 **CLOUD**Graph™

CloudGraph Design Team  
TerraMeta Software, Inc.

Architecture Overview

**Revision History**

| **Revision** | **Date** | **Description** | **Author** |
| --- | --- | --- | --- |
| 1.0 | January 0, 2012 | CloudGraph architecture description | CloudGraph Design Team |

| Table of Contents |
| --- |
| [Architecture Overview i](#_Toc332371599)  [1 Introduction 4](#_Toc332371600)  [**1.1** **Overview** 4](#_Toc332371601)  [1.1.1 Byte Array Conversion 4](#_Toc332371602)  [1.1.2 Data Graph Assembly 4](#_Toc332371603)  [1.1.3 Data Graph State 4](#_Toc332371604)  [1.1.4 Row Key Generation 4](#_Toc332371605)  [1.1.5 Graph Slice Query 4](#_Toc332371606)  [**1.2** **Diagrams** 5](#_Toc332371607) |

| Table of Figures |
| --- |
| **No table of figures entries found.** |

# **Introduction**

## **Overview**

Distributed “cloud” databases allow for a new level of scalability and flexibility. These new sparse, columnar databases support a practically unlimited number of columns within a single table, and managing hundreds or even thousands of column name/qualifiers within a client application can be a significant challenge.

CloudGraph provides services and infrastructure to impose the structure of your business domain model, regardless of its complexity, as a service layer over various supported big-table style “cloud” databases, such as HBase and Cassandra. Application developer’s deal with higher-level typed structures with meaning within the application domain, rather than row and column qualifiers and values, typically manipulated as un-typed Java byte arrays.

Numerous best practices have evolved out of the HBase and Hadoop open-source software ecosystem. Several are quite restrictive such as the support for ACID transactions only across a single HBase row. Other critical best practices involve the use of column families and in particular the format and length of composite row and column keys, as these all can effect the general performance and especially the even distribution of data across regions in an HBase cluster.

The HGraph implementation encapsulates many HBase best practices in each of these areas and provides the user with an intuitive convenient and standards-based API generated from a user-provided,

domain-specific business model. Complexities of terse and efficient physical row and column key generation are completely hidden and the client user is provided with a meaningful view of his/her business entities and attributes.

Imagine taking an average sized relational database composed of 30-40 tables and 200-300 columns and compressing this into a single tabular structure such as a spreadsheet. This is the type of challenge we face as we leverage the new sparse, columnar, distributed or “cloud” databases such as Apache HBase and Cassandra.

Relational database design practices have long taught us to subdivide our business domains into meaningful entities and to add attributes that describe each entity within the business context. The need for meaningful business entities exists regardless of the capabilities or structure of a particular data store.

# **Design**

## **Overview**

### Byte Array Conversion

CloudGraph handles the conversion and formatting of byte arrays, required by most cloud database API’s, to and from standard Java primitive and other data types specified in your business model. CloudGraph formats dates, timestamps as well as primitive types for minimal storage footprint within the target data store.

### Data Graph Assembly

CloudGraph assembles complex graph structures from low level cloud database column qualifier-value pairs according to a given business domain model.

### Data Graph State

### Row Key Generation

### Graph Slice Query

Table 1 - Example Enterprise Java™ Project Software Layers

| Software Layer |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

.

## **Diagrams**