Cell Stores

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

ABSTRACT

false

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 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

4.2 Hypercubes

4.3 Spreadsheet views

5. CANONICAL MAPPING TO THE RELATIONAL MODEL

6. STANDARDIZED DATA INTERCHANGE: XBRL

Facts, taxonomies
Concepts, dimensions, members, hypercubes
Formulas
Table linkbases

7. IMPLEMENTATION

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.