Lab Report: HBase Design and Programming

Course: ISIT312/ISIT912 Big Data Management

Semester: Spring 2023

Student: [Your Name]

Date: October 23, 2025

Executive Summary

This laboratory exercise introduced Apache HBase, a distributed, scalable NoSQL database built on top of Hadoop HDFS. The lab covered fundamental operations including starting HBase services, creating and managing tables with multiple column families, performing data manipulation operations (insert, update, delete), and understanding HBase's row-key based data model. The practical session also explored the differences between HBase's schema-less design and traditional relational databases, particularly focusing on denormalization strategies and version management.

Part 1: Starting HBase Services

1.1 Prerequisites

Before starting HBase, verified that all five Hadoop services were running as required.

1.2 Starting HBase Server

Command executed:

```bash
$HBASE_HOME/bin/start-hbase.sh
```

Terminal output:

```
localhost: starting zookeeper, logging to /usr/share/hbase/logs/hbase-bigdata-zookeeper-bigdata-VirtualBox.out
starting master, logging to /usr/share/hbase/logs/hbase-bigdata-master-bigdata-VirtualBox.out
OpenJDK 64-Bit Server VM warning: ignoring option PermSize=128m; support was removed in 8.0
OpenJDK 64-Bit Server VM warning: ignoring option MaxPermSize=128m; support was removed in 8.0
starting regionserver, logging to /usr/share/hbase/logs/hbase-bigdata-1-regionserver-bigdata-VirtualBox.out
```

Result: Three essential HBase services started successfully:

- **HMaster:** Master server managing region assignments
- **HRegionServer:** Region server handling data storage and retrieval

- **HQuorumPeer:** ZooKeeper coordination service

1.3 Launching HBase Shell

Command:

```bash
$HBASE_HOME/bin/hbase shell
```

Output:

```
HBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 1.2.6, rUnknown, Mon May 29 02:25:32 CDT 2017
```

Connection established successfully at prompt: `hbase(main):001:0>`

Part 2: Basic HBase Commands and System Verification

2.1 Checking System Status

Command:

```
status
```

Output:

```
1 active master, 0 backup masters, 1 servers, 0 dead, 3.0000 average load
```

Interpretation: System running normally with one active HMaster and one RegionServer, no dead nodes.

2.2 User Identity Verification

Command:

```
whoami
```

Output:

```
bigdata (auth:SIMPLE)
groups: bigdata, adm, cdrom, sudo, dip, plugdev, lpadmin, sambashare, vboxsf
```

2.3 Listing Existing Tables

Command:

```
list
```

Output:

```
TABLE
COURSEWORK
Company
task2
3 row(s) in 0.0390 seconds
```

Result: Three existing tables identified before creating new tables.

Part 3: HBase Table Design and Creation

3.1 Design Concept

The laboratory implemented a coursework submission tracking system with the following conceptual schema:

- **STUDENT** (snumber, first-name, last-name, degree)
- **SUBJECT** (code, title, credits)
- **SUBMISSION** (student, subject, submission-number, date, signature, files)
- **FILE** (file-number, path/filename)

Design Strategy: "Relational implementation style" using:

- Composite row keys: `entity-type:identifier`
- Four column families: STUDENT, SUBJECT, SUBMISSION, FILE
- Controlled denormalization for query optimization

3.2 Creating Base Table

Command:

```
create 'COURSEWORK', 'STUDENT'
```

Execution time: 2.3560 seconds

Verification command:

```
describe 'COURSEWORK'
```

Output:

```
Table COURSEWORK is ENABLED
COURSEWORK
COLUMN FAMILIES DESCRIPTION
{NAME => 'STUDENT', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false',
KEEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER',
COMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65536',
REPLICATION_SCOPE => '0'}
```

3.3 Adding Column Families

Commands executed:

```
alter 'COURSEWORK', {NAME=>'SUBJECT', VERSIONS=>'1'}
alter 'COURSEWORK', {NAME=>'FILE', VERSIONS=>'2'}
alter 'COURSEWORK', {NAME=>'SUBMISSION', VERSIONS=>'1'}
```

Execution times: 2.0720s, 1.9590s, 2.0130s respectively

Key observation: FILE column family configured with VERSIONS=>'2' to support file resubmissions.

Final table structure verification:

```
describe 'COURSEWORK'
```

Result: Four column families successfully created:

1. **FILE** - VERSIONS: 2
2. **STUDENT** - VERSIONS: 1
3. **SUBJECT** - VERSIONS: 1
4. **SUBMISSION** - VERSIONS: 1

Part 4: Data Insertion Operations

4.1 Inserting Student Records

First student (James Bond):

```
put 'COURSEWORK','student:007','STUDENT:snumber','007'
put 'COURSEWORK','student:007','STUDENT:first-name','James'
put 'COURSEWORK','student:007','STUDENT:last-name','Bond'
put 'COURSEWORK','student:007','STUDENT:degree','MIT'
```

Verification scan:

```
scan 'COURSEWORK'
```

Output:

```
ROW                COLUMN+CELL
student:007        column=STUDENT:degree, timestamp=1761208194145, value=MIT
student:007        column=STUDENT:first-name, timestamp=1761208192463, value=James
student:007        column=STUDENT:last-name, timestamp=1761208192534, value=Bond
student:007        column=STUDENT:snumber, timestamp=1761208192408, value=007
1 row(s) in 0.1030 seconds
```

Second student (Harry Potter):

```
put 'COURSEWORK','student:666','STUDENT:snumber','666'
put 'COURSEWORK','student:666','STUDENT:firstname','Harry'
put 'COURSEWORK','student:666','STUDENT:lastname','Potter'
put 'COURSEWORK','student:666','STUDENT:degree','BCS'
```

Note: Inconsistent column qualifier naming (firstname vs first-name) demonstrates HBase's schema flexibility.

4.2 Inserting Subject Records

Subject 312 (Big Data):

```
put 'COURSEWORK','subject:312','SUBJECT:code','312'
put 'COURSEWORK','subject:312','SUBJECT:title','Big Data'
put 'COURSEWORK','subject:312','SUBJECT:credits','6'
```

Subject 313 (Very Big Data):

```
put 'COURSEWORK','subject:313','SUBJECT:code','313'
put 'COURSEWORK','subject:313','SUBJECT:title','Very Big Data'
put 'COURSEWORK','subject:313','SUBJECT:credits','12'
```

4.3 Inserting Submission Records

First submission (student:007, subject:312, submission:1):

```
put 'COURSEWORK','submission:007|312|1','SUBMISSION:sdate','01-APR-2017'
put 'COURSEWORK','submission:007|312|1','SUBMISSION:esignature','jb'
put 'COURSEWORK','submission:007|312|1','SUBMISSION:totalfiles','2'
put 'COURSEWORK','submission:007|312|1','SUBMISSION:dayslate','0'
put 'COURSEWORK','submission:007|312|1','STUDENT:snumbner','007'
put 'COURSEWORK','submission:007|312|1','SUBJECT:code','312'
put 'COURSEWORK','submission:007|312|1','FILE:fnumber1','path/file-name1-1'
put 'COURSEWORK','submission:007|312|1','FILE:fnumber2','path/file-name1-1'
```

Key design feature: Row key format `submission:snumber|code|sub-number` enables efficient range scans.

Optional denormalization:

```
put 'COURSEWORK','submission:007|312|1','STUDENT:firstname','James'
put 'COURSEWORK','submission:007|312|1','STUDENT:lastname','Bond'
```

Trade-off: Improved query performance vs. increased storage and potential inconsistency.

4.4 Complete Data Load

Loaded five submission records total:

- submission:007|312|1 (James Bond, Big Data, assignment)
- submission:007|313|1 (James Bond, Very Big Data, project)
- submission:666|312|3 (Harry Potter, Big Data, assignment)
- submission:666|312|4 (Harry Potter, Big Data, assignment)
- submission:666|313|2 (Harry Potter, Very Big Data, project)

Part 5: Data Retrieval Operations

5.1 Full Table Scan

Command:

```
scan 'COURSEWORK'
```

Result: 9 rows retrieved showing all entities:

- 2 student rows
- 2 subject rows
- 5 submission rows

Observation: Row keys sorted lexicographically (student → subject → submission).

5.2 Retrieving Individual Rows

Command:

```
get 'COURSEWORK','student:007'
```

Output:

```
COLUMN                  CELL
STUDENT:degree          timestamp=1761208194145, value=MIT
STUDENT:first-name      timestamp=1761208192463, value=James
STUDENT:last-name       timestamp=1761208192534, value=Bond
STUDENT:snumber         timestamp=1761208281423, value=008
4 row(s) in 0.0360 seconds
```

5.3 Retrieving Individual Cells

Command:

```
get 'COURSEWORK','student:007','STUDENT:snumber'
```

Output:

```
COLUMN                  CELL
STUDENT:snumber         timestamp=1761208281423, value=008
1 row(s) in 0.0290 seconds
```

5.4 Retrieving Cell Versions

Command:

```
get 'COURSEWORK','student:007',{COLUMN=>'STUDENT:snumber',VERSIONS=>5}
```

Output:

```
COLUMN                CELL
STUDENT:snumber         timestamp=1761208281423, value=008
1 row(s) in 0.0090 seconds
```

Note: Only one version shown because STUDENT column family configured with VERSIONS=>'1'.

Part 6: Data Update Operations

6.1 Updating Cell Values

Command:

```
put 'COURSEWORK','student:007','STUDENT:snumber','008'
```

Result: Student number changed from '007' to '008'.

Verification scan showed:

```
student:007             column=STUDENT:snumber, timestamp=1761208281423, value=008
```

Key observation: Row key remains 'student:007' while cell value changed to '008'. This demonstrates:

- Row keys are immutable
- Cell values can be updated with new timestamps
- Single-version column families replace old values

6.2 Creating New Cell Versions

Context: FILE column family configured with VERSIONS=>'2' to support file resubmissions.

Command:

```
put 'COURSEWORK','submission:007|313|1','FILE:fnumber1','path/file-name3-3'
```

Verification:

```
get 'COURSEWORK','submission:007|313|1',{COLUMN=>'FILE:fnumber1',VERSIONS=>2}
```

Output:

```
COLUMN              CELL
FILE:fnumber1       timestamp=1761208501593, value=path/file-name3-3
FILE:fnumber1       timestamp=1761208258887, value=path/file-name3-1
2 row(s) in 0.0160 seconds
```

Result: Both versions preserved with different timestamps:

- **Version 1 (older):** path/file-name3-1 (timestamp: 1761208258887)
- **Version 2 (newer):** path/file-name3-3 (timestamp: 1761208501593)

Use case: Tracks file submission history for academic integrity verification.

Part 7: Projection Operations (Column Family Scans)

7.1 Scanning Specific Column Family

Command:

```
scan 'COURSEWORK',{COLUMN=>'STUDENT'}
```

Output:

```
ROW                 COLUMN+CELL
student:007         column=STUDENT:degree, timestamp=1761208194145, value=MIT
student:007         column=STUDENT:first-name, timestamp=1761208192463, value=James
student:007         column=STUDENT:last-name, timestamp=1761208192534, value=Bond
student:007         column=STUDENT:snumber, timestamp=1761208281423, value=008
student:666         column=STUDENT:degree, timestamp=1761208212271, value=BCS
student:666         column=STUDENT:firstname, timestamp=1761208211525, value=Harry
student:666         column=STUDENT:lastname, timestamp=1761208211566, value=Potter
student:666         column=STUDENT:snumber, timestamp=1761208211489, value=666
submission:007|312|1     column=STUDENT:firstname, timestamp=1761208237297, value=James
submission:007|312|1     column=STUDENT:lastname, timestamp=1761208237637, value=Bond
submission:007|312|1     column=STUDENT:snumbner, timestamp=1761208229029, value=007
submission:007|313|1     column=STUDENT:snumbner, timestamp=1761208258834, value=007
submission:666|312|3     column=STUDENT:snumbner, timestamp=1761208259094, value=666
submission:666|312|4     column=STUDENT:snumbner, timestamp=1761208259431, value=666
submission:666|313|2     column=STUDENT:snumbner, timestamp=1761208259741, value=666
7 row(s) in 0.0950 seconds
```

Analysis: Retrieved 7 rows (2 student entities + 5 submissions with student data).

7.2 Scanning Specific Column Qualifier

Command:

```
scan 'COURSEWORK',{COLUMN=>'STUDENT:first-name'}
```

Output:

```
ROW                    COLUMN+CELL
student:007            column=STUDENT:first-name, timestamp=1761208192463, value=James
1 row(s) in 0.0180 seconds
```

Note: Only retrieved exact match for 'first-name', not 'firstname' (different qualifier).

Part 8: Table Management Operations

8.1 Disabling Table

Command:

```
disable 'COURSEWORK'
```

Execution time: 2.3560 seconds

Purpose: Required before dropping or performing major alterations.

8.2 Dropping Table

Command:

```
drop 'COURSEWORK'
```

Execution time: 1.3120 seconds

Verification:

```
list
```

Result: COURSEWORK table no longer appears in table list.

8.3 Attempting Truncate on Dropped Table

Command:

```
truncate 'COURSEWORK'
```

Output:

```
ERROR: Unknown table COURSEWORK!
```

Expected behavior: Cannot truncate non-existent table.

Key Findings and Observations

1. HBase Data Model Characteristics

Row-oriented storage:

- Data organized by row keys (lexicographically sorted)
- Efficient for row-level operations and range scans
- Row key design critical for performance

Schema flexibility:

- Column qualifiers can vary between rows
- Demonstrated by inconsistent naming (firstname vs first-name)
- No schema enforcement at insertion time

Column family architecture:

- Different column families stored separately
- Version control configurable per column family
- FILE: VERSIONS=2, others: VERSIONS=1

2. Performance Characteristics

Operation timings observed:

- Table creation: ~2.4 seconds
- Column family addition: ~2 seconds each
- Single row insertion: 0.004-0.037 seconds
- Full table scan: ~0.11 seconds (9 rows)
- Row retrieval: ~0.036 seconds

- Cell retrieval: ~0.029 seconds

Comparison with Hive (from Lab 3):

- HBase PUT: milliseconds
- Hive INSERT: 12-19 seconds (MapReduce overhead)
- HBase better suited for real-time operations

3. Design Patterns and Trade-offs

Denormalization strategy:

```
Row: submission:007|312|1
 ├── SUBMISSION:* (submission details)
 ├── STUDENT:snumbner (foreign key)
 ├── STUDENT:firstname (denormalized)
 ├── STUDENT:lastname (denormalized)
 ├── SUBJECT:code (foreign key)
 └── FILE:* (file references)
```

Advantages:

- Single row retrieval for complete submission info
- Reduced join operations
- Improved read performance

Disadvantages:

- Data redundancy
- Update anomalies (student name changes require multiple updates)
- Increased storage requirements

4. Version Management

Single-version behavior (VERSIONS='1'):

- Old values replaced on PUT
- Timestamp updated
- Previous data lost

Multi-version behavior (VERSIONS='2'):

- Both versions retained

- Latest version returned by default
- Historical data preserved for audit

5. HBase vs Relational Databases

Aspect	HBase	Relational DBMS
Schema	Flexible, schema-on-read	Fixed, schema-on-write
Joins	Manual, application-level	Native SQL joins
Transactions	Row-level only	ACID across tables
Scalability	Horizontal (distributed)	Vertical (limited)
Query language	API/Shell commands	SQL
Normalization	Denormalized preferred	Normalized preferred

Challenges Encountered

1. Row Key Design Complexity

Challenge: Choosing appropriate row key format for efficient queries.

Solution: Used composite keys with prefixes:

- `student:007`
- `subject:312`
- `submission:007|312|1`

Benefit: Enables entity-type grouping and range scans.

2. Column Qualifier Inconsistency

Issue: Mixed naming conventions (first-name vs firstname, snumber vs snumbner).

Impact: Queries returned incomplete results when expecting consistent naming.

Lesson: Establish naming conventions before data loading.

3. Version Configuration Timing

Issue: FILE column family initially created with VERSIONS='2', but demonstration used later alter command.

Clarification: Both methods work:

- Set during creation: `create 'TABLE', {NAME=>'CF', VERSIONS=>'2'}`
- Set during alteration: `alter 'TABLE', {NAME=>'CF', VERSIONS=>'2'}`

4. Understanding Put Semantics

Initial confusion: Whether PUT replaces or creates new versions.

Resolution: Behavior depends on VERSIONS configuration:

- VERSIONS='1': Replacement
- VERSIONS>1: New version creation (up to limit)

Best Practices Identified

1. Row Key Design

- **Use meaningful prefixes** for entity type identification
- **Include hierarchical information** (e.g., student|subject|submission)
- **Avoid hotspotting** (sequential keys causing uneven distribution)
- **Consider query patterns** when designing keys

2. Column Family Organization

- **Group related data** in same column family
- **Minimize column families** (2-3 recommended, 10 maximum)
- **Configure appropriate versions** based on use case
- **Use consistent naming** for column qualifiers

3. Data Loading

- **Batch operations** when possible (future lab: bulk loading)
- **Verify structure** with describe before loading
- **Test with sample data** before full load
- **Monitor timestamps** for debugging

4. Version Management

- **Default to VERSIONS='1'** unless history required
- **Use VERSIONS>1** for:
  - Audit trails
  - Document versioning
  - Temporal data

- **Set retention policies** to manage storage

5. Query Optimization

- **Use get for known row keys** (faster than scan)
- **Limit scan scope** with STARTROW/STOPROW
- **Project specific columns** to reduce data transfer
- **Cache frequently accessed data** at application level

Comparison with Lab 3 (Hive)

Feature	HBase (Current Lab)	Hive (Lab 3)
Data model	Key-value (column-oriented)	Relational (row-oriented)
Schema	Flexible, dynamic	Fixed, defined at creation
Query language	Shell commands/API	HiveQL (SQL-like)
Write performance	Fast (milliseconds)	Slow (seconds, MapReduce)
Read performance	Fast single-row access	Efficient full-table scans
Use case	Random access, real-time	Analytics, batch processing
ACID support	Row-level atomicity	Table-level transactions
Data loading	put commands	LOAD DATA, INSERT
Versioning	Built-in (configurable)	Not supported

Complementary roles:

- **HBase:** Operational data store (OLTP-like)
- **Hive:** Data warehouse (OLAP)
- **Integration:** Often used together in lambda architecture

---

Conclusions

This laboratory successfully demonstrated:

1. **HBase Architecture:** Understanding of distributed NoSQL storage with ZooKeeper coordination, HMaster management, and RegionServer data handling.
2. **Table Design:** Implementation of denormalized schema using column families and composite row keys to optimize query performance.
3. **CRUD Operations:** Proficiency with put, get, scan, and delete operations through HBase shell.

4. **Version Control:** Practical experience with multi-version concurrency control for maintaining data history.
5. **Performance Characteristics:** Recognition of HBase's strength in real-time random access compared to Hive's batch-oriented approach.
6. **Design Trade-offs:** Understanding of denormalization benefits (query speed) versus costs (redundancy, update complexity).

Key takeaway: HBase excels at providing fast, scalable access to sparse data sets with flexible schemas, making it ideal for user profiles, time-series data, and real-time analytics, complementing batch-oriented tools like Hive.

Appendix A: Complete Command Reference

Table Management

```
create 'TABLE_NAME', 'COLUMN_FAMILY'
alter 'TABLE_NAME', {NAME=>'CF', VERSIONS=>'N'}
describe 'TABLE_NAME'
list
disable 'TABLE_NAME'
drop 'TABLE_NAME'
truncate 'TABLE_NAME'
```

Data Manipulation

```
put 'TABLE','ROW_KEY','CF:QUALIFIER','VALUE'
get 'TABLE','ROW_KEY'
get 'TABLE','ROW_KEY','CF:QUALIFIER'
get 'TABLE','ROW_KEY',{COLUMN=>'CF:QUALIFIER',VERSIONS=>N}
scan 'TABLE'
scan 'TABLE',{COLUMN=>'CF'}
scan 'TABLE',{COLUMN=>'CF:QUALIFIER'}
delete 'TABLE','ROW_KEY','CF:QUALIFIER'
deleteall 'TABLE','ROW_KEY'
```

System Commands

```
status
whoami
version
exit
```

Script Processing

```
source('script.hb')
$HBASE_HOME/bin/hbase shell < script.hb > report.rpt
```

Appendix B: Java API Examples (Optional)

The laboratory documentation provided Java API examples for:

- **ListTables.java** - Listing all tables
- **CreateTable.java** - Creating tables programmatically
- **PutRow.java** - Inserting data
- **GetRow.java** - Retrieving data
- **ScanTable.java** - Scanning tables
- **DelRow.java** - Deleting rows
- **DropTable.java** - Dropping tables

Compilation pattern:

bash

```bash
javac -classpath .:/usr/share/hbase/lib/* ClassName.java
java -cp .:/usr/share/hbase/lib/* ClassName
```

End of Report