Cell Stores

Ghislain Fourny
28msec, Inc.
Zürich, Switzerland
g@28.io

ABSTRACT

Cell stores provide a relational-like, tabular level of abstraction to business users while leveraging the latest database technologies, such as key-value stores and document stores, in order to benefit from the latest technological breakthroughs to scale up and out the efficient storage and retrieval of highly dimensional data. Cells are the primary citizens and exists in different forms: as a gas for efficient storage, as a solid for efficient retrieval, and as a liquid for efficient interaction with the business users.

1. INTRODUCTION

In 1970, Codd [cite] introduced the relational model as an alternative to the graph and network models (such as file systems) in order to provide a more suitable interface to users, and to protect them from internal representations ("data independence").

The relational model's first implementation was made public in 1976 by IBM [cite].

In the last four decades, the relational model has been enjoying undisputed popularity and has been widely used in enterprise environments. This is probably because it is both very simple to understand and universal. Furthermore, it is accessible to business users without IT knowledge, to whom tabular structures are very natural — as demonstrated by the strong usage of spreadsheet software (such as Microsoft Excel) as well as user-friendly front-ends (such as Microsoft Access).

However, in the years 2000s, the exponential explosion of the quantity of data to deal with increasingly showed the limitations of this model. Several companies, such as Google, Facebook or Twitter needed scaling up and out beyond the capabilities of any RDBMS, both because of the *quantity* of data (rows), and because of the *high dimensionality* of this data (columns). Each of them built their own, ad-hoc data management system (Big Table, Cassandra, ...). These technologies often share the same design ideas (scale up through clustering and replication, high dimensionality handling through data heterogeneity and tree structures), which led to the popular common denomination of NoSQL, a common roof for:

- key-value stores
- document stores
- column stores
- graph databases

NoSQL solves the scale-up issue, but at a two-fold cost:

- for developers, the level of abstraction provided by NoSQL stores is much lower than that of the relational model. These data stores often provide limited querying capability such as point or range queries, insert, delete and update (CRUD). Higher-level operations such as joins must be implemented in a host language, on a case by case basis.
- for business users, these data models are much less natural than tabular data. Reading and editing data formats such as XML, JSON requires at least basic IT knowledge. Furthermore, business users should not have to deal with indexes at all.

This is a major step back from Codd's intentions back in the 1970s, as the very representations he wanted to protect users from (tree-like data structures, storage, ...) are pushed back to the user.

Reluctance can be observed amongst users, and this might explain why the "big three" (Oracle, Microsoft, IBM) are heavily forcing the usage of the SQL language on top of these data stores.

This paper introduces the cell stores data paradigm, whose goal is to (i) leverage the technological advancements made in the last decade, while (ii) bringing back to business users control and understanding over their data.

Cell stores are at a sweet spot between on the one hand key-value stores, in that they scale up seamlessly and gracefully in the quantity of data as well as the dimensionality of the data, and on the other end the relational model, in that business users access the data in tabular views via familiar, spreadsheet-like interfaces.

Section... gives an overview of state of the art technologies for storing large quantities of highly dimensional data and their shortcomings. Section ... motivates the need for the cell stores paradigm. Section ... introduces the data model behind cell stores. Section ... shows how a relational

Figure 1: A cell

Dimension	Value
Concept	Assets
Period	Sept. 30th, 2012
Entity	Visa
Unit	US Dollars
Class of Stock	Class of Stock A
40,013,000,000	

database can be stored naturally in a cell store. Section... points out that there is a standard format for exchanging data between cell stores as well as other databases. Section... gives implementation-level details. Section... gives a few more miscellaneous remarks, and section... explores performance.

2. STATE OF THE ART

Relational databases
Key value stores
Document stores
Column stores

3. WHY CELL STORES

Sparseness
User-friendliness
Cell stores bring back control to business users:

- scales up with heterogeneous data
- accessible via spreadsheet-like interface
- control of taxonomies (hierarchies of concepts)
- business rules

4. THE CELL STORE DATA MODEL

4.1 Gas of cells

As the name "cell store" indicates, the first class citizen in this paradigm is the cell. If you think or OLAP or XBRL, it is also called fact, or measure. If you think of a spreadsheet, it really corresponds to a cell. If you think of a relational database, it corresponds to a single value on a row.

The cell can be seen as a atom of data, in that it represents the smallest possibly reportable unit of data. It has a single value, and this value is associated with dimensional coordinates that are string-value pairs. These dimensional coordinates are also called aspects, or properties, or characteristics. They uniquely identify a cell, and a consistent cell store should not contain any two cells with the exact same dimensional pairs.

There are no limits to the number of dimension names and their value space. Cell stores scale up seamlessly with the total number of dimensions. There is only one required dimension called *concept*, which describes *what* the value represents. All other dimensions are left to the user's imagination, although typically you will find a validity period (instant or duration: when), an entity (who), a unit (of what), etc.

Figure 1 shows an example of a single cell.

The main idea of the cell store paradigm is that there is a single one, big collection of cells. All the data is in this collection, and on a logical level, this collection is not partitioned or ordered in any (logical) way. An analogy can be made with a gas of molecules, where the molecules fly around without any particular order or structure.

In the same way as gas can be stored in containers, the cell gas can (should) be clustered and replicated to enhance the performance of the cell store. Whether clustering is done randomly or following a pattern based on dimension values is mostly driven by optimization and performance on a use case basis.

4.2 Hypercubes

4.2.1 Point queries and indices

Now that we have a cell of gas available, we can begin to play with it. The first idea that comes to mind is how to retrieve a cell from the gas (point query).

Point queries leverage the index capabilities of the underlying storage layer. If the cell gas is small and contains many concepts, a single hash index on the concept dimension will be enough. For bigger cell gas, other techniques allow scaling up, such as:

- compound keys: a single index on several fields such as concept, period and entity.
- separate hash keys: use single indices separately, and compute their intersection.

4.2.2 Hypercube queries

In technologies such as OLAP, the first class citizen is the hypercube, which can be seen as the *schema*. In cell stores, the hypercube can be seen as the *query*.

A hypercube is a dimensional range (as opposed to dimensional coordinates). It is made of a set of dimensions, and each dimension is associated with its range, which is a set of values. The range can be either an explicit enumeration (for example, for strings), or an interval (like the integers between 10 and 20), or also more complex multi-dimensional ranges (consider GIS).

Figure 2 shows an example of hypercube. It looks a bit like a cell, except that there is no value, and dimensions are associated with ranges rather than single values.

A cell belongs to a hypercube if:

- it has exactly the same dimensions
- for each dimension, the value belongs to the domain of that dimension as specified in the hypercube

Figure 2 also shows two cells satisfying the above criterion.

Like point queries, hypercube queries also leverage indices. Range indices, in addition to or as an alternative to hash indices, prove particularly useful in the case of numeric or date dimension values. Domain-specific indices like GIS also fit well in this picture.

4.2.3 Default dimension values

In cell stores, the number of dimensions and their names vary across cells. Hypercube queries accommodate for this flexibility with the notion of a default dimension value.

Figure 3 shows a hypercube that defines a default value of "Domain" for the "Class of Stock" dimension.

If a hypercube specifies a default value for a given dimension, then the condition that a cell must have that dimension

Figure 2: A hypercube containing 18 cells

Dimension	Value
Concept	Assets, Equity, Liabilities
Period	Sept. 30th, 2012, Dec. 31st, 2012
Entity	Visa, Mastercard, American Express
Unit	US Dollars

Example of cells within this hypercube:

Dimension	Value
Concept	Equity
Period	Dec. 31st, 2012
Entity	Mastercard
Unit	US Dollars
40,013,000,000	

Dimension	Value
Concept	Liabilities
Period	Dec. 31st, 2012
Entity	American Express
Unit	US Dollars
40,013,000,000	

to be included in the hypercube is relaxed. In particular, a cell will also be included if it does not have a "Class of Stock" dimension. When this happens, an additional dimensional pair is added to the cell on the fly, using the default value as value. This implies that in the end, the set of cells that gets returned for the hypercube query always has exactly the dimensions specified in the hypercube.

In particular, a hypercube is highly structured.

4.2.4 *The "Big Cube"*

Theoretically, it would be feasible to build a hypercube with all dimensions used in the gas of cells, allowing default values for all of these. Then all cells would belong to this hypercube. However, this is an extremely sparse hypercube, and the size of this hypercube would typically be orders of magnitude greater than the entire visible universe.

4.2.5 *Materialized hypercube*

The answer to a hypercube query can be showed in a consolidated way, resembling a relational table. Each column corresponds to a dimension, and the last column to the value. Figure 4 shows the materialized hypercube corresponding to the hypercube shown in Figure 3.

4.3 Spreadsheet views

A hypercube can be materialised into a table as shown in the former section. With the state of matter analogy, it can be seen as the solid version of a (very small) subpart of the gas of cells.

From a business viewpoint, tables are very useful because they can be understood without IT knowledge. However, a materialised hypercube displays the multidimensional data under a very raw form. This raw form is actually very common though, so that mainstream spreadsheet software provide a feature that allows interacting with multidimensional data with a better UI. This feature is often called *pivot table* and flattens the data to a two-dimensional sheet.

The dimensions are partitioned between:

Slicers All the data that does not match the slicers is discarded.

Figure 3: A hypercube using a default dimension value (shown in square brackets)

Dimension	Value
Concept	Assets, Equity, Liabilities
Period	Sept. 30th, 2012
Entity	Visa, Mastercard, American Express
Unit	US Dollars
Class of Stock	Class of Stock A, [Domain]

Example of cells within this hypercube:

Dimension	Value
Concept	Assets
Period	Sept. 30th, 2012
Entity	Visa
Unit	US Dollars
Class of Stock	Class of Stock A
40,013,000,000	

Dimension	Value
Concept	Assets
Period	Sept. 30th, 2012
Entity	Visa
Unit	US Dollars
Class of Stock	*[Domain]*
40,013,000,000	

Dicers They specify, for each row and column, what dimensional constraints the data inside must fulfil

Values (potentially aggregated) They specify, for each cell, which property is displayed as well as, if there are several values, how to aggregate them (sum, count, max, min, average, etc).

This functionality is straight-forward to implement on top of a cell store, because the raw data is in exactly the same form. The XBRL specifications also contain a feature called *table link base* that standardises how to specify such spreadsheet views.

Figure 5 shows an example of how the materialised hypercube on Figure 4 can be displayed in this more business-oriented manner.

Concretely, the construction of the view can be pushed to the server or the cell store itself:

- given a hypercube (and possibly the cells it contains, queried from the store), a spreadsheet can be smartly generated. Slicers are taken from all dimensions that only have a single value across the hypercube, concepts can be assigned to the rows and the remaining dimensions to the columns.
- given a spreadsheet definition (say, table link base), a hypercube can be generated in order to obtain all the relevant cells from the underlying cell store.

It is also still possible for a business user to obtain the materialised hypercube from the cell store, and to import it as is in their spreadsheet software.

As is commonly done in spreadsheets, business users can drag and drop dimensions across the different categories to fine tune their view over the data. Spreadsheet views are not only convenient to read, but also to write data back to the cell store, cell by cell. Compared to the cell gas and the solid materialised hypercube, the spreadsheet view is comparable

Figure 4: A materialized hypercube

Concept	Period	Entity	Unit	Class of Stock	Value
Assets	Sept. 30th, 2012	Visa	USD	Class of Stock A	1,000,000,000
Assets	Sept. 30th, 2012	Visa	USD	[Domain]	1,000,000,000
Assets	Sept. 30th, 2012	Mastercard	USD	Class of Stock A	1,000,000,000
Assets	Sept. 30th, 2012	Mastercard	USD	[Domain]	1,000,000,000
Assets	Sept. 30th, 2012	American Express	USD	Class of Stock A	1,000,000,000
Assets	Sept. 30th, 2012	American Express	USD	[Domain]	1,000,000,000
Equity	Sept. 30th, 2012	Visa	USD	Class of Stock A	1,000,000,000
Equity	Sept. 30th, 2012	Visa	USD	[Domain]	1,000,000,000
Equity	Sept. 30th, 2012	Mastercard	USD	Class of Stock A	1,000,000,000
Equity	Sept. 30th, 2012	Mastercard	USD	[Domain]	1,000,000,000
Equity	Sept. 30th, 2012	American Express	USD	Class of Stock A	1,000,000,000
Equity	Sept. 30th, 2012	American Express	USD	[Domain]	1,000,000,000
Liabilities	Sept. 30th, 2012	Visa	USD	Class of Stock A	1,000,000,000
Liabilities	Sept. 30th, 2012	Visa	USD	[Domain]	1,000,000,000
Liabilities	Sept. 30th, 2012	Mastercard	USD	Class of Stock A	1,000,000,000
Liabilities	Sept. 30th, 2012	Mastercard	USD	[Domain]	1,000,000,000
Liabilities	Sept. 30th, 2012	American Express	USD	Class of Stock A	1,000,000,000
Liabilities	Sept. 30th, 2012	American Express	USD	[Domain]	1,000,000,000

to a metal that gets melted before the blacksmith can shape it at will.

Hence, because cell stores can use all the experience accumulated over several decades on pivot tables from the spreadsheet industry, they offer a very powerful and business friendly interface, shielding users from the underlying dimensional complexity.

To a business user, working with a cell store feels like working on a spreadsheet, except that

- the size of the data is orders of magnitude bigger than a spreadsheet file;
- the data lies on a server and is shared across a department or a company;
- the latest database technologies are leveraged under the hood to scale up and out, without the need to go through the IT department for each change in the business taxonomy.

4.4 Maps

When many people define their own taxonomy, this often ends up in redundant terminology. For example, someone might use the term Equity and somebody else Capital. When either querying cells with a hypercube, or loading cells into a spreadsheet, a mapping can be applied so that this redundant terminology is transparent. This way, when a user asks for Equity, (i) she will also get the cells having the concept Capital, (ii) and it will be transparent to her because the Capital concept is overridden with the expected value Equity.

4.5 Rules

One of the reasons spreadsheets are very popular is that formulas can be entered into cells to automatically compute values.

Cell stores support an equivalent capability called *rules*. Like maps, rules are executed in a transparent way during a hypercube query, or when a spreadsheet is requested.

From a high-level perspective, cell stores support two kinds of rules:

Imputation rules computes a value for a missing cell (i.e., dimensional coordinates against which no value was reported). When generated, this cell comes along with an audit trail that indicates how the value was computed, and from which other cells.

Validation rules check that the value for a given cell is consistent with the values reported in other cells (often neighbours in the spreadsheet view). A validation rule typically results in a green tick or a red cross in the corresponding cell on the spreadsheet view.

Rules can be defined according to several metapatterns, as defined by Charles Hoffman. It is most intuitive to think about them having in mind the spreadsheet view.

Roll Up Several cells with different Concepts (but with the exact same other dimensions) are aggregated (often with a sum) into a roll up value. This corresponds to summing across a column or row in Excel.

Roll Forward A value for a new instant in time is deduced from the equivalent value at a former time, as well as the delta value on the corresponding time interval.

Compound Fact This is the same as a roll up, except that instead of the Concept varying, the aggregation is computed against a different dimension.

Adjustment This is similar to a roll forward, except that the time correspond to transaction time, not valid time, and the delta corresponds to a correction or an amendment.

Variance This is similar to a compound fact, except that the dimension used has the semantic of two different scenarios.

Complex Computation This is a generalized roll-up.

Grid This metapattern involves the conjunction of two other metapatterns, for example, in the spreadsheet view, a Roll Up on the rows and a Compound Fact on the columns.

Figure 5: A spreadsheet view over a hypercube

Unit	USD					
Period	Sept. 30th, 2012					
Line items	**Entity**					
	Visa		Mastercard		American Express	
	Class of Stock		**Class of Stock**		**Class of Stock**	
	Class of Stock A		Class of Stock A		Class of Stock A	
Assets	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000
Equity	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000
Liabilities	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000	1,000,000,000

To facilitate the definition of rules, concepts and dimension values are organised in hierarchies. For example, a roll up is typically a parent's being the sum of its children.

5. CANONICAL MAPPING TO THE RELATIONAL MODEL

There is a direct, two-way canonical mapping between hypercubes and relational tables.

A materialized hypercube can be converted to a relational table with equivalent semantics by removing the Concept column, and replacing the Value column with one column for each Concept, as shown on Figure 6. The set of all attributes corresponding to the dimension columns acts as a primary key to the table.

Conversely, any relational table can be converted to a cell gas and its corresponding hypercube as follows: each attribute in the primary key is converted to a dimension. A cell is then created for each row and for each value on that row that is not a primary key. This cell is associated with the dimensions values corresponding to the primary keys on the same row, plus the Concept dimension associated with the name of the attribute corresponding to the column.

The consequence of this is that an entire relational database with multiple tables, or even several relational databases, can be converted into a single cell store. Likewise, relational views can be built dynamically on top of a cell store.

6. STANDARDIZED DATA INTERCHANGE: XBRL

Many ideas behind the cell store paradigm originate from the XBRL standard. There are three main reasons for this:

- The XBRL standard was designed by a team who is aware of the needs of business users, and of the challenges of business reporting.
- It is important that the data stored in a cell store is not locked in this cell store, i.e., that it can be exported in such a way that other users, even not cell store users, can understand it and use it without ETL efforts. The XBRL standard makes sure that this is so: cell stores can import XBRL data, and export their content into the XBRL format.
- Cell stores were designed with the goal of providing efficient storage and retrieving capabilities for XBRL data. They provide an abstract data model on top of XBRL that is a viable and efficient alternative to other implementations, such as storing each XBRL hypercube in ROLAP, or such as importing raw XBRL filings into an XML database.

XBRL is very complex and involves many different specifications.

XBRL, on the physical level, uses XML technologies: filings are reported with a (flat) XML format, and metadata (called taxonomies) using XML Schema and XML Linkbases.

The counterpart of a cell is called a fact. A numeric fact may also be stamped with information on the precision or the number of decimals.

In XBRL, dimensions are called aspects. There are three "builtin" aspects in addition to Concept: Period, Entity and Unit, and taxonomies may define more dimensions.

Taxonomies define concepts, hypercubes, dimensions, dimension values, can be shared at any level (reporting authority, company, department, etc) and extended at will.

XBRL Linkbases provide metadata information in the form of "networks":

Definition networks allow, among others, building hypercubes and specifying which concepts are bound to which hypercubes, which hypercubes have which dimensions, and which dimensions have which values. Dimensions may either have a typed value space, or be an explicitly enumerated set.

Presentation networks allow the hierarchisation of concepts and dimension values in a spreadsheet view (=Table link base in XBRL). For example, a balance sheet may be divided into an Assets hierarchy and an EquityAndLiabilities hierarchy. A presentation network can also contain abstract concepts, i.e., they are only here to organise and partition other concepts.

Table linkbase networks define how a spreadsheet view looks like, i.e., which are the slicers, the dicers, how the dicers are organised in rows and columns, etc.

Calculation networks are the simplest kind of roll up rules.

Label networks associate business-friendly labels to concepts, dimensions and dimension values, because the latter are often stored in a very raw form that is not palatable to non IT-savvy users.

Formula networks define rules to automatically impute or validate fact values.

7. IMPLEMENTATION

Figure 6: A relational table corresponding to a hypercube

Period	Entity	Unit	Class of Stock	Assets	Equity	Liabilities
Sept. 30th, 2012	Visa	USD	Class of Stock A	1,000,000,000	1,000,000,000	1,000,000,000
Sept. 30th, 2012	Visa	USD	[Domain]	1,000,000,000	1,000,000,000	1,000,000,000
Sept. 30th, 2012	Mastercard	USD	Class of Stock A	1,000,000,000	1,000,000,000	1,000,000,000
Sept. 30th, 2012	Mastercard	USD	[Domain]	1,000,000,000	1,000,000,000	1,000,000,000
Sept. 30th, 2012	American Express	USD	Class of Stock A	1,000,000,000	1,000,000,000	1,000,000,000
Sept. 30th, 2012	American Express	USD	[Domain]	1,000,000,000	1,000,000,000	1,000,000,000

7.1 On top of a document store: NoLAP

7.2 On top of a column store (e.g. Big Table)

7.3 On top of a key-value store (Cassandra)

8. MISCELLANEOUS

8.1 Temporal databases

Transaction time
Valid time

9. PERFORMANCE

secxbrl.info

10. CONCLUSION

11. ACKNOWLEDGEMENTS

This is joint work. The implementation of the first cell store on top of a document store has been made as a team effort by Matthias Brantner, William Candillon, Federico Cavalieri, Dennis Knochenwefel, Alexander Kreutz and myself. We received a lot of very useful advice from Charles Hoffman.