Assignment 1
HBase Theory and Practical Example

Theodore Fitch
Department of Data Analytics, University of Maryland Global Campus

DATA 650: Big Data Analytics

Dr. Ozan Ozcan

May 28th, Summer 2024

Part 1:

1. Discuss the key components of HBase and the function of each component.

HBase, inspired by Google’s Bigtable, is a distributed, scalable, NoSQL database built on top of Hadoop’s HDFS (Cognitive Class, 2014). Its architecture is composed of several key components that work in unison to manage and store vast amounts of structured and semi-structured data efficiently. The HBase Master serves as the control node, managing the distribution and assignment of regions (subsections of tables) to the various Region Servers. These Region Servers are the workhorses of HBase, handling read and write requests and managing regions. Each Region Server communicates with HDFS to store data persistently (Tutorials Point, n.d.).

Regions themselves are the fundamental units of scalability and load balancing, dynamically split and reallocated as data grows. Data within these regions is stored in HFiles, immutable files that ensure durability and high throughput (Tutorials Point, n.d.). The MemStore, an in-memory write buffer, temporarily holds data before it is flushed to HFiles, optimizing write performance. To ensure data integrity, HBase employs a Write-Ahead Log (WAL), recording all changes before they are committed to the MemStore. ZooKeeper, another critical component, coordinates distributed processes and maintains the overall health of the HBase cluster, ensuring that the Master and Region Servers are always synchronized and operational (Tutorials Point, n.d.).

2. Discuss the differences between row-oriented and column-oriented databases. Provide examples of each.

Row-oriented and column-oriented databases differ fundamentally in their data storage methodologies, each optimized for distinct types of workloads. Row-oriented databases, such as MySQL, PostgreSQL, and Oracle, store data in rows. This approach is ideal for transactional systems (Online Transaction Process, or OLTP) where complete records are frequently accessed, inserted, or updated (Tutorials Point, n.d.). For example, in an e-commerce application, retrieving or updating a customer's order requires accessing multiple columns within the same row, making row-oriented databases highly efficient for these operations.

Conversely, column-oriented databases like HBase, Apache Cassandra, and Apache Kudu store data by columns. This architecture excels in analytical processing (Online Analytical Processing, or OLAP), where queries often involve aggregates or summaries of a few columns over many rows (Tutorials Point, n.d.). For instance, in a data warehousing scenario, an analyst might query total sales per region, necessitating the reading of only the sales and region columns across millions of rows. This approach minimizes the amount of data read from disk, significantly boosting performance for such queries. While column-oriented databases provide excellent read performance for analytical queries, they can be less efficient for transactional operations involving entire rows.

3. When would you use Hive instead of HBase?

Hive and HBase cater to different needs within the Hadoop ecosystem, each optimized for distinct types of data processing tasks. Hive is designed for batch processing and querying large datasets using a SQL-like language called HiveQL (Simplilearn, 2021). It translates these queries into MapReduce jobs, making it highly suitable for data warehousing tasks that involve complex queries and data analysis over extensive datasets (MindMajix, n.d.). For example, generating monthly sales reports or performing trend analysis over several years of data are scenarios where Hive excels. Its ability to handle large-scale ETL (Extract, Transform, Load) operations and provide a familiar SQL interface makes it a preferred choice for data analysts and business intelligence applications.

On the other hand, HBase is a NoSQL database optimized for real-time read/write access to large volumes of sparse data. It is designed to handle workloads that require low-latency access and high throughput for operations involving individual records (MindMajix, n.d.). For instance, applications such as online gaming, social media platforms, and real-time analytics dashboards benefit from HBase's ability to quickly retrieve and update specific rows of data. Therefore, while Hive is ideal for batch processing and complex queries, HBase is the better choice for applications demanding real-time data access and updates.

4. Discuss 7 HBase shell data manipulation commands and what they do

In HBase, data manipulation is primarily performed through the HBase shell, a command-line interface that allows users to interact with the database. Seven essential HBase shell commands are integral for managing and manipulating data (Taylor, 2024):

1. *create*: This command is used to create a new table in HBase. For instance, *create* 'students', 'EmplID' creates a table named 'students' with a column family 'EmplID'.

2. *put*: The put command inserts or updates a cell value in a table. An example is *put* 'students', 'row1', 'info:name', 'John Wayne', which inserts the value 'John Wane' into the 'name' column of the 'info' column family for the row 'row1'.

3. *get*: To retrieve data from a specific cell, the *get* command is employed. For example, *get* ‘students’, ‘row1’, ‘info:name’ fetches the value of the ‘name’ column in the ‘info’ column family for ‘row1’ which from the previous *put* command would be ‘John Wayne’

4. *scan*: The *scan* command scans and retrieves data from a range of rows. Simply executing *scan* ‘students’ retrieves all rows and columns from the ‘students’ table.

5. *delete*: This command removes a specific cell value. For instance, *delete* ‘students’, ‘row1’, ‘info:name’ deletes the value in the ‘name’ column of the ‘info’ column family for ‘row1’.

6. *deleteall*: To delete all cells in a specific row, the *deleteall* command is used. Accordingly, *deleteall* ‘students’, ‘row1’ removes all column data for ‘row1’.

7. *truncate*: The *truncate* command effectively deletes all data in a table by disabling, dropping, and recreating it. For example, *truncate* ‘students’ removes all data but keeps the table schema intact.

These commands are only 7 examples of the many HBase commands which comprise a comprehensive toolkit. This enables data scientists to manage and manipulate data within HBase, facilitating both simple and complex operations.

5. How would you implement a query that joins multiple tables in HBase?

HBase, unlike traditional relational databases, does not support direct SQL-like join operations due to its NoSQL architecture. However, joining multiple tables in HBase can be achieved through several alternative methods. Some examples include MapReduce jobs or utilizing Apache Phoenix for SQL capabilities.

Using MapReduce (Barrera, 2016): A MapReduce job can be designed to perform the join operation by reading data from multiple HBase tables and combining them based on a common key. In the map phase, the job emits key-value pairs where the key is the join attribute, and the value is the associated data. During the reduce phase, records with the same key are combined, effectively performing the join.

1. Map Phase: The mapper reads data from the relevant tables and emits (join_key, (source, data)) pairs.
2. Reduce Phase: The reducer groups these pairs by the join key and merges records from the different sources.

Example Implementation:

- Mapper: Emits pairs like (user_id, (‘table1’, user_data)) and (user_id, (‘table2’, order_data)).
- Reducer: Combines records from both tables where the user_id matches, effectively joining the data.

Using Apache Phoenix (Apache, n.d.): Apache Phoenix provides an SQL-like interface over HBase, enabling complex queries, including joins. After installing Phoenix, tables can be defined using Phoenix DDL, and SQL queries can be executed to join tables.

Example:

sql

Copy code

```
SELECT a.*, b.* FROM table1 a JOIN table2 b ON a.join_key = b.join_key;
```

This SQL query, executed through Phoenix, performs the join operation efficiently, leveraging HBase’s underlying storage.

6. What is a namespace in HBase?

A namespace in HBase functions as a logical grouping of tables, akin to schemas in traditional relational databases (HDFS Tutorial Team, n.d.). It provides an organizational structure that aids in managing tables more effectively, especially in environments where multiple teams or applications share the same HBase cluster. By default, HBase includes a default namespace where all tables are created unless specified otherwise. Users can create custom namespaces to segregate tables, enhancing manageability and isolation. Namespaces are particularly useful in multi-tenant environments where different projects or departments need isolated data spaces. For example, an organization might create namespaces like sales, engineering, and marketing to group related tables under these logical categories. Creating a namespace is straightforward with the create_namespace command, and tables within a namespace can be referenced using the namespace:table syntax.

7. What happens when you delete table cell(s) in HBase?

Deleting table cell(s) in HBase involves adding a tombstone marker to the specified cells, rather than immediately removing the data. This tombstone marker signifies that the cell is deleted, and subsequent read operations will treat the cell as if it is no longer there (Oracle, 2022). The actual data remains in the HFile until a major compaction occurs. During major compaction, HBase scans the data and permanently removes cells marked with tombstones, freeing up space and optimizing storage. For instance, when executing the command delete ‘table_name’, ‘row1’, ‘cf:column1’, a tombstone marker is added to the ‘column1’ cell in the ‘cf’ column family of ‘row1’. The data will appear deleted to any read operation, but the physical removal will only occur during the next major compaction. This approach ensures data integrity and consistency, allowing for the recovery of recently deleted data if necessary.

8. Discuss the approaches for storing multimedia data, including videos and images, in HBase.

Storing multimedia data, such as videos, audio, and images, in HBase involves addressing the challenges associated with large binary objects. One common approach is to encode the binary data using Base64 encoding, which converts the binary data into a text format suitable for storage in HBase cells (Aggarwal, 2015). This method ensures data integrity but can increase the storage size due to the encoding overhead. An alternative approach is chunking, where large multimedia files are split into smaller, manageable chunks. Each chunk is stored in a separate cell, and metadata is maintained to keep track of the sequence of these chunks. This method facilitates efficient storage and retrieval, as the entire file can be reconstructed by aggregating the chunks based on their metadata. For example, a large video file can be divided into several smaller chunks, with each chunk stored in a row identified by a unique key. The metadata might include the file name, chunk number, and total number of chunks, ensuring that the file can be accurately reassembled when needed.

Another efficient strategy involves leveraging HDFS for storing the actual multimedia files and using HBase to store the metadata and file references (Apache, 2024). This hybrid approach takes advantage of HDFS's capability to handle large files while using HBase for quick metadata access. By storing file paths or URLs in HBase, applications can efficiently retrieve and stream multimedia content from HDFS. These approaches ensure that HBase can effectively manage large multimedia datasets, combining its scalability and real-time access capabilities with the robust storage mechanisms of HDFS.

9. The short pseudocode example concludes with discussion of conceptual view after parsing 2 sample records. Show the conceptual view after parsing the following third sample record.

The following is the record given by the homework assignment:

Member ID: 2011078; Member Name: Ozcan Ozan; Member since: 2011; Exercise: Exercise: Bicep Curl, Weight: 25; Exercise: Triceps Deeps, Weight: Body Weight; Exercise: Plank, Weight: Body Weight; Trainer 1: Heather Smith; Trainer 2: Barbara Eden; Trainer 3: James Martin

The pseudocode for this record is given in the accompanying file "Pseudocode_Fitch" and found in *Figure 1*. The following is the example conceptual view with this third record parsed into the database. Most data from the record were added to previously established columns; but, 'plank' and 'trainer 3' were new columns created to account for the data parsed in record 3. Thus, this is how the database would be designed and would continue to grow in this fashion where new records are appended line by line, with new columns created in the column families as needed.

	Member			Weight									Trainer		
Key	Member Id	name	Member since	squat	Bicep curl	pushup	deadlift	Triceps extension	Triceps deeps	Dead row	lunges	Plank	Trainer 1	Trainer 2	Trainer 3
1	2000055	Yelena Bytenskaya	2000	40	15	Body weight	30	20							
2	2010022	Dave Linesh	2010	70	25				Body weight	45	35		Jim Smith	Barbara Eden	
3	2011078	Ozcan Ozan	2011		25				Body Weight			Body Weight	Heather Smith	Barbara Eden	James Martin

Part 2:

Introduction:

In this modern age when every company is generating exponentially more and more data, companies must find savvy ways to store their data. If data is inaccessible or stored poorly, then it cannot be used for predictive analysis. Companies that cannot leverage AI, predictive analysis, and their own data will quickly fall behind the companies that are using these necessary tools to increase profits and minimize losses. Some companies like Walmart have chosen to fully lean into the future creating a 40+ petabyte cloud (Marr, 2017). This is the world's largest cloud which will process 2.5 petabytes of data every hour. They have generated a specialized center of excellence just for their data so that they can analyze sales and generate more revenue. While Walmart has many other competitors and brand issues, they continue beating their competition because they have leaning into predictive analytics and leveraging e-commerce sales. Other companies must follow in these footsteps if they're going to survive in the future.

Farmer (2022) gives a non-comprehensive, high-level view of what are the top 8 benefits of harnessing big data. They will have better customer insights and increased market intelligence. They will have more agile supply chain management. Companies will gain smarter recommendations and audience targeting. Businesses will gain better data-driven innovation. They will have diverse use cases for datasets, and future-proofed data analytics platforms. Lastly, they will have improved business operations in nearly every area whether HVAC systems, production schedules, customer behavior, or network security. Big datasets are what allow machine learning, deep learning, and powerful predictive analytics to be so accurate to improve performance. Thus, in order to use this data is must be stored properly in an accessible and flexible way. It becomes obvious that the upsides to leveraging big data are endless. So the question becomes how institutions can store their data in an efficient way such that the data is usable, accessible, and consistent among other properties. This is the biggest opportunity for current data engineers.

Problem Statement:

The challenge presented for the purpose of this study is designing a way to store the UMGC records system. The current system is 50 years old and is being stored in Ascii-based text files. We are told some metadata regarding the data. For example, the students are only enrolled in one program at a time but may have as many specializations as they desire. New specializations can also be chosen as they are created. Students take multiple classes and subsequently receive a grade for each class. Each class has at least one professor and could have more than one. Students who graduate can become professors, and thus could have both a student ID and an employee ID. The text files are being stored on high volume hard disks, optical discs, CDs, and DVDs. The issues with storing data this way are endless; it’s extremely difficult to access data quickly; it’s painful to manage; it’s difficult to back up. There are also a wide variety of media on which the data is being stored. At the same time, there is a high volume of data.

This archaic system should be replaced with a new big data system which allows for some critical features. Of highest importance, this system needs to be scalable and accessible. It needs to track all courses and course attempts with varying grades. It needs to handle multiple identification keys for students who might become instructors. It should be flexible enough to incorporate new specialization from programs. It should be able to store and easily handle several hundred million records. It must also support dynamic academic programs which were different in the past, are different now, and will continue to evolve in the future. It clearly becomes evident that this challenge can be easily addressed by a modern big data system which fulfills the major requirements like being flexible, accessible, and scalable.

Design:

To address the UMGC academic records system, the best approach would leverage the big data systems of HBase and Hadoop. There are many tools within the big data ecosystem each with their own purpose. However, these two tools together are the best solution, which will easily meet all of the previously mentioned requirements. HBase is a distributed, NoSQL database modeled after Google's Bigtable which offers scalability and flexibility making it suitable for storing large volumes of structured data (Cognitive Class, 2014a). It uses a column family approach when a new row is appended to a table. New variables are assigned to particular column family and a new column is written for the data to be stored in. It offers read/write access in real-time to large datasets which is needed for the vast variety of end users of this system (including students, professors, IT personnel, advisors, and HR accessing grades and student records). HBase operates on top of the HDFS (Hadoop Distributed File System), which ensures reliable storage of large datasets by distributing data across multiple nodes, thus enhancing fault tolerance and scalability. No one ever wants to lose their data; thus this allows in-built systems to prevent massive failures in data loss. It also scales horizontally allowing more data to be gathered easily by simply adding more servers. Thus, an initial setup would allow for a couple hundred million entries with the ability to easily add more hardware to expand the maximum storage capabilities of the system.

On the other hand, Hadoop is an open-source framework designed for the distributed storage and framework of big data across clusters of commodity hardware (Simplilearn, 2021). It has two core components: 1. HFDS (a scalable, fault-tolerant file system which breaks files into blocks and distributes them in multiple nodes across a cluster) 2. MapReduce (a programming model for processing large datasets in parallel across a Hadoop cluster which divides tasks into small subtasks, processes them at the same time, and aggregates the results). Thus, Hadoop would handle the data storage (via the HDFS) and processing (via MapReduce) while HBase is used for access to real-time data using its column family model approach. As previously mentioned, this setup would allow for high scalability because it uses commodity servers as the hardware to make up the Hadoop cluster. More servers can easily be added as they are needed as long as they meet the needed requirements for CPU, memory, and disk storage. In this manner, the servers make up the needed hardware, Hadoop makes up the approach towards file storage/processing, and HBase makes up the database system using Hadoop. In summary, from a high-level view, that describes the needed hardware and software for this project.

The basic HBase table model design is observed in Table *1* and has 10 column families: Student Info, Program Info, Grades (for the 4 mentioned class subtypes in the records of DATA, DBST, ITEC, AND PMAN), and Instructors (for the same 4 class types). Since there were only 4 mentioned classes in the 3 sampled records, only 4 class types were used in this example. However, this table would need to have a column family for each class type at UMGC (and one for grades and one for instructors). These would easily be found in the UMGC catalog and cross-checked to ensure all class types are accounted for. It needs to be noted that this table design is limited by the 3 provided records and the metadata known. More records should be read to understand the variety of types of records (like how they changed over time). UMGC would need to ensure that the metadata and user requirements were also 100% complete before a table design is fully decided upon. A working group with representatives of each end user and the data engineering team should meet to ensure best practices are also agreed upon for future data collection and storage.

Implementation Methods:

Thus, in practice hardware would need to be set up for the current and anticipated big data requirements (including total disk storage needed, CPU, and memory). The software, HBase and Hadoop, would need to be installed. Then, code would need to be written for processing the records. Pseudocode is written to demonstrate how this would be done to read, parse, and transform each record (this code is attached as a separate document and found in *Figure 2*). First, a command is given to generate the table in HBase with the 10 aforementioned column families. The resultant table would appear similar to Table *1*. Then, the command would use the *put* function to place the record into the first row of the table. It would first find the column family, then look for the column. If the column doesn't yet exist, it would create it and then place the record into the row. Table *2* shows how the HBase table would appear with the first record read, parsed, and transformed into the table. 14 new columns would be generated within the 10 column families. The pseudocode similarly shows how the *put* command would be used to take the record and write the first record into this table. One unique feature this first record shows is that a single course could have multiple professors (for DBST651). This could become confusing when parsing records for a student where a student took a course more than once such that there are multiple grades for the same class, and multiple professors for each time that student took that class. In this case, code should be written to give a unique identifier to each class in the variable name. For example, the first instance column header should then read DBST-1 and the second DBST-2 under the Grades column family and Instructors column family. This would prevent confusion for past records and should definitely be used for future records. In addition, unique identifiers for each future class should be assigned such that there is no ambiguity which class grade is associated with which professor. This would require a slightly different table configuration but would be best practice for future data engineering efforts. Record 1 also demonstrates the first specialization being added where it is appended with "-1". Further specializations would appear with "-2" and so on.

Lastly, Record 1 shows how the student has both a student ID and an instructor ID both contained within their prospective columns. Likewise, Record 2 also shows both student and instructors IDs. It likewise shows a unique feature of a class being taken twice (DATA630) due to a failing grade. This class must have used the same professor for each class since there is only 1 DATA630 professor in the record. This further shows the need for unique identifiers for classes. With this failing grade in mind, whenever a query is written to pull data in would need to include nuance to ensure that only the most recent grade was used. Record 3 shows multiple unique features. First, this shows a student which has not yet graduated and is not an instructor. It shows this student has chosen 3 specializations causing 2 new columns for specialization to be generated by the code. There is a failing grade for DBST651 with a followed attempt and this record appears after DBST667 since that column was already generated for Record 1. Again, it would be ideal to either have a unique class code or a modifier on this column like "DBST651-2" to indicate a second attempt. It also shows a class with a TA (Yelena Bytenskaya) in which case a new column was written for the class with the modifier "_TA". Lastly, it also shows an incomplete record where a grade is given for PMAN but no instructor is given. This again shows the importance of unique identifiers since an incomplete record like this could be cross referenced to find the missing data.

There are countless types of queries that might be made of a dataset like this. The most obvious one being a transcript request. Code would need to be written to gather the relevant info and stylize it correctly into a report (whether for an official or unofficial transcript). The main concern would need to be using the letter grade from the final time a course is taken (which is why a “-2” modifier should be used in the variable name or a unique class code should be used). The code would read, record, and replace the original letter grade if it finds a retaken class. Another query could be looking at Grade Point Averages (GPAs) for a specific class per professor. This could help to find if a professor may be giving too high or low grades. Similarly, HR may be interested in how many students have been enrolled with each professor to find historical enrollment rates for each class. Another example would be seeing all specializations for each student and program. This would be helpful information to determine how common each specialty is and how the school is growing or shrinking. Yet another example would be determining GPAs for each major, and specialization. This would be helpful in seeing how students are performing in each program/specialization. There are countless other types of queries that may be run for a database like this, but these are simply a couple of examples.

There are definitely limitations to this HBase table example. First, as mentioned the class records are not specific enough and future records should have unique identifiers for every class and every section. This would require a slightly different table design but would be much clearer. This current table design does not accommodate new courses easily. A new column family would be needed for each new course type (like BUS for Business classes). This current table design does not show which classes associate with which program specializations. If this information is not in the historical records, it should be added to the future ones. Furthermore, it would be a more accurate table design to make each class have its own column family, and add all relevant data (like grade, instructor(s), retake status, date, etc.) to the columns. This way there is less confusion about having multiple professors on one course and it keeps the more relevant data closer together.

Conclusion:

Choosing between ACID principles (Atomicity, Consistency, Isolation, Durability) and CAP (Consistency, Availability, Partition Tolerance) principles is crucial for designing a robust database system (Cognitive Class, 2014a). Traditional relational databases adhere to ACID properties, ensuring strong consistency and data integrity. This model is ideal for applications requiring strict transaction guarantees, such as banking systems or Quality Control monitored systems in pharmaceuticals, food, or manufacturing. Likewise, it is more difficult to scale horizontally and doesn’t perform as well for larger datasets. For big data applications like the UMGC records, CAP properties are more appropriate. HBase, follows the CAP theorem, which states that a distributed system can achieve only two of the three (Consistency, Availability, Partition Tolerance) at any given time (Cognitive Class, 2014a). HBase prioritizes availability and partition tolerance, which are critical for handling large volumes of data across distributed nodes. Thus, the obvious con of using the CAP principles in a database is that it may be less consistent; however, that price is worth paying so that the data is always readily available and the system may continue operating even if one part has a failure.

As previously discussed, transitioning to a big data system using HBase and Hadoop offers a scalable and flexible solution for managing UMGC’s extensive academic records. This approach ensures more efficient data storage, retrieval, and processing while accommodating the dynamic nature of academic programs. There are several ways that the speed of this massive system could be improved. Secondary indexes could be generated for frequently queried columns. MapReduce would be used for parallel processing to increase speed when querying the large dataset. Lastly, further table optimization should be considered for minimize read/write latency. Despite the trade-offs between consistency and availability, the aforementioned design provides a robust framework for handling several hundred million student records and supporting future growth. Once the hardware and software are installed, the records can easily be parsed and stored using the discussed code. Then, reporting models can be generated for the proposed queries as well. By leveraging advanced indexing, caching, and parallel processing techniques, the system can achieve high performance and responsiveness for end-users. UMGC's future record system will allow records to be highly accessible and have high partition tolerance making each potential end user happy. Whether students, professors, HR, IT, or administrators, each user of this system will be able to easily access the data of historical and future records.

References:

Aggarwal, C. (2015). *Data Classification*. IBM T.J. Watson Research Center. https://www.charuaggarwal.net/classbook.pdf

Apache. (n.d.). *Joins*. Apache Phoenix. https://phoenix.apache.org/joins.html

Apache. (2024). *Apache HBase ™ Reference Guide*. Apache. https://hbase.apache.org/book.html

Barrera, E. (2016). *Joining Big Data Tables Using MapReduce*. LinkedIn. https://www.linkedin.com/pulse/joining-big-data-tables-using-mapreduce-emmanuel-jan-barrera/

Cognitive Class. (2014a). *Introduction to hbase - part 1*. YouTube. https://www.youtube.com/watch?v=-ICE7U59x8E&ab

Farmer, D. (2022). *8 benefits of using big data for businesses*. TechTarget. https://www.techtarget.com/searchbusinessanalytics/feature/6-big-data-benefits-for-businesses

HDFS Tutorial Team. (n.d.). *HBase Namespace Commands and Examples*. HdfsTutorial. https://hdfstutorial.com/blog/hbase-namespace-commands-examples/

Marr, B. (2017). Really big data at Walmart: Real-time insights from their 40+ petabyte data cloud. *Forbes. com*, *23*.

MindMajix. (n.d.). *Hive vs HBase*. MindMajix. https://mindmajix.com/apache-hive-vs-apache-hbase

Oracle. (2022). *Using Oracle GoldenGate for Big Data*. Oracle. https://docs.oracle.com/en/middleware/goldengate/big-data/19.1/gadbd/using-hbase-handler.html#GUID-1A9BA580-628B-48BD-9DC0-C3DF9722E0FB

Simplilearn. (2021). *Hadoop Ecosystem Explained | Hadoop Ecosystem Architecture And Components | Hadoop | Simplilearn*. YouTube. https://www.youtube.com/watch?v=p0TdBqIt3fg&ab_channel=Simplilearn

Taylor, D. (2024). *HBase Shell Commands With Examples*. Guru99. https://www.guru99.com/hbase-shell-general-commands.html

Tutorials Point. (n.d.). *HBase Tutorial*. Tutorials Point. https://www.tutorialspoint.com/hbase/hbase_tutorial.pdf

Appendix:

	Student Info	**Program Info**	**Grades DATA**	**Grades DBST**	**Grades ITEC**	**Grades PMAN**	**Instructors DATA**	**Instructors DBST**	**Instructors ITEC**	**Instructors PMAN**
Key										
1										
2										
3										

Table 1. The baseline HBase table setup with 10 column families and 3 prepared empty rows for entries.

	Student Info					**Program Info**		**Grades DATA**	**Grades DBST**		**Grades DATA**	**Grades PMAN**	**Instructors ITEC**	**Instructors DBST**			**Grades ITEC**	**Grades PMAN**
Key	Student Name	EmplID	Username	Instructor_ID	Graduation_Status	Program	Specialization-1		DBST651	DBST667			ITEC630	DBST651	DBST651	DBST667	ITEC630	
1	Yelena Bytenskaya	123456	ybytensk	234567	Yes	Information Technology	Database Systems		A	A			Jennifer Lopez	James Smith	Jennifer Lopez	Catharine Murphy	B	
2																		
3																		

Table 2. The HBase table with the first record read, parsed, and transformed shows how 14 new columns were generated within the preassigned 10 column families.

	Student Info					**Program Info**		**Grades DATA**							**Grades DBST**		**Grades ITEC**	**Grades PMAN**
Key	Student Name	EmplID	Username	Instructor_ID	Graduation_Status	Program	Specialization-1	DATA610	DATA620	DATA630	DATA630	DATA640	DATA650	DATA670	DBST651	DBST667	ITEC630	
1	Yelena Bytenskaya	123456	ybytensk	234567	Yes	Information ology	Database ns								A	A	B	
2	Linesh Dave	567890	ldave	567907	Yes	Data Analytics		B	A	C	A	B	A	B				
3																		

Instructors DATA									**Instructors DBST**			**Instructors**	**Instructors PMAN**
DATA610	DATA620	DATA630	DATA640	DATA650	DATA650	DATA650_TA	DATA670	DATA670	DBST651	DBST651	DBST667	ITEC630	
									James Smith	Jennifer Lopez	Catharine Murphy	Jennifer Lopez	
Steve Knode	Caroline Beam	Bati Firdu	Steve Knode	Elena Gortcheva	Ozan Ozcan	Yelena Bytenskaya	Jon McKeeby	Steve Knode					

Table 3. The HBase table with the first and second records read, parsed, and transformed shows the additional new columns being generated within the column families. The table extended horizontally it had to be shown here in two pieces.

	Student Info					**Program Info**				**Grades DATA**						
Key	Student Name	EmplID	Username	Instructor_ID	Graduation Status	Program	Specialization-1	Specialization-2	Specialization-3	DATA610	DATA620	DATA630	DATA630	DATA640	DATA650	DATA670
1	Yelena Bytenskaya	123456	ybytensk	234567	Yes	Information Technology	Database Systems									
2	Linesh Dave	567890	ldave	567907	Yes	Data Analytics				B	A	C	A	B	A	B
3	Jeff Martin	987654	jmartin		No	Information Technology	Database Systems	Project Management	Software Engineering							

Grades DBST			**Grades ITEC**			**Grades PMAN**	**Instructors DATA**								
DBST651	DBST667	DBST651	ITEC630	ITEC610	ITEC620	PMAN634	DATA610	DATA620	DATA630	DATA640	DATA650	DATA650	DATA650_TA	DATA670	DATA670
A	A		B												
							Steve Knode	Caroline Beam	Bati Firdu	Steve Knode	Elena Gortcheva	Ozan Ozcan	Yelena Bytenskaya	Jon McKeeby	Steve Knode
F		B		B	A	C									

Instructors DBST				**Instructors ITEC**			**Instructors PMAN**
DBST651	DBST651	DBST667	DBST651_TA	ITEC630	ITEC610	ITEC620	
James Smith	Jennifer Lopez	Catharine Murphy		Jennifer Lopez			
James Green			Yelena Bytenskaya		Brandon Morris	Elena Gortcheva	

Table 4. The HBase table with the all 3 records read, parsed, and transformed shows more additional columns being generated within the column families. There are now a total of 40 columns. The table extended horizontally so far it had to be shown here in 3 pieces.

Figure 1. Part 1 Pseudocode for Question 9

```
//
put workout row2 {
        row key <-2
        Member:Member_id<-2011078
        Member:Name<-Ozcan Ozan
        Member:Member_since<-2011
        Weight:Bicep_Curl<-25
        Weight:Triceps_Deeps<-Body_Weight
        Weight:Plank<-Body_Weight
        Trainer:Trainer1<-Heather Smith
        Trainer:Trainer2<-Barbara Eden
        Trainer:Trainer3<-James Martin
}

//
```

Figure 2. Pseudocode to parse the 50 years old UMGC academic records system in the part 2 of Assignment 1

```
//
Create table UMGC_Academic_Record with column family Student_Info, Program_Info,
Grades_DATA, Grades_DBST, Grades_ITEC, Instructors_DATA, Instructors_DBST, Instructors_ITEC
put UMGC_Academic_Record row1 {
        row key <-1
        Student_Info:Student_Name<-Yelena Bytenskaya
        Student_Info:EmplID<-123456
        Student_Info:Username<-ybytensk
        Student_Info:Instructor_ID<-234567
        Student_Info:Graduation_Status<-Yes
        Program_Info:Program<-Information Technology
        Program_Info:Specialization1<-Database Systems
        Grades_DBST:DBST651<-A
        Grades_DBST:DBST667<-A
        Grades_ITEC:ITEC630<-B
        Instructors_DBST:DBST651<-James Smith
        Instructors_DBST:DBST651<-Jennifer Lopez
        Instructors_DBST:DBST667<-Catharine Murphy
        Instructors_ITEC:ITEC630<-Jennifer Lopez
}
put UMGC_Academic_Record row2 {
        row key <-2
        Student_Info:Student_Name<-Linesh Dave
        Student_Info:EmplID<-567890
        Student_Info:Username<-ldave
        Student_Info:Instructor_ID<-567907
        Student_Info:Graduation_Status<-Yes
        Program_Info:Program<-Data Analytics
        Grades_DATA:DATA610<-B
        Grades_DATA:DATA620<-A
        Grades_DATA:DATA630<-C
        Grades_DATA:DATA630<-A
        Grades_DATA:DATA640<-B
        Grades_DATA:DATA650<-A
        Grades_DATA:DATA670<-B
        Instructors_DATA:DATA610<-Steve Knode
        Instructors_DATA:DATA620<-Caroline Beam
        Instructors_DATA:DATA630<-Bati Firdu
        Instructors_DATA:DATA640<-Steve Knode
        Instructors_DATA:DATA650<-Elena Gortcheva
        Instructors_DATA:DATA650<-Ozan Ozcan
        Instructors_DATA:DATA650_TA<-Yelena Bytenskaya
        Instructors_DATA:DATA670<-Jon McKeeby
        Instructors_DATA:DATA670<-Steve Knode
}
put UMGC_Academic_Record row3 {
        row key <-3
        Student_Info:Student_Name<-Jeff Martin
        Student_Info:EmplID<-987654
        Student_Info:Username<-jmartin
        Student_Info:Graduation_Status<-No
        Program_Info:Program<-Information Technology
        Program_Info:Specialization1<-Database Systems
        Program_Info:Specialization2<-Project Management
        Program_Info:Specialization3<-Software Engineering
        Grades_DBST:DBST651<-F
        Grades_DBST:DBST651<-B
        Grades_ITEC:ITEC610<-B
        Grades_ITEC:ITEC620<-A
        Instructors_DBST:DBST651<-James Green
        Instructors_DBST:DBST651_TA<-Yelena Bytenskaya
        Instructors_ITEC:ITEC610<-Brandon Morris
        Instructors_ITEC:ITEC620<-Elena Gortcheva
}
```