of collected stream tuples, instead of just a single one. The window processing model increases the expressiveness of stream systems, allowing for aggregations and joins in addition to simple filtering queries.

- **Query languages**

Taking all the previous factors into account, the existence of new data models and query languages was necessary for the establishments of the DSMSs. The language used in relational databases, was not sufficient to represent the different nature and semantics of stream data and queries.

Given these differences, and the unique characteristics and needs of continuous query processing, the pioneering DSMS architects naturally considered that the existing DBMS architectures are inadequate to support stream processing and achieve the desired performance. Another aspect is that the initial stream applications had quite simple requirements in terms of query processing. This made the existing DBMS systems look overloaded with functionalities. These factors led researchers to design and build new architectures from scratch and several DSMS solutions have been proposed over the last years giving birth to very interesting ideas and system architectures, e.g., (Babcock et al., 2004; Balakrishnan et al., 2004; Chandrasekaran et al., 2003; Chen et al., 2000; Cranor et al., 2003; Girod et al., 2007).
1.4 The DataCell: a DSMS into the heart of a DBMS

1.4.1 Motivation

As we discussed earlier, the diverse needs for persistent data management and continuous queries processing, brought two different system architectures. However, the data management evolution does not seem to stop here. The last few years a new processing paradigm is born, where incoming data (stream) needs to quickly be analyzed and possibly combined with existing data to discover trends and patterns. Subsequently, the new data enters the data warehouse and is stored as normal for further analysis if necessary.

Natural sciences such as astronomy and biology that deal with large amounts of data, also motivate that paradigm. In 2015 the astronomers will be able to scan and catalog the entire night sky from a mountain-top in Chile, recording 30 Terabytes of data every night which incrementally will result in an absolutely massive 150 Petabyte database (over the operation period of ten years). It will be capturing changes to the observable universe evaluating huge statistical calculations over the entire database. Another characteristic data-driven example is the Large Hadron Collider (LHC) (LHC, 2010), a particle accelerator that will revolutionize our understanding for the universe, generating almost 40 Terabytes of data every day and collecting 15 petabytes of data annually. The same model stands for modern data warehouses which enrich their data on a daily basis creating a strong need for quick reaction and combination of scalable stream and traditional processing (Winter and Kostamaa, 2010).

In this new paradigm incoming streams of data need to quickly be analyzed and possibly combined with existing data to discover trends and patterns. We need scalable query processing that can combine continuous querying for fast reaction to incoming data with traditional querying for access to the existing data. However, neither pure database technology nor pure stream technology are designed for this purpose.

In this thesis, we propose that a complete integration of database and streaming technology is the way to go. We focus on the design, study and development of such a system that integrates both streaming and database technologies in the most natural way. A fully functional stream engine, called DataCell, is designed on top of an extensible DBMS kernel. Our goal is to fully exploit the generic storage and execution engine of the DBMS as well as its complete optimizer stack. The end goal is a single system that does combine properties and
features from both the database world and the stream world, and thus achieves efficient performance for both one-time and continuous queries.

1.4.2 The Basics

The ultimate goal of this thesis is to support full data management of persistent and streaming data within an integrated processing kernel. Instead of building a new system from scratch we opt to work over an extensible DBMS kernel such that we can exploit mature techniques and algorithms in the area of database systems. The challenge becomes how to extend such a scalable system such that it supports stream processing in addition to one-time processing.

Stream researchers in the past argued that this is not feasible as it would be very inefficient. DataCell shows that this is not true anymore, and it successfully combines both paradigms.

Our design and implementation is over the MonetDB system. MonetDB is an open source column-store database management system, developed and maintain at CWI. Several aspects of MonetDB make this research possible. For instance, MonetDB allows for easy manipulation and extension of its optimizer module which allows us to easily introduce new optimizer rules specific for DataCell while at the same time exploiting all existing optimizer rules a DBMS has to offer. In addition, MonetDB is one of the leading column-store systems. We heavily exploit its column-store nature in our techniques to speed up stream processing exploiting critical column-store features such as vectorization.

The main idea is that when stream tuples arrive into the system, they are immediately stored in (appended to) a new kind of lightweight tables, called baskets. By temporarily collecting tuples into baskets, we can evaluate the continuous queries over the baskets as if they were normal one-time queries and thus we can reuse any kind of algorithm and optimization designed for a modern DBMS. Once a tuple has been seen by all relevant queries/operators, it is dropped from its basket.

Continuous query plans are represented by factories, i.e., a kind of co-routine, whose semantics are extended to align with table producing SQL functions. Each factory encloses a query plan that once it is evaluated it produces a partial result at each call. For this, a factory continuously reads data from the input baskets, evaluates its query plan and creates a result set, which it then places in its output baskets. The factory remains active as long as the continuous query remains in the systems, and it is always ready to consume incoming stream data.

The execution of the factories is orchestrated by the DataCell scheduler. The
firing condition is aligned to arrival of events, once there are tuples that may be relevant to a waiting query we trigger its evaluation over these tuples. Furthermore, the scheduler manages the time constraints attached to event handling, which leads to possibly delaying events in their baskets for some time. One important merit of the DataCell architecture, is the natural integration of baskets and tables within the same processing fabric. A single factory can interact both with tables and baskets, this way we can naturally support queries interweaving the basic components of both models.

By introducing the baskets, the factories and the DataCell scheduler, our architecture becomes able to proceed sufficiently data streams, without also losing any database functionality. That is the natural first step that covers the gap between the two incompatible processing models. However, numerous research and technical questions immediately arise. The most prominent issues are the ability to provide specialized stream functionality and hindrances to guarantee real-time constraints for event handling. Also, we need to cope with (and exploit) similarities between the many standing queries, in order to deal with high performance requirements.

1.4.3 Research Challenges

It is a major challenge for the DataCell architecture to efficiently support and integrate all specialized stream features. The above description gives the first directions that allow the exploration of quite flexible strategies, once we have to deal with low latency deadlines or multi-query processing.

The road-map for DataCell research calls for innovation in many important aspects of stream processing and the combination with already stored data. Thus, one can distinguish between challenges that come from the fact that stream processing is performed in a DBMS and challenges that arise by combining the two query processing paradigms in one.

Regarding the first challenge, the goal is to provide all essential streaming functionality and features without losing the DBMS’s strong storage and querying capabilities. We draw a path where most of the streaming functionality is provided via plan rewriting and minimal lower level operator changes. For example, resource management, scheduling, and optimization in the presence of numerous queries is a critical topic. Similarly to incremental processing, this area has received a lot of attention with innovative solutions, e.g., (Sharaf et al., 2008). DataCell offers all the available ingredients to achieve similar levels of multi-query optimizations while keeping the underlying generic engine intact. For example, a single factory (i.e., plan) may dynamically split into multiple
pieces or merge with other relevant factories to allow for efficient sharing of processing costs leading to very interesting scenarios in how the network of factories and baskets is organized and adapts. Again, these issues can be resolved at a higher level through plan rewriting. The intermediates created for incremental processing can be reused by many queries, while partitioning and scheduling decisions can also adapt to the new parameters. We have the appropriate technology to make multiple queries to cache and exploit intermediates in a column-store kernel.

Regarding the second challenge, a plethora of rich topics arise especially when optimization becomes an issue. For example, query plans that touch both streaming data and regular tables might require new optimizer rules or adaptations of the current ones. Overall, DataCell opens the road for an exciting research path by looking at the stream query processing issue from a different perspective.

1.4.4 Contributions

The particular contributions of this thesis can be summarized as follows:

1. **A new Stream Paradigm.** We show that the past belief that stream query processing requires a specialized engine only for stream processing is not sufficient anymore, especially due to the increasing scalability requirements.

2. **DataCell architecture.** We introduce the basic DataCell architecture, to exploit the notion of scalable systems that can provide both streaming and database functionality. We describe what are the minimal additions that allow for stream processing within a DBMS kernel.

3. **Incremental processing.** We show how to efficiently support core streaming functionalities in DataCell, i.e., incremental stream processing and window-based processing.

4. **Multi-query processing.** We investigate multi-query processing opportunities, another critical feature required in stream processing. Sharing access of common basket, splitting, merging and dynamically reorganizing the factories content are some cards we use on the performance hunting game.
1.4. **THE DATACELL: A DSMS INTO THE HEART OF A DBMS**

(5) **A Query Language for DataCell.** We propose a language for DataCell that extends SQL. It can be used to access both streaming and database data at the same time.

(6) **Research Path.** We discuss in detail the new research area that opens with the notion of DataCell and what are the future challenges towards systems that can handle streams of multiple Terabytes on a daily basis.

### 1.4.5 Published Papers

The content of this thesis is built based on a number of publications in major international conferences in the area of database management systems, of computer science.


(3) Erietta Liarou and Martin Kersten. DataCell: Building a Data Stream Engine on top of a Relational Database Kernel. In Proceedings of the 35th International Conference on Very Large Data Bases, VLDB PhD Workshop, Lyon, France, August 2009

(4) Erietta Liarou, Stratos Idores, Stefan Manegold and Martin Kersten. Enhanced Stream Processing in a DBMS Kernel. Submitted for publication at the moment of printing this thesis.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2 we extensively discuss related work. First, we search the roots of data stream management systems in the heart of database applications and technology. We dedicate enough space to describe and compare our work with other successful pure data stream management systems, from the academic and commercial world. Finally, we provide the mandatory background of our development platform, the MonetDB system.

In Chapter 3 we describe the basic DataCell architecture. First, we explain how we represent continuous queries and data streams in DataCell, the two main stream concepts that until now were unknown in conventional databases. Then we describe the full architecture and the newly introduced components that give full stream functionality to our system.

In Chapter 4 we present the DataCell language interface. We propose a semi-procedural language as a small extension of SQL, that can be used to access both streaming and database data at the same time. The language concepts introduced are compared against building blocks found in “pure” stream management systems. They can all be expressed in a concise way and demonstrate the power of starting the design from a full-fledged SQL implementation.

Chapter 5 presents how we handle in DataCell one of the most important specialized stream processing requirements, i.e., incremental window processing. Even with the conventional underlying infrastructure that MonetDB offers to DataCell, we manage to compete against a specialized stream engine, elevating incremental processing at the query plan level, instead of building specialized stream operators.

Chapter 6 concludes the thesis and discusses a number of future interesting open topics and possible research directions towards a complete data management architecture that integrates database and stream functionalities in the same kernel. DataCell opens the road for an exciting research path by looking at the stream query processing issue from a different perspective and by taking into account the needs of modern data management for scalable stream processing combined with traditional query processing. Topics we discuss in
this chapter include multi-query processing, adaptive query processing, query relaxation, distributed processing, DataCell in different architectures, etc.