NoSQL Databases

Advanced Databases

Enric Biosca Trias ebiosca@maia.ub.es

Dept. Matemàtica Aplicada i Anàlisi.

Universitat de Barcelona

Introduction

- NoSQL databases in context of cloud computing
- Relaxing ACID properties
- NoSQL databases
  - Categories of NoSQL databases
  - Typical NoSQL API
  - Representatives of NoSQL databases
- Conclusions

Cloud computing, cloud databases

- Cloud computing
  - data intensive applications on hundreds of thousands of commodity servers and storage devices
  - basic features:
    - elasticity,
    - fault-tolerance
    - automatic provisioning
- Cloud databases: traditional scaling up (adding new expensive big servers) is not possible
  - requires higher level of skills
  - is not reliable in some cases
- Architectural principle: scaling out (or horizontal scaling) based on data partitioning, i.e. dividing the database across many (inexpensive) machines

Cloud computing, cloud databases

- Technique: data sharding, i.e. horizontal partitioning of data (e.g. hash or range partitioning)
- Consequences:
  - manage parallel access in the application
  - scales well for both reads and writes
  - not transparent, application needs to be partition-aware

Relaxing ACID properties

- Cloud computing: ACID is hard to achieve, moreover, it is not always required, e.g. for blogs, status updates, product listings, etc.
- Availability
  - Traditionally, thought of as the server/process available 99.999 % of time
  - For a large-scale node system, there is a high probability that a node is either down or that there is a network partitioning
- Partition tolerance
  - ensures that write and read operations are redirected to available replicas when segments of the network become disconnected

Eventual Consistency

- Eventual Consistency
  - When no updates occur for a long period of time, eventually all updates will propagate through the system and all the nodes will be consistent
  - For a given accepted update and a given node, eventually either the update reaches the node or the node is removed from service
- BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) properties, as opposed to ACID
    - Soft state: copies of a data item may be inconsistent
    - Eventually Consistent – copies becomes consistent at some later time if there are no more updates to that data item
    - Basically Available – possibilities of faults but not a fault of the whole system

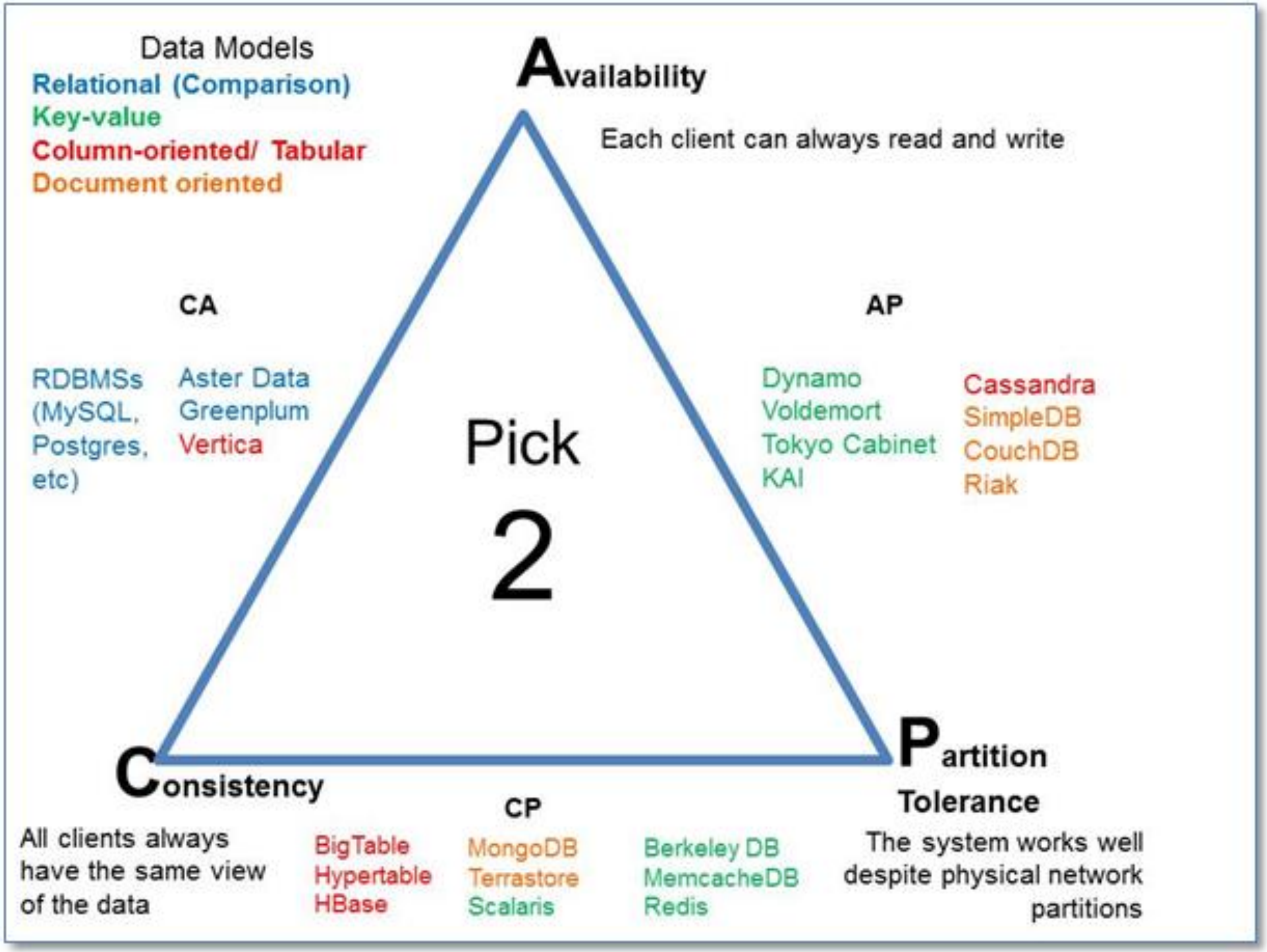
CAP Theorem

- Suppose three properties of a system
  - Consistency (all copies have same value)
  - Availability (system can run even if parts have failed)
  - Partitions (network can break into two or more parts, each with active systems that can not influence other parts)
- Brewer’s CAP “Theorem”: for any system sharing data it is impossible to guarantee simultaneously all of these three properties
- Very large systems will partition at some point
  - it is necessary to decide between C and A
  - traditional DBMS prefer C over A and P
  - most Web applications choose A (except in specific applications such as order processing)

CAP Theorem

- Drop A or C of ACID
  - relaxing C makes replication easy, facilitates fault tolerance,
  - relaxing A reduces (or eliminates) need for distributed concurrency control.

CAP theorem



Data Models

Relational (Comparison)

Key-value

Column-oriented/ Tabular

Document oriented

Availability

Each client can always read and write

CA

RDBMSs (MySQL, Postgres, etc)

Aster Data
Greenplum
Vertica

AP

Dynamo
Voldemort
Tokyo Cabinet
KAI

Cassandra
SimpleDB
CouchDB
Riak

Pick 2

Consistency

All clients always have the same view of the data

CP

BigTable
Hypertable
HBase

MongoDB
Terrastore
Scalaris

Berkeley DB
MemcacheDB
Redis

Partition Tolerance

The system works well despite physical network partitions

NoSQL databases

- The name stands for **N**ot **O**nly **SQL**
- Common features:
  - non-relational
  - usually do not require a fixed table schema
  - horizontal scalable
  - mostly open source
- More characteristics
  - relax one or more of the ACID properties (see CAP theorem)
  - replication support
  - easy API (if SQL, then only its very restricted variant)
- Do not fully support relational features
  - no join operations (except within partitions),
  - no referential integrity constraints across partitions.

Categories of NoSQL databases

- key-value stores
- column NoSQL databases
- document-based
- XML databases (myXMLDB, Tamino, Sedna)
- graph database (neo4j, InfoGrid)

Categories of NoSQL databases

- key-value stores
- column NoSQL databases
- document-based
- XML databases (myXMLDB, Tamino, Sedna)
- graph database (neo4j, InfoGrid)

Key-Value Data Stores

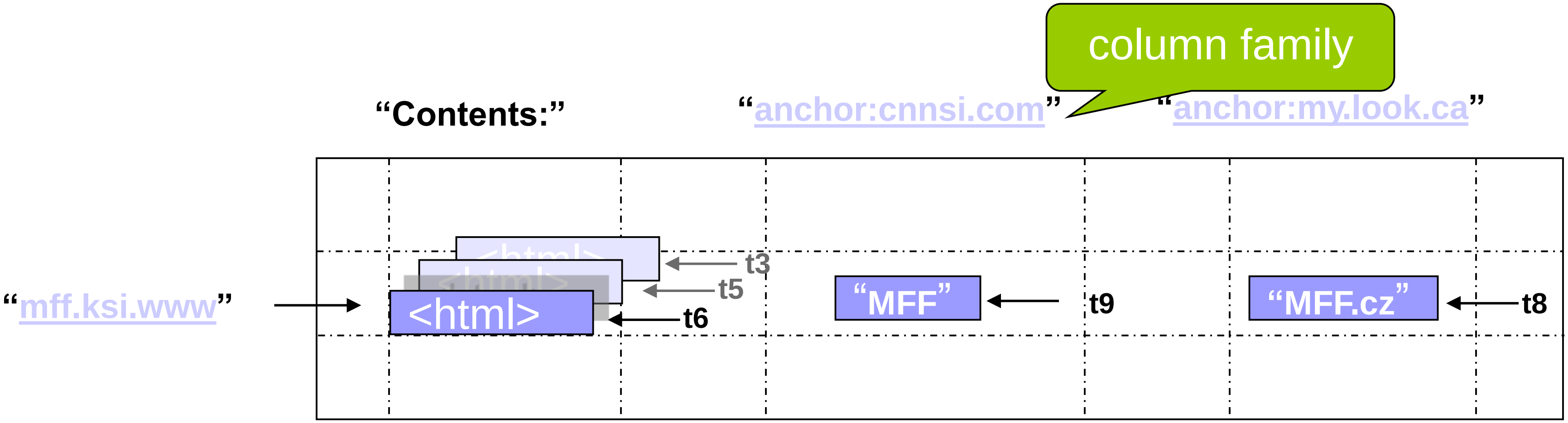
- Example: SimpleDB
  - Based on Amazon’s Single Storage Service (S3)
  - items (represent objects) having one or more pairs (name, value), where name denotes an attribute.
  - An attribute can have multiple values.
  - items are combined into domains.

Column-oriented*

- store data in column order
- allow key-value pairs to be stored (and retrieved on key) in a massively parallel system
  - data model: families of attributes defined in a schema, new attributes can be added
  - storing principle: big hashed distributed tables
  - properties: partitioning (horizontally and/or vertically), high availability etc. completely transparent to application

* Better: extendible records

Column-oriented



- Example: BigTable
  - indexed by row key, column key and timestamp. i.e. (row: string , column: string , time: int64) → String.
  - rows are ordered in lexicographic order by row key.
  - row range for a table is dynamically partitioned, each row range is called a tablet.
  - columns: syntax is family:qualifier

A table representation of a row in BigTable

Row key	Time stamp	Column name	Column family Grandchildren		
http://ksi....	t1	"Jack"	"Claire" 7		
	t2	"Jack"	"Claire" 7	"Barbara" 6	
	t3	"Jack"	"Claire" 7	"Barbara" 6	"Magda" 3

Column-oriented

- Example: Cassandra
  - keyspace: Usually the name of the application; e.g., 'Twitter', 'Wordpress‘.
  - column family**:** structure containing an unlimited number of rows
  - column**:** a tuple with name, value and time stamp
  - key**:** name of record
  - super column**:** contains more columns

Document-based

- based on JSON format: a data model which supports lists, maps, dates, Boolean with nesting
- Really: *indexed* semistructured documents
- Example: Mongo
  - {Name:"Jaroslav",

    Address:"Malostranske nám. 25, 118 00 Praha 1“

    Grandchildren: [Claire: "7", Barbara: "6", "Magda: "3", "Kirsten: "1", "Otis: "3", Richard: "1"]

    }

Typical NoSQL API

- Basic API access:
  - get(key) -- Extract the value given a key
  - put(key, value) -- Create or update the value given its key
  - delete(key) -- Remove the key and its associated value
  - execute(key, operation, parameters) -- Invoke an operation to the value (given its key) which is a special data structure (e.g. List, Set, Map .... etc).

NoSQL databases - Key-Valued

Name	Producer	Data model	Querying
SimpleDB	Amazon	set of couples (key, {attribute}), where attribute is a couple (name, value)	restricted SQL; select, delete, GetAttributes, and PutAttributes operations
Redis	Salvatore Sanfilippo	set of couples (key, value), where value is simple typed value, list, ordered (according to ranking) or unordered set, hash value	primitive operations for each value type
Dynamo	Amazon	like SimpleDB	simple get operation and put in a context
Voldemort	LinkeId	like SimpleDB	similar to Dynamo

NoSQL databases Column-Oriented

Name	Producer	Data model	Querying
BigTable	Google	set of couples (key, {value})	selection (by combination of row, column, and time stamp ranges)
HBase	Apache	groups of columns (a BigTable clone)	JRUBY IRB-based shell (similar to SQL)
Hypertable	Hypertable	like BigTable	HQL (Hypertext Query Language)
CASSANDRA	Apache (originally Facebook)	columns, groups of columns corresponding to a key (supercolumns)	simple selections on key, range queries, column or columns ranges
PNUTS	Yahoo	(hashed or ordered) tables, typed arrays, flexible schema	selection and projection from a single table (retrieve an arbitrary single record by primary key, range queries, complex predicates, ordering, top-k)

NoSQL databases document-based

Name	Producer	Data model	Querying
MongoDB	10gen	object-structured documents stored in collections; each object has a primary key called ObjectId	manipulations with objects in collections (find object or objects via simple selections and logical expressions, delete, update,)
Couchbase	Couchbase[1]	document as a list of named (structured) items (JSON document)	by key and key range, views via Javascript and MapReduce

Conclusions (1)

- NoSQL database cover only a part of data-intensive cloud applications (mainly Web applications).
- Problems with cloud computing:
  - SaaS applications require enterprise-level functionality, including ACID transactions, security, and other features associated with commercial RDBMS technology, i.e. NoSQL should not be the only option in the cloud.
  - Hybrid solutions:
    - Voldemort with MySQL as one of storage backend
    - deal with NoSQL data as semistructured data

      ⇒ integrating RDBMS and NoSQL via SQL/XML

Conclusions (2)

- next generation of highly scalable and elastic RDBMS: NewSQL databases (from April 2011)
  - they are designed to scale out horizontally on shared nothing machines,
  - still provide ACID guarantees,
  - applications interact with the database primarily using SQL,
  - the system employs a lock-free concurrency control scheme to avoid user shut down,
  - the system provides higher performance than available from the traditional systems.
- Examples: MySQL Cluster (most mature solution), VoltDB, Clustrix, ScalArc, …

Conclusions (3)

- New buzzword: SPRAIN – 6 key factors for alternative data management:
  - Scalability
  - Performance
  - relaxed consistency
  - Agility
  - Intricacy (complexitat)
  - Necessity
- Conclusion in conclusions: These are exciting times for Data Management research and development

Introduction

Chapter 1

Chapter Topics

Introduct to HBase

- What is HBase?
- Strengths of HBase
- HBase in Product i on
- Weaknesses of HBase
- Essent i al Points

What is HBase?

- **HBase is a NoSQL database that runs on top of HDFS**
- **HBase is…**
  - Highly available and fault tolerant
  - Very scalable, and can handle high throughput
  - Able to handle massive tables
  - Well suited to sparse rows where the number of columns varies
  - An open-source, Apache project
- **HDFS provides:**
  - Fault tolerance
  - Scalability

Google BigTable

- **HBase is based on Google's BigTable**
- **The Google use case is to search the entire Internet**
  - BigTable is at the center of how the company accomplishes this
- **Engineering considerations for searching the Internet:**
  - How do you efficiently download the Web pages?
  - How do you store the entire Internet?
  - How do you index the Web pages?
  - How do you efficiently search the Web pages?

Hbase Principal value?

- **HBase helps solve data access issues where random access is required**
- **HBase scales easily, making it ideal for Big Data storage and processing needs**
- **Columns in an HBase table are defined dynamically, as required**

HBase Usage Scenarios (1)

- **High capacity**
  - Massive amounts of data
    - Hundreds of gigabytes, growing to terabytes or even petabytes
  - Example: Storing the entire Internet
- **High read and write throughput**
  - 1000s/second per node
  - Example: Facebook Messages
    - 75 billion operations per day
    - Peak usage of 1.5 million operations per second
    - Runs entirely on HBase

HBase Usage Scenarios (2)

- Scalable in-memory caching

When To Use HBase

Chapter Topics

Introduction to HBase

- What is HBase?
- Strengths of HBase
-
- Weaknesses of HBase
- Essential Points

UNIVERSITAT DE BARCELONA

HBase In Production

- Many enterprises are using HBase in production

http://wiki.apache.org/hadoop/Hbase/PoweredBy

HBase Use Cases

Messaging

Simple Entities

Graph Data

Metrics

HBase Use Case Characteristics

Messaging

Simple Entities

Graph Data

Metrics

Chapter Topics

Introduction to HBase

- What is HBase?
- Strengths of HBase
- HBase in Product i on
-
- Essent i al Points

Features Missing from HBase

- HBase does not provide certain popular RDBMS features

Different Design Approach in HBase

- Designing an HBase application requires developers to engineer the

Chapter Topics

Introduction toHBase

- What is HBase?
- Strengths of HBase
- HBase in Product i on
- Weaknesses of HBase
-

UNIVERSITAT DE BARCELONA

Essential Points

- There are many technological challenges when dealing with Big Data

HBase Tables

Chapter 3

Course Chapters

- Introduction
- Introduction to Hadoop and HBase
- **HBase Tables**
- HBase Shell
- HBase Architecture Fundamentals
- HBase Schema Design
- Basic Data Access with the HBase API
- More Advanced HBase API Features
- HBase on the Cluster
- HBase Reads and Writes
- HBase Performance Tuning
- HBase Administration and Cluster Management
- HBase Replication and Backup
- Using Hive and Impala with HBase
- Conclusion
- Appendix: Using Python and Thrift to Access HBase Data
- Appendix: OpenTSDB

HBase Tables

In this Chapter you will learn

Chapter Topics

HBase Tables

-
- HBase Table Fundamentals
- Thinking About Table Design
- Hands-On Exercise: HBase Data Import
- Essential Points

HBase Terms

Fundamental HBase Concepts

- HBase stores data in tables
  -
-
  -

UNIVERSITAT DE BARCELONA

What is a Table?

- HBase tables are comprised of rows, columns, and column families
-
-
-
-

Chapter Topics

HBase Tables

- HBase Concepts
-
- Thinking About Table Design
- Hands-On Exercise: HBase Data Import
- Essential Points

About HBase Tables

- HBase is essentially a distributed, sorted map

Example Application Data (1)

- [illegible] examples, we will use a table that holds

UNIVERSITAT DE BARCELONA

Example Application Data (2)

- Not every field will have a value

Rows

- Tables are comprised of rows, columns, and column families

Columns

- Columns hold the data for the table

HBase Columns and Column Families

- All columns belonging to the same column family have the same prefix

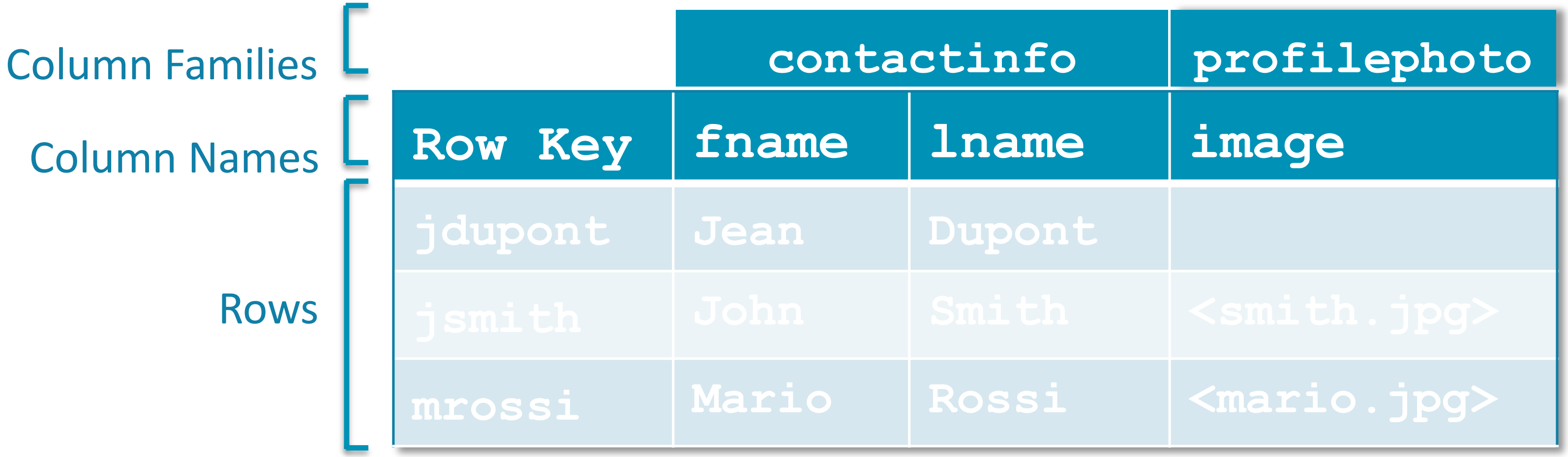
HBase Tables: A Conceptual View

- A column is referenced using its column family and column name (or

-

  -

  -



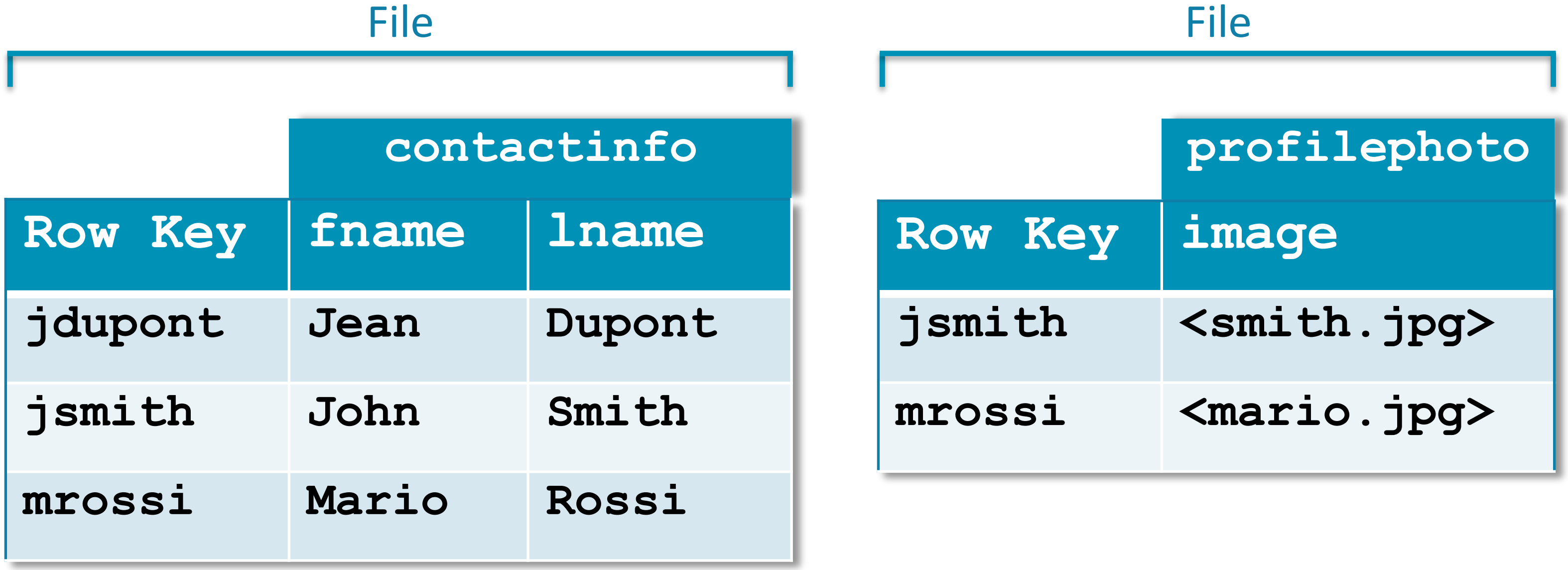
	contactinfo		profilephoto
Row Key	fname	lname	image
jdupont	Jean	Dupont	
jsmith	John	Smith	<smith.jpg>
mrossi	Mario	Rossi	<mario.jpg>

Storing Data in Tables

- Data in HBase tables is stored as byte arrays

Table Storage On Disk: Basics

- ally stored on disk on a per-column family basis
-



File

	contactinfo	
Row Key	fname	lname
jdupont	Jean	Dupont
jsmith	John	Smith
mrossi	Mario	Rossi

File

	profilephoto
Row Key	image
jsmith	<smith.jpg>
mrossi	<mario.jpg>

Data Storage Within a Column Family

Sorted by Row Key and Column

Row Key	Column	Timestamp	Cell Value
jdupont	contactinfo:fname	1273746289103	Jean
jdupont	contactinfo:lname	1273878447049	Dupont
jsmith	contactinfo:fname	1273516197868	John
jsmith	contactinfo:lname	1273871824184	Smith
mrossi	contactinfo:fname	1273616297446	Mario
mrossi	contactinfo:lname	1273971921442	Rossi

HBase Operations (1)

- All rows in HBase are identified by a row key

HBase Operations (2)

- [illegible] ks data as having been deleted

Chapter Topics

HBase Tables

- HBase Concepts
- HBase Table Fundamentals
-
- Hands-On Exercise: HBase Data Import
- Essential Points

Row Key is the Only Indexed Column

- RDBMS can have as many index columns as required

Features Comparison: RDBMS vs. HBase

	RDBMS	HBase
Data layout	Row- or column-oriented	Column family-oriented
Transactions	Yes	Single row only
Query language	SQL	get/put/scan
Security	Authentication/ Authorization	Access control at per-cell level, also at cluster, table, or row level
Indexes	Yes	Row key only
Max data size	TBs	PB+
Read/write throughput limits	1000s queries/second	Millions of queries/second

Replacing RDBMSs with HBase

- Replacing an RDBMS-based applicaCon with HBase requires significant re-

Comparison with RDBMS Table Design

HBase vs. RDBMS Schema Design

- Goals for designing an RDBMS schema

Chapter Topics

HBase Tables

- HBase Concepts
- HBase Table Fundamentals
- Thinking About Table Design
- Hands-On Exercise: HBase Data Import
-

Essential Points

- Tables are comprised of rows, columns, and column families
-

HBase Shell

Chapter 4

Course Chapters

- Introduction
- Introduction to Hadoop and HBase
- HBase Tables
- **HBase Shell**
- HBase Architecture Fundamentals
- HBase Schema Design
- Basic Data Access with the HBase API
- More Advanced HBase API Features
- HBase on the Cluster
- HBase Reads and Writes
- HBase Performance Tuning
- HBase Administration and Cluster Management
- HBase Replication and Backup
- Using Hive and Impala with HBase
- Conclusion
- Appendix: Using Python and Thrift to Access HBase Data
- Appendix: OpenTSDB

HBase Shell

In this chapter you will learn

Chapter Topics

HBase Shell

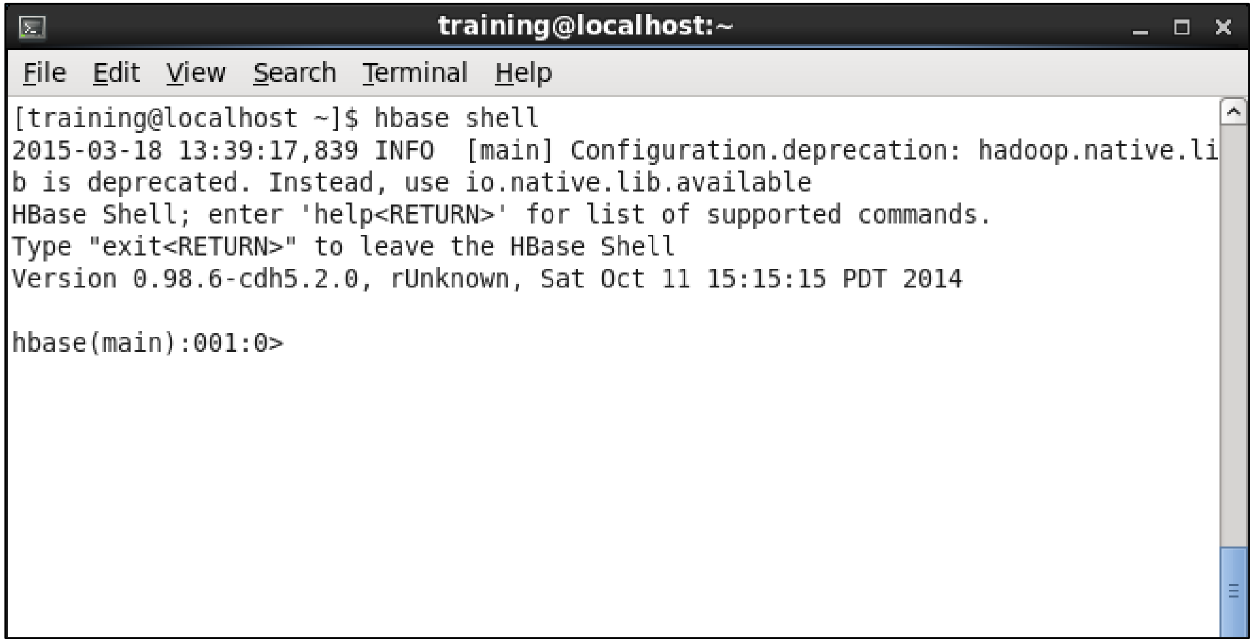
-
- Working with Tables
- Hands-On Exercise: Using the HBase Shell
- Working with Table Data
- Hands-On Exercise: Data Access in the HBase Shell
- Essential Points

HBase Shell Basics

- The HBase Shell is an interactive shell for sending commands to HBase

Running the HBase Shell

```
$ hbase shell
```



```
[training@localhost ~]$ hbase shell
2015-03-18 13:39:17,839 INFO  [main] Configuration.deprecation: hadoop.native.li
b is deprecated. Instead, use io.native.lib.available
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.98.6-cdh5.2.0, rUnknown, Sat Oct 11 15:15:15 PDT 2014

hbase(main):001:0>
```

Basic Commands

```
hbase> help
```

```
hbase> status
3 servers, 0 dead, 1.3333 average load
```

```
hbase> version
0.98.6-cdh5.2.0, rUnknown, Sat Oct 11 15:15:15 PDT 2014
```

Shell Command Syntax

- Shell commands are oKen followed by parameters

```
hbase> command 'parameter1', 'parameter2'
```

```
hbase> command 'parameter1', {PARAMETER2 => 'stringvalue',
PARAMETER3 => intvalue}
```

```
hbase> create 'movie', {NAME => 'desc', VERSIONS => 5}
```

UNIVERSITAT DE BARCELONA

Shell Scripting

- The HBase Shell works in interac=ve and batch modes
  -
-

```
$ hbase shell pathtorubyscript.rb
```

-

```
hbase> require ’pathtorubyscript.rb’

hbase> myRubyFunction ’parameter1’
```

HBase Table Creation

- [illegible] and column families need to be specified at table crea=on

CreaGng Tables

```
create 'tablename', {NAME => 'colfam' [, options]} [,{...}]
```

```
create 'movie', {NAME => 'desc'}

create 'movie', {NAME => 'desc', VERSIONS => 2}
create 'movie', {NAME => 'desc'}, {NAME => 'media'}
```

```
create 'movie', 'desc', 'media'
```

Managing HBase Namespaces

- HBase provides the capability to define and manage namespaces

```
create_namespace 'namespaceName'

drop_namespace 'namespaceName'


alter_namespace 'namespaceName', {METHOD => 'set',
'PROPERTY_NAME' => 'PROPERTY_VALUE'}
```

Creating Tables in Namespaces

- General form:

```
create 'namespace:tablename', {NAME => 'colfam' [, options]}
    [,{...}]
```

-

```
create_namespace 'entertainment'

create 'entertainment:movie', {NAME => 'desc'}
```

Chapter Topics

HBase Shell

- Creating Tables with the HBase Shell
-
- Hands-On Exercise: Using the HBase Shell
- Working with Table Data
- Hands-On Exercise: Data Access in the HBase Shell
- Essential Points

Listing and Describing Tables

- List all the Tables in HBase

```
hbase> list
```

-
  -

```
hbase> describe 'movie'
```

Disabling and Enabling Tables

- Disabling a table puts it in a maintenance state
  -
  -
  -

```
hbase> disable 'movie'
```

-

```
hbase> enable 'movie'
```

Deleting

- …mmand removes the table from HBase and deletes all its files

```
hbase> disable 'movie'

hbase> drop 'movie'
```

```
hbase> truncate 'movie'
```

Altering Tables

- Tables can be changed aKer crea=on
  - The entire table can be modified
  - Column families can be added, modified, or deleted
- For some opera=ons, tables must be disabled while the changes are being applied
  - Re-enable the table aSer changes are complete
  - As of HBase 0.92, schema changes such as column family changes do not require the table to be disabled
  - By default, online schema changes are enabled
    - – The `hbase.online.schema.update.enable` property is set to `true`

Adding, Deleting, and Modifying Column Families

- Add a new column family to an existing table

```
hbase> alter 'movie', NAME => 'media'
```

```
hbase> alter 'movie', NAME => 'media', METHOD => 'delete'
```

```
hbase> alter 'movie', NAME => 'desc', VERSIONS => 5
```

Altering Column Families Asynchronously

- You can use the shell command alter_async to alter a column family
  -
  -

```
hbase> alter_async 'movie', NAME => 'desc', VERSIONS => 6
```

-