HBase Cheatsheet

A comprehensive cheat sheet for HBase, covering architecture, data model, key operations, and administration.

HBase Architecture & Data Model

Core Components

HRegionServer	Hosts and manages HRegions; handles read/write requests.
HMaster	Assigns regions to RegionServers, handles schema changes, and performs administrative tasks.
ZooKeeper	Maintains configuration information, naming, and distributed synchronization.
HDFS	Hadoop Distributed File System; stores HBase data persistently.

Data Model

HBase is a distributed, scalable, big data store.

It's designed to store and retrieve data within large tables. Data is stored as key-value pairs.

- **Table:** Contains rows.
- **Row Key:** Unique identifier for a row.
- **Column Family:** Groups related columns.
- **Column Qualifier:** Identifies a column within a column family.
- **Version:** Each cell value has a version timestamp.

Key Concepts

Regions	Tables are split into regions. A region contains a subset of rows. Regions are the unit of distribution and scalability.
Store	Each region contains one or more stores. A store contains a MemStore and zero or more StoreFiles (HFiles).
MemStore	In-memory buffer that stores recent writes.
HFile	Sorted key-value pairs stored on HDFS.

Basic HBase Operations (CLI)

Table Management

- `create 'table_name', 'column_family1', 'column_family2'` - Creates a new table.
- `list` - Lists all tables.
- `describe 'table_name'` - Describes the table schema.
- `disable 'table_name'` - Disables a table.
- `enable 'table_name'` - Enables a table.
- `drop 'table_name'` - Drops a table (must be disabled first).
- `exists 'table_name'` - Checks if table exists.

Data Manipulation

- `put 'table_name', 'row_key', 'column_family:qualifier', 'value'` - Inserts or updates a cell value.
- `get 'table_name', 'row_key'` - Retrieves all columns for a given row.
- `get 'table_name', 'row_key', 'column_family:qualifier'` - Retrieves a specific cell.
- `scan 'table_name'` - Scans the entire table.
- `scan 'table_name', {STARTROW => 'row_key1', STOPROW => 'row_key2'}` - Scans a range of rows.
- `delete 'table_name', 'row_key', 'column_family:qualifier'` - Deletes a specific cell.
- `deleteall 'table_name', 'row_key'` - Deletes all cells in a row.

HBase Shell Commands

Namespace Management

- `create_namespace 'namespace_name'` - Creates a namespace.
- `list_namespace` - Lists all namespaces.
- `describe_namespace 'namespace_name'` - Describes a namespace.
- `alter_namespace 'namespace_name', {METHOD => 'set', 'PROPERTY_NAME' => 'property_value'}` - Alters a namespace property.
- `drop_namespace 'namespace_name'` - Drops a namespace.

Advanced Scan Operations

- `scan 'table_name', {COLUMNS => ['col_family1', 'col_family2:qualifier']}` - Scans specific column families or columns.
- `scan 'table_name', {LIMIT => 10}` - Limits the number of rows returned.
- `scan 'table_name', {REVERSED => true}` - Scans in reverse order.
- `scan 'table_name', {FILTER => "RowFilter(=, 'binary:row_key')"}` - Applies a filter to the scan.
- `scan 'table_name', {VERSIONS => 5}` - Retrieves the last 5 versions of each cell.
- `count 'table_name'` - Counts the number of rows in a table.

Configuration

HBase configuration is managed through `hbase-site.xml`. Key properties include:

- `hbase.rootdir`: HDFS directory for HBase data.
- `hbase.zookeeper.quorum`: List of ZooKeeper servers.
- `hbase.cluster.distributed`: Set to `true` for distributed mode.

HBase Java API

Connecting to HBase

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.HBaseConfiguration;
import org.apache.hadoop.hbase.client.Connection;
import org.apache.hadoop.hbase.client.ConnectionFactory;

Configuration config = HBaseConfiguration.create();
config.set("hbase.zookeeper.quorum", "localhost"); // Replace with your
ZooKeeper quorum

try (Connection connection = ConnectionFactory.createConnection(config)) {
    // Use the connection
} catch (Exception e) {
    e.printStackTrace();
}
```

Basic Operations (Java)

```
import org.apache.hadoop.hbase.TableName;
import org.apache.hadoop.hbase.client.*;
import org.apache.hadoop.hbase.util.Bytes;

TableName tableName = TableName.valueOf("mytable");

try (Table table = connection.getTable(tableName)) {
    // Put data
    Put put = new Put(Bytes.toBytes("row1"));
    put.addColumn(Bytes.toBytes("cf1"), Bytes.toBytes("qual1"),
Bytes.toBytes("value1"));
    table.put(put);

    // Get data
    Get get = new Get(Bytes.toBytes("row1"));
    Result result = table.get(get);
    byte[] value = result.getValue(Bytes.toBytes("cf1"),
Bytes.toBytes("qual1"));
    String valueStr = Bytes.toString(value);
    System.out.println("Value: " + valueStr);

    // Scan data
    Scan scan = new Scan();
    try (ResultScanner scanner = table.getScanner(scan)) {
        for (Result row : scanner) {
            // Process each row
        }
    }
} catch (Exception e) {
    e.printStackTrace();
}
```