ISIT312 Big Data Management

HBase Data Model

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

HBase Data Model

Outline

- Background
- Logical view of data
- Design fundamentals
- Physical implementation
- Performance

Background

Hbase is open source distributed database based on a data model of Google's BigTable

HBase provides a BigTable view of data stored in HDFS

HBase is also called as Hadoop DataBase

HBase still provides a tabular view of data however it is also very different from the traditional relational data model

HBase data model is a sparse, distributed, persistent multidimensional sorted map

It is indexed by a row key, column key, and timestamp

HBase Data Model

Outline

Logical view of data

HBase organizes data into tables

HBase table consists of rows

Each row is uniquely identified by a row key

Data within a row is grouped by a column family

Column families have an important impact on the physical implementation of HBase table

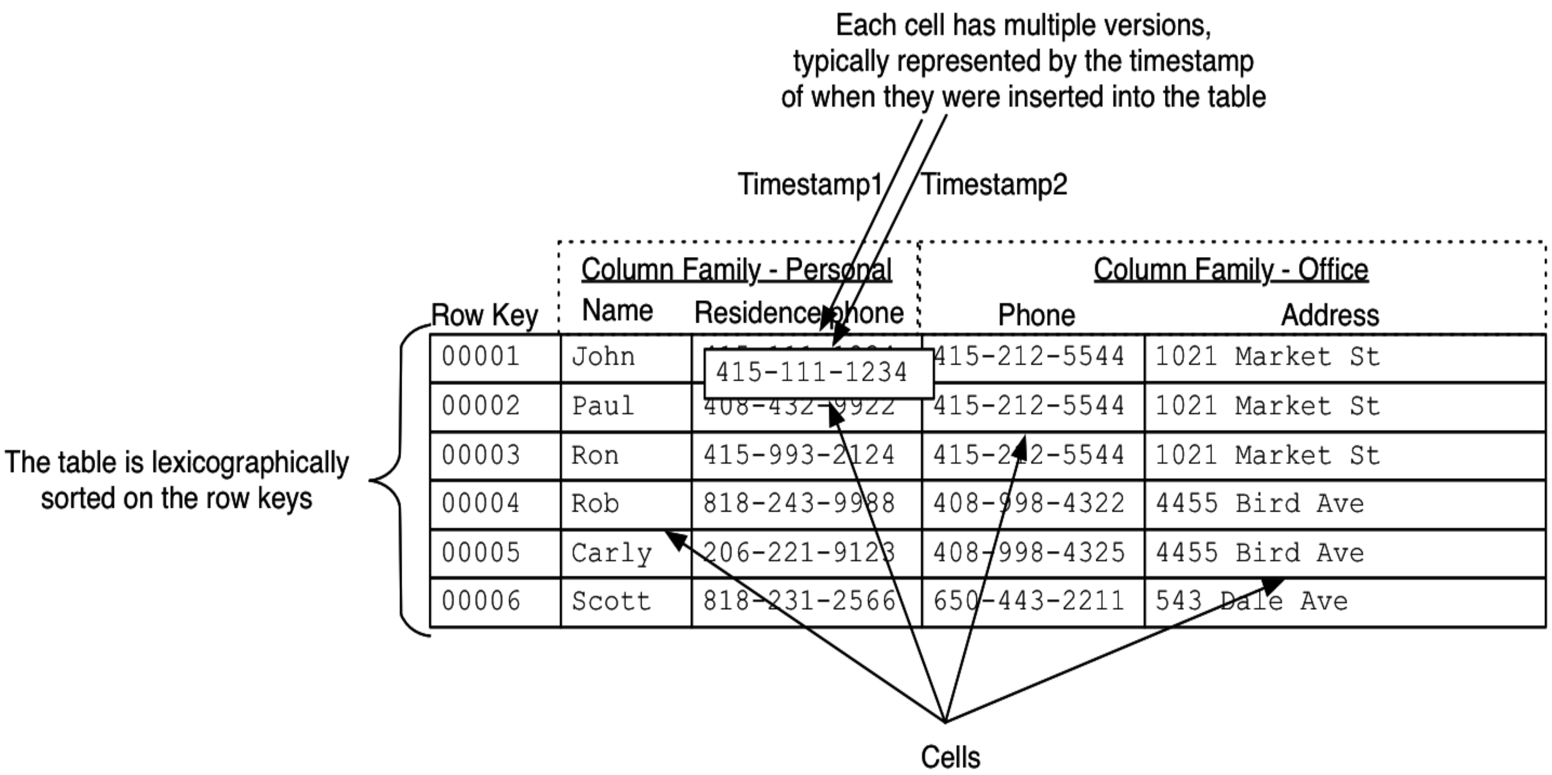
Every row has the same column families although some of them can be empty

Data within a column family is addressed via its column qualifier, or simply, column name

Hence, a combination of row key, column family, and column qualifier uniquely identifies a cell

Values in cells do not have a data type and are always treated as sequences of bytes

Logical view of data



Values within a cell have multiple versions

Versions are identified by their version number, which by default is a timestamp when the cell was written

Logical view of data

If a timestamp is not determined at write time, then the current timestamp is used

If a timestamp is not determined during a read, the latest one is returned

The maximum allowed number of cell value versions is determined for each column family

The default number of cell versions is three

Logical view of data

A view of HBase table as a nested structure

HBase Table
```
{"Row-0001":
            {"Home":
                    {"Name":
                            {"timestamp-1":"James"}
                     "Phones":
                              {"timestamp-1":"2 42 214339"
                               "timestamp-2":"2 42 213456"
                               "timestamp-3":"+61 2 4567890"}
                    }
             "Office":
                      {"Phone":
                               {"timestamp-4":"+64 345678"}
                       "Address":
                                 {"timestamp-5":"10 Ellenborough Pl"}
                      }
            }
}
```

Logical view of data

A view of HBase table as a nested structure

HBase Table

```
{"Row-0002":
            {"Home":
                    {"Name":
                            {"timestamp-6":"Harry"}
                     "Phones":
                              {"timestamp-7":"2 42 214234"}
                    }
             "Office":
                      {"Phone":
                               {"timepstamp-8":"+64 345678"}
                       "Address":
                                 {"timestamp-9":"10 Bong Bong Rd"
                                  "timestamp-10":"23 Victoria Rd"}
                      }
            }
}
```

Logical view of data

A view HBase table as a key->value store

```
"Row-0001"->{"Home":{"Name":{"timestamp-1":"James"}
                             "Phones":{"timestamp-2":"2 42 214339"
                                       "timestamp-3":"2 42 213456"
                                       "timestamp-4":"+61 2 4567890"}
                             }
                    "Office":{"Phone":{"timestamp-5":"+64 345678"}
                              "Address":{"timestamp-6":"10 Ellenborough Pl"}
                              }
                    }
```

Key->value

```
"Row-0001" "Home"->{"Name":{"timestamp-1":"James"}
                            "Phones":{"timestamp-2":"2 42 214339"
                                      "timestamp-3":"2 42213456"
                                      "timestamp-4":"+61 2 4567890"}
                            }
```

Key->value"

```
"Row-0001" "Home" "Phones"->{"timestamp-2":"2 42 214339"
                             "timestamp-3":"2 42 213456"
                                    "timestamp-4":"+61 2 4567890"}
```

Key->value

Logical view of data

A key can be row key or a combination of a row key, column family, qualifier, and timestamp depending on what supposed to be retrieved

If all the cells in a row are of interest then a key is a row key

If only specific cells are of interest, the appropriate column families and qualifiers are a part of a key

HBase Data Model

Outline

- Background
- Logical view of data
- Design fundamentals
- Physical implementation
- Performance

Design Fundamentals

When designing Hbase table we have to consider the following questions:

- What should be a row key and what should it contain?
- How many column families should a table have?
- What columns (qualifiers) should be included in each column family?
- What information should go into the cells?
- How many versions should be stored for each cell?

In fact HBase table is a four level hierarchical structure where a table consists of rows, rows consists of column families, column families consist of columns and columns consists of versions

If cells contain the keys then HBase table becomes a network/graph structure

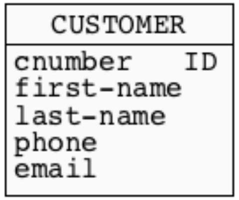
Design Fundamentals

Important facts to remember:

- Indexing is performed only for a row key
- Hbase tables are stored sorted based on a row key
- Everything in Hbase tables is stored as untyped sequence of bytes
- Atomicity is guaranteed only at a row level and there are no multi-row transactions
- Column families must be defined at Hbase table creation time
- Column qualifiers are dynamic and can be defined at write time
- Column qualifiers are stored as sequences of bytes such that they can represent data

Design Fundamentals

Implementation of Entity type



HBase Table

```
{"007":
        {"CUSTOMER":
                    {"first-name": {"timestamp-1":"James"}
                     "last-name": {"timestamp-2":"Bond"}
                     "phone": {"timestamp-1":"007"}
                    }
        }
}
```

Design Fundamentals

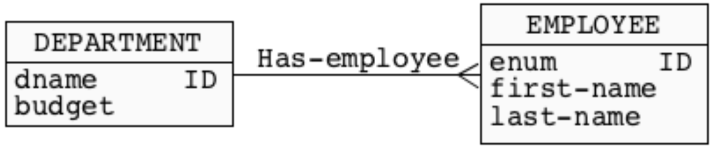
Implementation of one-to-one relationship

HBase Table

```
{"Sales":
        {"DEPARTMENT":
                    {"budget": {"timestamp-1":"1000"}
                    }
        }
        {"MANAGER":
                 {"enumber": {"timestamp-2":"007"}
                  "first-name": {"timestamp-3":"James"}
                  "last-name": {"timestamp-4":"Bond"}
                 }
        }
}
```

Design Fundamentals

Implementation of one-to-many relationship

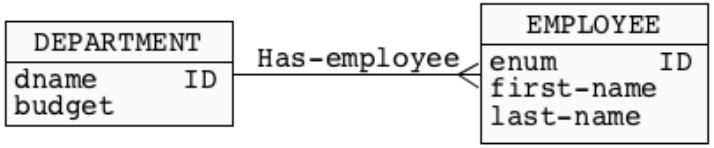


HBase Table

```
{"007":
       {"EMPLOYEE":
                   {"enumber": {"timestamp-1":"007"}
                    "first-name": {"timestamp-2":"James"}
                    "last-name": {"timestamp-3":"Bond"}
                    "department":{"timestamp-4":"Sales"}
                   }
       }
}
```

Design Fundamentals

Another implementation of one-to-many relationship

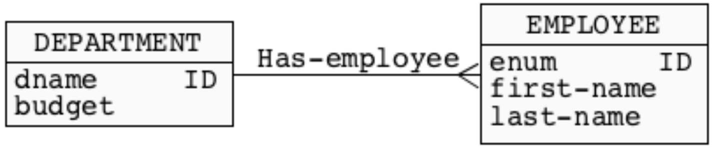


HBase Table

```
{"Sales":
       {"DEPARTMENT":
                     {"budget": {"timestamp-1":"1234567"}
                     }
       }
       {"EMPLOYEE":
                   {"007": {"timestamp-2":"James Bond"}
                    "008": {"timestamp-3":"Harry Potter"}
                    "009": {"timestamp-4":"Robin Hood" }
                    ...
                   }
       }
}
```

Design Fundamentals

Yet another implementation of one-to-many relationship

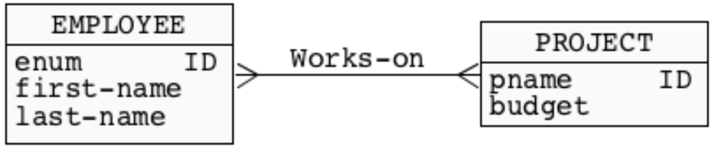


HBase Table

```
{"Sales":
        {"DEPARTMENT":
                    {"budget": {"timestamp-1":"1000"}
        }
        {"HAS-EMPLOYEES":
                       {"employees": {"timestamp-2":"007 James Bond"}
                                     {"timestamp-3":"008 Harry Potter"}
                                     {"timestamp-3":"009 Robin Hood"}
                                     ...
                       }
        }
}
```

Design Fundamentals

Implementation of many-to-many relationship

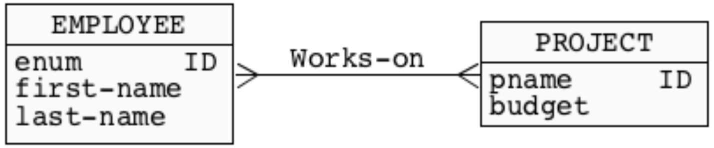


HBase Table

```
{"participation-1":
                   {"EMPLOYEE":
                               {"enumber": {"timestamp-1":"007"}
                                "first-name": {"timestamp-2":"James"}
                                "last-name": {"timestamp-3":"Bond"}
                                "pnumber": {"timestamp-4":"project-1"}
                                           {"timestamp-5":"project-2"}
                                             ...
                               }
                  }
}
```

Design Fundamentals

Another implementation of many-to-many relationship

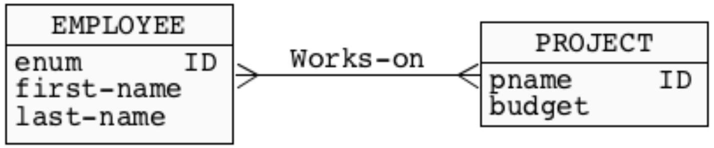


HBase Table

```
{"participation-1":
                    {"PROJECT":
                               {"pnumber": {"timestamp-1":"project-1"}
                                "budget:": {"timestamp-2":"12345.25"}
                                "employee": {"timestamp-3":"007"}
                                            {"timestamp-4":"008"}
                                            {"timestamp-5":"009"}
                                             ...
                               }
                    }
}
```

Design Fundamentals

Another implementation of many-to-many relationship



HBase Table

```
{"participation-1":
                 {"PARTICIPATION":
                                {"pnumber": {"timestamp-1":"project-1"}
                                 "employee": {"timestamp-2":"employee-007"}
                                }
                 }
}
```

HBase Table

```
{"participation-2":
                 {"PARTICIPATION":
                                {"pnumber": {"timestamp-1":"project-1"}
                                 "employee": {"timestamp-2":"employee-008"}
                                }
                 }
}
```

Design Fundamentals

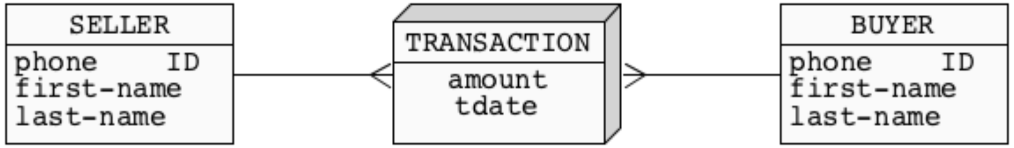
Note, that it is possible to group in one Hbase table rows of different types

HBase Table

```
{"employee-007":
                 {"EMPLOYEE":
                              {"enumber": {"timestamp-1":"007"
                               "first-name": {"timestamp-2":"James"}
                               "last-name": {"timestamp-3":"Bond"}
                              }
                 }
}
{"project-1":
              {"PROJECT":
                          {"pnumber": {"timestamp-4":"1"}
                           "budget": {"timestamp-5":"12345.25"}
                                     }
                    }
}
{"participation-2":
                    {"PARTICIPATION":
                                      {"pnumber": {"timestamp-1":"project-1"}
                                       "employee": {"timestamp-2":"employee-007"}
                                      }
                    }
}
```

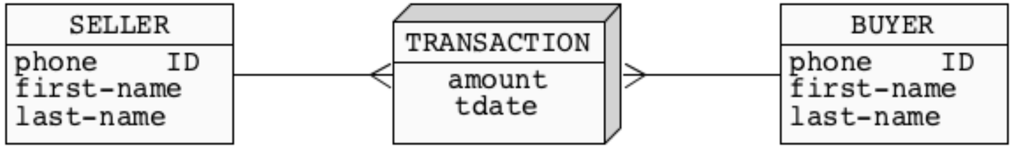
Design Fundamentals

Implementation of fact with dimensions



Design Fundamentals

Implementation of fact with dimensions

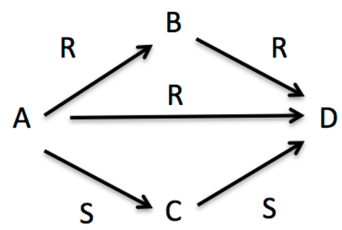


HBase Table

```
{"1234567":
            {"MEASURE":
                        {"amount": {"timestamp-1":"1000000"}
                        }
             "BUYER":
                        {"phone": {"timestamp-1":"242214339"}
                         "first-name": {"timestamp-1":"James"}
                         "last-name": {"timestamp-1":"Bond"}
                        }
             "SELLER":
                        {"phone": {"timestamp-1":"242215612"}
                         "first-name": {"timestamp-1":"Harry"}
                         "last-name": {"timestamp-1":"potter"}
                        }
            }
}
```

Design Fundamentals

Implementation of graph structure



HBase Table

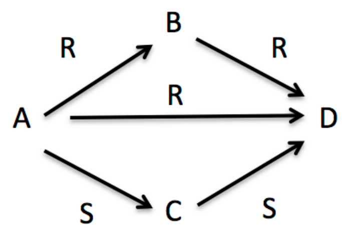
```
{"A":
     {"R":
          {"1":{"timestamp-1":"B"}
           "2":{"timestamp-1":"D"}
          }
      "S":
          {"1":{"timestamp-1":"C"}
          }
     }
}
```

HBase Table

```
{"B":
     {"R":
          {"1":{"timestamp-1":"D"}
          }
     }
}
```

Design Fundamentals

Implementation of graph structure



HBase Table

```
{"C":
     {"S":
          {"1":{"timestamp-1":"D"}
          }
     }
}
```

HBase Table

```
{"D":
}
```

HBase Data Model

Outline

Physical implementation

HBase is a database built on top of HDFS

HBase tables can scale up to billions of rows and millions of columns

Because Hbase tables can grow up to terabytes or even petabytes, Hbase tables are split into smaller chunks of data that are distributed across multiple servers

Chunks of data are called as regions and servers that host regions are called as region servers

Region servers are usually collocated with data nodes of HDFS

The splits of Hbase tables are usually horizontal, however, it is also possible to benefit from vertical splits separating column families

Region assignments happen when Hbase table grows in size or when a region server is malfunctioning or when a new region server is added

Physical implementation

Two special HBase tables `-ROOT-` and `.META.` keep information where the regions for the tables are hosted

A table `-ROOT-` never splits into more than one region, while `.META.` can be distributed over many regions

When a client application wants to access a particular row -ROOT- points it to the region of the `.META.` table that contains information about a row location

The entry point for an HBase system is provided by another system called ZooKeeper

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services

HBase Data Model

Outline

Performance

Too many regions affect performance

More regions means smaller memory flushes to persistent storage and smaller HFiles stored in HDFS

It requires HBase to process many compaction to keep the number of HFiles low

HBase can handle regions from 10 to 40 Gb

Too many regions may occur due to

- Over-splitting with HBase's split feature
- Improper presplitting with HBase presplit feature

Offline merging regions can be used to reduce number of regions

Performance

Too many column families affect performance

A magic total number of column families number is 3

Column families are built to group data with similar format or similar access pattern

Consequences of too many column families

- Memory: HBase shares write cache among all column families of the same regions, more column families means less transient memory for each one of them
- Compactions: the total number of column families affects the total number of store files create during flushes and subsequently number of compactions
- Split: HBase stores column families into separate files and directories, when a directory becomes too large it is split; split affects all column families, if some of them are small only few cells are stored in a directory

A solution is a correct schema design followed by delete column family, merge column family, or separate column family in a new HBase table

Performance

All writes and reads should be uniformly distributed across the regions

Hotspotting occurs when a given region serviced by a single Region Server receives most or all of the read or write requests

Causes of hotspotting:

- Monotonically incrementing keys: only last bits or bytes are slowly changin, e.g timestamp used as a key, all write to a consecutive area of keys go to the same region
- Poorly distributed keys: e.g keys are always appended with the same prefix going to the same region
- Small reference (lookup) tables: All reference to lookup tables must be handled by the same region server
- Application issues: e.g. writes always performed on the same region
- Meta region hotspotting: e.g creating too many connections

References

Kerzner M., Maniyam S., HBase Design Patterns, Packt Publishing 2014
(Available from UoW Library)

Jiang, Y. HBase Adminstration Cookbook, Pack Publishing, 2012
(Available from UoW Library)

Dimiduk N., Khurana A., HBase in Action, Mannig Publishers, 2013

Spaggiari J-M., O'Dell K., Architecting HBase Applications, O'Reilly, 2016