Cognitive performance application.

Shade EL-Hadik
University of Louisville

Follow this and additional works at: https://ir.library.louisville.edu/etd
Part of the Architectural Technology Commons

Recommended Citation
https://doi.org/10.18297/etd/2894

This Doctoral Dissertation is brought to you for free and open access by ThinkIR: The University of Louisville's Institutional Repository. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of ThinkIR: The University of Louisville's Institutional Repository. This title appears here courtesy of the author, who has retained all other copyrights. For more information, please contact thinkir@louisville.edu.
COGNITIVE PERFORMANCE APPLICATION

By
Shade El-Hadik
B.S., American University in Cairo, 2004
M.S., University of Louisville, 2005

A Dissertation
Submitted to the Faculty of the
J.B. Speed School of Engineering of the University of Louisville
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

in Computer Science and Engineering

Department of Computer Engineering and Computer Science
University of Louisville
Louisville, Kentucky

May 2018
COGNITIVE PERFORMANCE APPLICATION

By
Shade El-Hadik
B.S., American University in Cairo, 2004
M.S., University of Louisville, 2005

A Dissertation

Approved on February 23, 2018
By the Following Dissertation Committee:

________________________________________
Dissertation Director
Dr. Ahmed Desoky CECS

________________________________________
Dr. Adel Elmaghraby CECS

________________________________________
Dr. Dar-Jen Chang CECS

________________________________________
Dr. Juw Park CECS

________________________________________
Dr. Michael Losavio
ABSTRACT

COGNITIVE PERFORMANCE APPLICATION

Shade El-Hadik
January 12, 2018

This work shows that combining the techniques of neural networking and predictive analytics with the fundamental concepts of computing performance optimization is genuine in many ways. It has the potentials to: (1) reduce infrastructure upgrade costs (2) reduce human interactions, by enabling the system to learn, analyze, and make decisions on its own, and (3) generalize the solutions to other performance problems. This paper attempts to tackle a JVM performance optimization from a different dimension and in a way that can be scaled to other common utilized resources, such as file systems, static contents, search engines, web services...etc. It shows how to build a framework that monitors the performance metrics to determine patterns leading to bottleneck incidents and then benchmark these performance metrics. The framework uses artificial neural network in its core to accomplish this first steps with immediate benefit of eliminating the need to a domain expert analyzing which of these metrics is more important or has more weight on constituting the bottleneck condition, and hence enable the system to deal with more ambiguous situations. The framework uses an analytics engine, to establish predictive patterns between the system bottleneck and library of factors to establish an early alert system and thus enhancing the weight of the bottleneck signal. Finally, the framework acts in defense when the deadlock signal is triggered from the learning and/or the analytics engine through streaming down concurrent transactions into a temporarily queuing data structure. We put our model into a test and built a simulation to quantify the added benefit of each component of our framework. The results are proven to demonstrate the immediate benefit of our framework and open doors for other future work.
# TABLE OF CONTENTS

Chapter 1. INTRODUCTION .................................................................................................................. 1

Chapter 2. CONSTANT LOAD SIMULATION ...................................................................................... 7

  2.1. Simulation .................................................................................................................................. 7
  2.2. Populating the database ........................................................................................................... 9
  2.3. System Specifications and Simulation ....................................................................................... 11
  2.4. Demand Simulation Snapshot and CPU Utilization ................................................................. 12
  2.4.1. Demand Simulation Snapshot Run Summary ........................................................................ 13
  2.5. Demand Simulation Snapshot and Memory Utilization ............................................................. 15
  2.5.1. Demand Simulation Snapshot Run Summary ........................................................................ 16
  2.6. Simulation of real ecommerce user sessions ........................................................................... 18
  2.6.1. Demand Simulation Snapshot Run Summary ........................................................................ 21
  2.7. Summary .................................................................................................................................. 23

Chapter 3. LEARNING ENGINE .......................................................................................................... 24

  3.1. Neural Network Type .............................................................................................................. 25
  3.2. Neural Network Input ............................................................................................................. 26
  3.3. Learning procedure .................................................................................................................. 27
  3.4. Sigmoid Activation Function .................................................................................................. 28
  3.5. Neural Network Classification Method ..................................................................................... 29
  3.6. Neural Network Kernel ........................................................................................................... 30
  3.7. Mean Squared error .................................................................................................................. 31
  3.8. Learning Engine Class Diagram ................................................................................................ 32

Chapter 4. ANALYTICS ENGINE ....................................................................................................... 33

  4.1. Factory Design Pattern ............................................................................................................ 35
  4.2. Best Fit Model Criteria ............................................................................................................ 35
  4.3. Predictive Analytics Models ..................................................................................................... 37
  4.3.1. Linear Regression Model .................................................................................................. 37
  4.3.2. Multiple Linear Regression Model ..................................................................................... 39
  4.3.3. Polynomial Regression Model .......................................................................................... 42
  4.3.4. Models Comparison ............................................................................................................ 44
  4.4. Run Example ........................................................................................................................... 45
4.5. Analytics Engine Class Diagram ................................................................................. 47
4.6. Summary .................................................................................................................. 48
Chapter 5. QUEUING ENGINE ...................................................................................... 49
Chapter 6. RANDOM LOAD SIMULATION .................................................................... 53
6.1. Discrete vs. Continuous .......................................................................................... 53
6.2. Pseudo-random Number Sampling ......................................................................... 53
6.2.1. Normalization ..................................................................................................... 54
6.3. Results of First Run ............................................................................................... 55
6.3.1. CPU Utilization ................................................................................................ 56
6.3.2. Response Time Per Thread ................................................................................ 57
6.3.3. Heap Ratio ........................................................................................................ 58
6.4. Results of Second Run ........................................................................................... 59
6.4.1. CPU Utilization ................................................................................................ 61
6.4.2. Response Time Per Thread ................................................................................ 62
6.4.3. Heap Ratio Per Thread ...................................................................................... 63
6.5. Results of Third Run .............................................................................................. 64
6.5.1. CPU Utilization ................................................................................................ 65
6.5.2. Response Time Per Threads ................................................................................. 66
6.5.3. Heap Ratio Per Thread ...................................................................................... 67
Chapter 7. CONCLUSION ............................................................................................... 69
REFERENCES ................................................................................................................. 71
Appendix A .................................................................................................................... 74
A.1. Estore Schema Create Statement ........................................................................... 74
A.2. Address Table Create Statement ........................................................................... 74
A.2.1. Description ........................................................................................................ 74
A.2.2. SQL Statement .................................................................................................. 79
A.3. Inventory Table Create Statement ......................................................................... 80
A.3.1. Description ........................................................................................................ 80
A.3.2. SQL Statement .................................................................................................. 81
A.4. Language Table Create Statement ......................................................................... 81
A.4.1. Description ........................................................................................................ 81
A.4.2. SQL Statement .................................................................................................. 84
A.5. Listprice Table Create Statement .......................................................................... 84
A.5.1. Description ........................................................................................................ 84
LIST OF FIGURES

Figure 1 – Correlated Components of the Cognitive Performance Application..........................4
Figure 2 – Cognitive Performance Application Diagram..........................................................5
Figure 3 – Transaction Response Time......................................................................................8
Figure 4 – Online Store Database Schema................................................................................9
Figure 5 – Class Diagram for Java Object Populating Database.............................................10
Figure 6 – JVM Heap Ratio Chart............................................................................................14
Figure 7 – CPU Utilization and Response Chart........................................................................14
Figure 8 – JVM Heap Ratio Chart............................................................................................17
Figure 9 – CPU Utilization and Response Charts......................................................................17
Figure 10 – HP2B Site Page View Stats.....................................................................................20
Figure 11 – JVM Heap Ratio.....................................................................................................22
Figure 12 – CPU Utilization and Response Time Charts..........................................................22
Figure 13 – Learning Engine.......................................................................................................24
Figure 14 – Implemented Neural Network................................................................................25
Figure 15 – Neural Network Systematic Learning...................................................................27
Figure 16 – MSE per Epoch.........................................................................................................31
Figure 17 – Learning Engine Class Diagram.............................................................................32
Figure 18 – One Flavor of the Analytics Engine.........................................................................34
Figure 19 – Analytics Engine Example Run..............................................................................46
Figure 20 – Analytics Engine UML Class Diagram.................................................................47
Figure 21 – Java Queuing Engine...............................................................................................50
Figure 22 – System CPU Utilization Charts.............................................................................57
Figure 23 – Response Time in Seconds per Thread.................................................................58
Figure 24 – Heap Ratio per Thread............................................................................................59
Figure 25 – System CPU Utilization Charts.............................................................................61
Figure 26 – Response Time in Seconds per Thread.................................................................62
Figure 27 – Heap Ratio per Thread............................................................................................63
Figure 28 – System CPU Utilization Charts.............................................................................65
Figure 29 – Response Time in Seconds per Thread.................................................................66
Figure 30 – Heap Ratio per Thread............................................................................................67
LIST OF TABLES

Table 1 - CPU Utilization demand simulation snapshot Summary ........................................13
Table 2 - CPU Utilization demand simulation summary ......................................................13
Table 3 - Memory Utilization demand snapshot summary ..................................................16
Table 4 - Memory Utilization demand simulation summary ..............................................16
Table 5 - HP2B page demands ..........................................................................................19
Table 6 - HP2B page demand percentage per user ............................................................19
Table 7 - HP2B simulations run summary .........................................................................21
Table 8 - HP2B simulations run summary .........................................................................21
Table 21 - CPU Per User Session .....................................................................................45
Table 10 - HP2B daily user load .........................................................................................55
Table 11 - First Run Summary ..........................................................................................56
Table 12 - First Run CPU data Summary ..........................................................................57
Table 13 - First run thread response time summary ..........................................................58
Table 14 - First run heap ratio summary ..........................................................................59
Table 15 - Second Run Summary .....................................................................................60
Table 16 - Second run CPU data summary .......................................................................61
Table 17 - Second run thread response time data summary ............................................62
Table 18 - Second run heap ratio data summary ...............................................................63
Table 19 - Third run data summary ..................................................................................65
Table 20 - Third run CPU utilization data summary ..........................................................65
Table 21 - Third run response time data summary .............................................................66
Table 22 - Third run heap ratio data summary ..................................................................67
Applications running on Java Virtual Machine (JVM) are extensively adopted in numerous industrial and commercial applications around the world. The main advantage in Java is the concept of parallel programming or multiple threading and the fact that it can run in different operating system without the need to change the coding syntax. In the commercial world, Java is mainly known for its web programming capabilities. While this is all true and considered redundant info, Java applications running on multiple JVM’s on single or multiple servers usual end up exploiting common resources such as a database management system (DBMS). Database management systems are more complex, more rapidly evolving, and more essential to the conduct of business than those of even a few years ago. The result is an increasing need for techniques that assist in optimizing the overall performance of these database driven systems in order to enhance the system availability and the overall user experience in a world where website up times and mobile apps are now expected to have 99.99% around the clock availability. The need for new techniques that help these systems avoid bottleneck situations in a world where technical glitches translate into a negative user experience and therefor a lost business opportunity. The challenge is that relational databases were not designed to be scaled in distributed model, such as the cloud architecture, which leads the entire solution to halt or crash in many situations. In this paper we will use a customized database known as an e-store to simulate real-world commercial Java application such as an e-commerce store that rely heavily on DB integration.
There are many theoretical and practical proposed techniques that stopped short from providing a comprehensive solution to the current challenges stated above. This document roles off some of these proposals, and builds upon other ones, such as “A Queuing Model to Achieve Proper Elasticity for Cloud Cluster Jobs,” [1] which focuses on providing an understanding of the database behavior through the introduction of an analytical model based on queuing theory to determine at any given time the workload conditions and the minimum number of computing resources needed for executing query jobs on a cloud cluster. Another example of academic work that helped shaping the direction of our solution is the “Performance Modeling and Analysis of a Database Server with Write-Heavy Workload,” [2] which studies the performance anomalies dynamics that are difficult to monitor and control, through the development of a queuing based performance model for database servers with write-heavy workload. In both papers and in other papers such as “Solving Enterprise Applications Performance Puzzles - Queuing Models to the Rescue,” [3] the authors use a queuing model to rectify the system performance which we will prove is necessary and will constitute some portion of our work as explained in the subsequent sections of this document.

On the other hand, we also examined more conventional or widely adopted commercial solutions related to the same challenges, and we found that they mainly evolve around four main concepts listed below:

- Increase Memory
- Caching,
- Query optimization,
- Splitting databases
These solutions are not always the optimum from performance or cost prospective. For example, a caching solution ends up replicating the database into another database, which runs into technical issues of caching synchronization and validation and invalidation. The practice of query optimization constitutes another hurdle especially for highly integrated complicated queries that were evolving over time to serve multiple enterprise agents. Finally, the option of splitting the system database into two always introduces more complexities such as keeping the enterprise database systems synchronized.

This document discusses a cognitive or an AI alternative approach to autonomously enhance the system availability and optimize the performance of a JVM interacting with common computing resources, such as a database management system. An example of such a database driven system would a Java implementation of an online store deployed on a single or multiple servers utilizing highly paralyzed query jobs resulting from multiple and concurrent user sessions. The proposed solution will rely on machine learning practices in order to introduce an integrated cognitive framework. As shown in the below figure, the Cognitive Performance Application solution proposes three correlated phases or engines. These are

- Learning Engine,
- Analytics Engine,
- Queuing Engine.
The first component would learn how to determine or detect a JVM performance bottleneck, by monitoring correlated parameters such as CPU utilization and query response times using neural network technology. Meanwhile, solution package will also construct an advanced analytics-based alert mechanism to predict, based on historical performance records, the future bottleneck situations where the demand for the database and the other system computing resources is trending up. The high demand in a real world would have positive correlations with external library of factors such weather conditions or shopping seasonality. Finally, and based on the outcome of the learning and predictive components of this solution, a signal will be triggered for the system to rectify the bottleneck situation through building a temp queuing structure to stream down the system demand. The queuing component is elastic in nature will then dissolve on its own once the system recovers back to normal operation levels as the performance conditions improve. The process flow of the system and the different functionalities of the cognitive application components are demonstrated on the below graph, and are subject to further explanation in the subsequent segments of this document.
The uniqueness of this approach lies in the fact that it can be generalized to other performance problem dealing with high demand for common resources. Same approach also can be scaled horizontally to other common resources utilized by Java applications such file systems, static contents, search engines, web services…etc. Moreover, it widens the dimension of solving conventional performance challenges of database driven systems without further exhausting customary solutions, but rather shifting the focus up to the requestor layer to queue up computing jobs intelligently and only when needed to avoid bottleneck consequences. In addition, the proposed solution has a high viability for limited budget implementations, compared to other usual solutions that mandate investing on scaling the hardware specifications. Meanwhile the proposed solution has another advantage on areas like maintenance reduction and eliminating partially or totally.
interruption system admins, who would otherwise consistently monitor the system performance.

This document is structured on different chapters, the first of which focuses on building the components of a typical modern ecommerce store database system, based on a scaled down version of IBM WebSphere e-commerce database schema [5]. The sample store database records are populated with millions of customers, addresses, shopping carts…etc. database records. Also in the first chapter we list the system specifications under study, and then simulate the system demand with constant user load in order to take snapshots of the different performance parameters enhancing our understanding of the different correlations among them and when the system reaches a bottleneck. The document then moves to other chapters describing the three components of the Cognitive Performance Package and dives into more details regarding architecture decision, logic, code snippets, integration, statistical calculations …etc. Toward the end of this document we run different random simulations of the user load to test our solution and compare the system performance with and without our solution. We finally add another section concluding our findings and recommendations.
CZ Afk Æfl2 CONSTANT LOAD SIMULATION

2.1. Simulation

A critical part of this development is to build a Java based framework to simulate a discrete system demand, as in the number of concurrent threads competing for the CPU time, which translate to number of active user sessions accommodated by the online ecommerce store server over a particular time range. To achieve this goal, the simulation model abstracts a queuing network model of a standalone system that receives and serves a group of parallel database queries (tasks) in an attempt to distil, from the mass of details that is the system itself, exactly those aspects that are essential to the system’s behavior. Modeling provides a framework for congregating, organizing, evaluating, and understanding information about a current system subject to this study. Once a model has been defined through this abstraction process, it can be parameterized to reflect any of the alternatives under study, and then evaluated to determine its performance under this alternative. [6] The objective using this approach is to quantify a model, which accurately reflect the performance measures of the online store.

This simulation framework will further rely heavily on the integration between the online store and the database management system, in order to make it a query driven simulator system augmenting the likelihood of matching a typical deployment of an online store. The usage of Java multi-threading feature will make it conceivable to simulate an independent entity, or a user session browsing the system, while timing between independent database activity during certain time slots will be randomized using a normal
distribution. In the queue-based system, the JVM connected to a DBMS is considered the service center where the work gets done. To accomplish a given task, it takes the JVM, a certain amount of time. If a task arrives faster than it can be processed, a queue builds and the response time grows as shown below.

![Diagram showing transaction response time](image)

**Wait Time + Service Time = Response Time**

*Figure 3 – Transaction Response Time*

As the demand on the JVM mountains, the CPU utilization increases and it becomes more likely that a newly arriving query will have to wait since there are other queries ahead. In general, the response time degradation is more pronounced the busier the resource is. It is the responsibility of the Learning Engine to benchmark the performance metrics over a configurable period of time. In this application a number of threads will be measured against response times, memory usage, and CPU utilizations. The simulation framework will utilize different types of SQL transactions, each of which will result in different response times. The Learning Engine will be a supervised model that detects the bottleneck based on the previously mentioned system performance parameters. More of these details are covered in the next subsections.
2.2. Populating the database

To start simulating a real ecommerce solution, we will start by building and populating a similar ecommerce database. Below is the database schema diagram that is build to mimic a very known industrial ecommerce store, namely IBM WebSphere e-commerce. [5]
A detailed description of each database table is covered in Appendix A. Worth mentioning that part of the efforts to build a simulator was to build Java classes that use JDBC driver to connect to the DBMS and utilize “insert” SQL statements. These classes would run for one time in order to initialize or populate hundreds or thousands of database records. All these classes are included in a Java package named “edu.louisville.cs.db.” In this section, we will not go into detail describing each method in these classes, as they are insignificant to overall goal. However, a class diagram summarizing these classes is displayed below.

![Class Diagram for Java Object Populating Database](image-url)

*Figure 5 – Class Diagram for Java Object Populating Database*
2.3. System Specifications and Simulation

This section will cover how to simulate a snapshot of the system performance. We will start first by detailing the platform specification used for this study.

- The system is Windows Server 2008 based.
- Runs on four processors cores
- Has 6144 MB of RAM
- Has 53GB of hard disk
- Has MySQL database V.56
- Has JVM V.1.7

To simulate a demand of single server, we need to consider the application server topology. In real commerce implantation, the production environment consists of one or more application server(s). Each of which has single or multiple server nodes, each node has shared pooled of database connections to be distributed across the different user web sessions. The ecommerce site requires enough JDBC pool connections in order to insure that no threads need to wait for an available database connection. For consideration regarding the user experience, the number of database connections is set to be relative or equal to the number of allowed concurrent active web sessions. In a typical active web session, users would login and become authenticated against the database, search products and retrieve their information from database, and finally proceed to the checkout flow where the entire order transaction will again be recorded in the database. Through the user journey, there is only one database connection through which different database transactions (queries) are executed. In other words, to simulate the demand of an online store hosted on a singled server, the simulation system would utilize multi-concurrent
threads. Each of these Java threads represents an active web session. Each thread will establish a new JDBC connection and would run number of queries under each connection.

2.4. Demand Simulation Snapshot and CPU Utilization

To study the demand of the system against the response time and the CPU utilization and in isolation from the heap allocation or the memory utilization, we will start by running 100-concurrent threads in order to record a snapshot of the system performance. In order understand the system behavior that could lead to a bottleneck situation in regard to only the CPU utilization and query response time and in isolation of the memory allocation, each of these 100 parallel threads will use a simple insert database query, to avoid substantial returned result, and hence avoid substantial memory allocation of the JVM heap. Same query will run for certain number under each database connection initialized in each thread. We will record the following performance metrics: number of threads in memory competing for the database resources, the system CPU utilization percentage, response time, start time of each thread, end time of each thread, JVM heap ratio, allocated heap memory in KB, Max Heap Memory in KB, and finally the number of threads served. Each of these metrics will be gathered and recorded toward the end of each thread execution. In addition, data summary will be calculated to include mean, standard deviation, median, maximum, and minimum data points. The data summary will be used later on to normalize the input to the neural network as described in subsequent sections. The complete result of the snapshot run is shown in the below table.
<table>
<thead>
<tr>
<th>Threads in the Memory</th>
<th>CPU Utilization</th>
<th>Response Time (MS)</th>
<th>Start Time</th>
<th>End Time</th>
<th>JVM Heap Ratio</th>
<th>Allocated Memory (KB)</th>
<th>Max Memory (KB)</th>
<th>Threads Served</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>55.90%</td>
<td>64.7394</td>
<td>9471.1332</td>
<td>9535.8726</td>
<td>0.00599</td>
<td>1518</td>
<td>253440</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>39.60%</td>
<td>98.8712</td>
<td>9471.4200</td>
<td>9570.2912</td>
<td>0.00599</td>
<td>1518</td>
<td>253440</td>
<td>2</td>
</tr>
<tr>
<td>98</td>
<td>28.53%</td>
<td>169.4794</td>
<td>9471.6857</td>
<td>9641.1651</td>
<td>0.01052</td>
<td>2667</td>
<td>253440</td>
<td>3</td>
</tr>
<tr>
<td>97</td>
<td>23.49%</td>
<td>222.4609</td>
<td>9472.3764</td>
<td>9694.8373</td>
<td>0.01401</td>
<td>3550</td>
<td>253440</td>
<td>4</td>
</tr>
<tr>
<td>96</td>
<td>23.83%</td>
<td>230.4046</td>
<td>9471.8583</td>
<td>9702.2629</td>
<td>0.01266</td>
<td>3208</td>
<td>253440</td>
<td>5</td>
</tr>
<tr>
<td>95</td>
<td>23.94%</td>
<td>237.8465</td>
<td>9471.2631</td>
<td>9709.1096</td>
<td>0.01250</td>
<td>3167</td>
<td>253440</td>
<td>6</td>
</tr>
<tr>
<td>94</td>
<td>22.17%</td>
<td>259.6832</td>
<td>9472.1675</td>
<td>9731.8507</td>
<td>0.01256</td>
<td>3184</td>
<td>253440</td>
<td>7</td>
</tr>
<tr>
<td>93</td>
<td>20.75%</td>
<td>289.4151</td>
<td>9472.0572</td>
<td>9761.4723</td>
<td>0.00599</td>
<td>1518</td>
<td>253440</td>
<td>8</td>
</tr>
<tr>
<td>92</td>
<td>20.32%</td>
<td>300.1542</td>
<td>9472.0468</td>
<td>9772.2010</td>
<td>0.01664</td>
<td>415</td>
<td>253440</td>
<td>9</td>
</tr>
<tr>
<td>91</td>
<td>20.11%</td>
<td>316.4318</td>
<td>9471.7506</td>
<td>9788.1824</td>
<td>0.01297</td>
<td>3288</td>
<td>253440</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1-CPU Utilization demand simulation snapshot Summary

2.4.1. Demand Simulation Snapshot Run Summary

<table>
<thead>
<tr>
<th>CPU Utilization</th>
<th>Response Time</th>
<th>Heap Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>13.21%</td>
<td>64.73</td>
</tr>
<tr>
<td>Max</td>
<td>55.90%</td>
<td>1094.3</td>
</tr>
<tr>
<td>Stand Dev</td>
<td>5.48%</td>
<td>728.04</td>
</tr>
<tr>
<td>Average</td>
<td>16.10%</td>
<td>579.544</td>
</tr>
<tr>
<td>Median</td>
<td>14.49%</td>
<td>579.544</td>
</tr>
</tbody>
</table>

Table 2-CPU Utilization demand simulation summary
From the previous records, we could easily spot a correlation between the number of threads, or web session, and the overall system performance. We can conclude with great degree of confidence that the system performance in terms of CPU utilization improves as
the number of threads competing for the system services decreases. In other words, the system could reach a deadlock as more threads are competing for its resources. This fact, as apparent as it is, will justify our future work to rectify the system’s bottleneck situation by including a queue component in order to manage or stream down elastically an overwhelming system demand once benchmarked and detected.

2.5. Demand Simulation Snapshot and Memory Utilization

To further study the bottleneck of a system and its relation to a high memory allocation, or a high JVM heap utilization as in our case, we will run the same 100-concurrent threads in order to record a snapshot of the system performance. Each of these threads will use a search database query, where the returned result is big enough to cause higher object allocation in the heap. Fresh search query will run for certain number under each database connection included in each thread. We will record the same previous performance metrics: number of threads in memory competing for the database resources, CPU utilization, response time, start time of each thread, end time of each thread, JVM heap ration, allocated heap memory in KB, Max Heap Memory in KB, number of threads served. A data summary from average to standard deviation will also be included in a separate table.

<table>
<thead>
<tr>
<th>Threads in the Memory</th>
<th>CPU Utilization</th>
<th>Response Time (MS)</th>
<th>Start Time</th>
<th>End Time</th>
<th>JVM Heap Ratio</th>
<th>Allocated Memory (KB)</th>
<th>Max Memory (KB)</th>
<th>Threads Served</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100.00%</td>
<td>74.10382</td>
<td>1593.50065</td>
<td>1667.60447</td>
<td>0.96275</td>
<td>244</td>
<td>253.44</td>
<td>1</td>
</tr>
<tr>
<td>99</td>
<td>100.00%</td>
<td>95.75259</td>
<td>1593.64834</td>
<td>1689.40093</td>
<td>0.96275</td>
<td>244</td>
<td>253.44</td>
<td>2</td>
</tr>
<tr>
<td>98</td>
<td>100.00%</td>
<td>116.71327</td>
<td>1593.10269</td>
<td>1709.81596</td>
<td>0.76941</td>
<td>195</td>
<td>253.44</td>
<td>3</td>
</tr>
<tr>
<td>97</td>
<td>98.63%</td>
<td>134.75497</td>
<td>1593.06380</td>
<td>1727.81877</td>
<td>0.94697</td>
<td>240</td>
<td>253.44</td>
<td>4</td>
</tr>
<tr>
<td>96</td>
<td>93.77%</td>
<td>161.04976</td>
<td>1593.43684</td>
<td>1754.48660</td>
<td>0.80887</td>
<td>205</td>
<td>253.44</td>
<td>5</td>
</tr>
<tr>
<td>95</td>
<td>91.87%</td>
<td>176.59385</td>
<td>1593.44190</td>
<td>1770.03575</td>
<td>0.91935</td>
<td>233</td>
<td>253.44</td>
<td>6</td>
</tr>
</tbody>
</table>
### Table 3 - Memory Utilization demand snapshot summary

#### 2.5.1. Demand Simulation Snapshot Run Summary

<table>
<thead>
<tr>
<th>Min</th>
<th>Max</th>
<th>Stand Dev</th>
<th>Average</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td>61.85%</td>
<td>100.00%</td>
<td>10.04%</td>
<td>69.46%</td>
<td>65.06%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU Utilization</th>
<th>Response Time</th>
<th>Heap Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>61.85%</td>
<td>74.10</td>
</tr>
<tr>
<td>Max</td>
<td>100.00%</td>
<td>1288.75</td>
</tr>
<tr>
<td>Stand Dev</td>
<td>10.04%</td>
<td>858.88</td>
</tr>
<tr>
<td>Average</td>
<td>69.46%</td>
<td>681.43</td>
</tr>
<tr>
<td>Median</td>
<td>65.06%</td>
<td>681.42</td>
</tr>
</tbody>
</table>

### Table 4 - Memory Utilization demand simulation summary
Figure 8 – JVM Heap Ratio Chart

Figure 9 – CPU Utilization and Response Charts
From the previous results it shows that the high object allocation in the JVM heap is not as significant factor for the system to reach a bottleneck situation. This could be contributed to the fact that the JVM heap under this study only occupy smaller space of the system RAM or the hard disk storage and hence saturating the heap size is less significant on causing a system bottleneck. However, it is evident that larger object allocations lead to higher the CPU utilization. Also in real application a large heap size could lead to an out of memory outages. The result from this section and the previous section will justify the use of queuing system to rectify an overwhelming system demand and will also justify the fact that we monitor the heap utilization as a performance metric and added as a parameter to the learning engine to detect a bottleneck situation when it reaches more than 90%.

2.6. Simulation of real ecommerce user sessions

To simulate a real-life user web session on an ecommerce store in respect to the database load, we collected a real production data from a Business-to-Business (B2B) ecommerce store to understand the type of queries rendered through the user journey on the online store. We first started by studying the average user requests per day. The below data table was collected from a newly published “HP2B” online ecommerce store. The table below is usually collected on daily basis by a monitoring tool for the purpose of showing the page load time as a mean of collecting daily performance snapshot of the production system and to understand where the user may face potential issue while processing HTTP requests. The “Load Time(s)” column is an average of all the requests targeted the system on that specific day. The “Beacons” column shows the number of user requesting URL’s for the associated webpages shown in the “Page Group” table.
<table>
<thead>
<tr>
<th>Row</th>
<th>Page Group</th>
<th>Load Time (s)</th>
<th>Beacons</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Home Page</td>
<td>7.15</td>
<td>4,164</td>
</tr>
<tr>
<td>2</td>
<td>Product Detail Page</td>
<td>10.57</td>
<td>2,476</td>
</tr>
<tr>
<td>3</td>
<td>Login Page</td>
<td>3.63</td>
<td>1,347</td>
</tr>
<tr>
<td>4</td>
<td>Search Page</td>
<td>2.63</td>
<td>765</td>
</tr>
<tr>
<td>5</td>
<td>Others</td>
<td>6.08</td>
<td>663</td>
</tr>
<tr>
<td>6</td>
<td>Product Listing Page</td>
<td>6.3</td>
<td>480</td>
</tr>
<tr>
<td>7</td>
<td>View Cart Page</td>
<td>4.67</td>
<td>372</td>
</tr>
<tr>
<td>8</td>
<td>Checkout Page</td>
<td>4.97</td>
<td>127</td>
</tr>
</tbody>
</table>

Table 5 - HP2B page demands

To better understand the above table, another pivot table and chart, shown below, are derived from the above data table, to plot the relation between the page type and the user request. From the below derived table we found that during the user journey from the login page to the checkout page, most users spend the majority of their web session times on the home page, interacting with their account details, more than they spend time buying or submitting an order.

<table>
<thead>
<tr>
<th>Row</th>
<th>Page Group</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Home Page</td>
<td>40.06%</td>
</tr>
<tr>
<td>2</td>
<td>Product Detail Page</td>
<td>23.82%</td>
</tr>
<tr>
<td>3</td>
<td>Login Page</td>
<td>12.96%</td>
</tr>
<tr>
<td>4</td>
<td>Search Page</td>
<td>7.36%</td>
</tr>
<tr>
<td>5</td>
<td>Others</td>
<td>6.38%</td>
</tr>
<tr>
<td>6</td>
<td>Product Listing Page</td>
<td>4.62%</td>
</tr>
<tr>
<td>7</td>
<td>View Cart Page</td>
<td>3.58%</td>
</tr>
<tr>
<td>8</td>
<td>Checkout Page</td>
<td>1.22%</td>
</tr>
</tbody>
</table>

Table 6 - HP2B page demand percentage per user
To map these results to the system under study, we draw a relation between each page and its association with the database tables. The home page and the login page account for around 53% of the web session load. And both are associated with the member and address tables. Product Detail Page (PDP) accounts for 25% of the web session load and is associated with the product and inventory tables. The search and the product listing page (PLP) accounts of about 10% of the session load and again are associated with the product and inventory tables. The view cart and the checkout pages account for 5% of the session load and is associated with the “order” table and so on. To simulate the database demand during the user web session, we will query the database, in each Java thread, number of times relative to the percentages of the user web session load calculated above and we will run a new snapshot as shown below. We expect the new reading to be in alignment of the previous readings specially those from the very last section with different variation.
### Table 7 - HP2B simulations run summary

#### 2.6.1. Demand Simulation Snapshot Run Summary

<table>
<thead>
<tr>
<th></th>
<th>CPU Utilization</th>
<th>Response Time</th>
<th>Heap Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Min</strong></td>
<td>69.94%</td>
<td>26.98</td>
<td>15.0000</td>
</tr>
<tr>
<td><strong>Max</strong></td>
<td>100.00%</td>
<td>699.255</td>
<td>25.0000</td>
</tr>
<tr>
<td><strong>Stand Dev</strong></td>
<td>4.42%</td>
<td>475.37</td>
<td>2.96573</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>72.61%</td>
<td>363.11</td>
<td>20.1800</td>
</tr>
<tr>
<td><strong>Median</strong></td>
<td>71.40%</td>
<td>363.11</td>
<td>20.0000</td>
</tr>
</tbody>
</table>

#### Table 8 - HP2B simulations run summary
Figure 11 – JVM Heap Ratio

Figure 12 – CPU Utilization and Response Time Charts

Figure 12 – CPU Utilization and Response Time Charts
2.7. Summary

In this chapter, we took three snapshots of the system to understand the correlation between certain parameters and overall system performance. We also concluded that there is a direct relation or dependency between the number of active resources competing for the processor time, the heap utilization or the memory allocation, on one hand, and the CPU utilization and the response time, on the other hand. The results from these snapshots will constitute all of our future work in the next chapters.
LEARNING ENGINE

This engine monitors different system metrics and learns when to benchmark and detect a system bottleneck. The learning engine monitors the performance metrics of the requestor system, where the multi-threading Java application is running, and the provider system where the common resources run, i.e. the DBMS in our case. The Learning Engine will continuously monitor performance metrics such memory usage and CPU utilizations and will be able to adapt to any changes in the monitored system (the provider system) to accurately identify a bottleneck situation. Once this monitoring model is benchmarked, it will assess and quantify the relation between performance metrics and the occurrences of a bottleneck.

To achieve this goal, continuous learning and adaptability, learning engine will use a lightweight implementation of Artificial Neural Network (ANN). ANN will assign random
interconnection weights for the performance metrics. i.e. response time, CPU usage, and memory utilizations, by applying a set of training or learning samples. The final effects of a learning process are tuned parameters of a network the will help point to a bottleneck situation. Moreover, it will be able to figure out the right weights of the performance input parameters in cases where an ambiguity surround whichever performance metrics has the most effect on the system performance. The lightweight is also a requirement for the Cognitive Performance Application not to constitute a performance burden on the host system.

3.1. Neural Network Type

The neural network implemented in our application is of a type feedforward. Meaning the flow of the signal is on one direction. The input signals are fed into the input layer and then forward to the next layer. The network learning is also a supervised because the objective is to map input data (X’s) to an output (Y) through a learning function f.

\[ F: X \rightarrow Y \] where Y data acts as the supervisor. [7]

The below diagram shows the neural network implemented.
In this neural network there is one input layer and one hidden layer following monolayer network architecture for simplicity and outmost performance.

3.2. Neural Network Input

The input data in our case will be the CPU utilization ranging percentage and the JVM heap ratio. Both inputs normalized and are ranging from 0 to 1. The output on the other hand will be either 0 or 1 as a classification of the system bottleneck, where 1 is true and 0 is false. Training data will be provided for the neural network at the time of initialization as in two-dimension array shown below.

```java
float[][] trainingData = new float[][] {
    {0.70f, 0.73f }, {1.0f, 0.81f }, {0.1f, 0.86f },
    {0.90f, 0.95f }, {1.0f, 0.45f }, {0.2f, 0.70f },
    {0.95f, 0.51f }, {0.6f, 0.89f }, {0.3f, 0.79f },
    {0.93f, 0.51f }, {0.5f, 0.89f }, {0.4f, 0.79f },
    {0.91f, 0.71f }, {0.89f, 1.0f }, {0.9f, 0.39f },
    {0.92f, 0.1f }, {0.93f, 1.0f }, {0.94f, 0.4f },
    {0.88f, 0.37f }, {0.87f, 0.87f },{0.86f, 0.76f},
};
```

The actual results of these learning input will be also provided as one dimensional array of zero’s and one’s as shown below.

```java
int[] actuals = new int[]{
    0,1,0, 1,1,0,
    1,0,0, 1,0,0,
    1,0,1, 1,1,1, 0,0,0};
```

Worth mentioning, that the input data in this above form satisfies both the vectorization and normalization prerequisites of artificial neural network input data. [8] Below code shows how input and output weights, and bias are randomly initialized.

```java
private void initWeights(int neurons, int dimension) throws ZeroNeuronsException,
ZeroInputDimensionException {
    if (neurons == 0)
        throw new ZeroNeuronsException();
    if (dimension == 0)
        throw new ZeroInputDimensionException();
    for (int i = 0;i<neurons;i++){
```
3.3. Learning procedure

The below flow diagram summarizes the neural network systematic learning process.

Learning procedure is iterative process and is controlled or stopped when reaching the maximum number of iterations (private int maxEpochs = 10000;). Another important parameter is the learning rate (float learningRate = 0.05f;), which dictates how strongly the neural network would vary in the weights’ hyperspace. [8] ANN will check it is response or result against the input data and will calculate how far off the results (float error = actual - fOut). Then it will use the “Delta rule”, which is an algorithm based on the gradient descent method to account for nonlinearity, to determine how the input and
output weights (W) and biases will be updated. The complete method implementation is shown below.

```java
private void initLearn(float actual, float fOut, float[] expectedOutput, float[] outWeights, float[][] inputWeights, float[] bias, float bOut, int neurons, float[] input, int dimension) {
    float error = actual - fOut;
    float learningRate = 0.05f;
    float dv;
    float[] dw = new float[neurons];
    float[][] dwi = new float[dimension][neurons];
    float[] dbi = new float[neurons];
    float[] db = new float[neurons];

    // Modify out weights
    dv = fOut * (1 - fOut) * error;
    for (int i = 0; i < neurons; i++){
        this.outWeights[i] = outWeights[i] + learningRate * dv * expectedOutput[i];
    }

    // Modify out bias out
    float dbOut = learningRate * dv * 1;
    this.bOut = (bOut + dbOut);

    // Modify input weights
    for (int i = 0; i < neurons; i++){
        dw[i] = expectedOutput[i] * (1 - expectedOutput[i]) * outWeights[i] * dv;
        for (int j = 0; j < dimension; j++){
            dwi[j][i] = learningRate * dw[i] * input[j];
            this.inputWeights[j][i] = inputWeights[j][i] + dwi[j][i];
        }
    }

    // Modify input bias
    for (int i = 0; i < neurons; i++){
        dbi[i] = expectedOutput[i] * (1 - expectedOutput[i]) * outWeights[i] * dv;
        db[i] = learningRate * dbi[i] * 1;
        this.bias[i] = bias[i] + db[i];
    }
}
```

3.4. Sigmoid Activation Function

The activation function calculates a “weighted sum” of its input, adds a bias as below

\[ \Sigma (weight \times input) + bias \]

This is implemented as shown below

```java
private float calculateFOut(float[] x){
    for (int i = 0; i < neurons; i++){
        float sum = 0;
        for (int j=0; j<dimension; j++){
            sum = sum + (x[j] * wWeights[j][i]);
        }
        this.fOutArray[i] = ActivationFunction.activate(sum + bias[i]);
    }
}
```
And then decides whether it should be “activated” or not using the output of the following equation:

\[ A = \frac{1}{1 + e^{-x}} \]

, which is implemented as shown below

```java
public class SigmoidFunction implements ActivationFunction {
    @Override
    public float activate(float value) {
        return (float)(1/(1+Math.exp(-value)));
    }
}
```

As the outcome of classification problem is binary, we also added “parser” to convert the output to only 1 if the result is above (0.5) and zero otherwise as shown below.

```java
@Override
public Integer parseResult(float result) {
    return (result < 0.5) ? 0 : 1;
}
```

### 3.5. Neural Network Classification Method

Once the neural network has learned, or the final values of the weights and bias have been declared, we surround the method described earlier “calculateFOut” with another wrapper to restrict the output to the only zero’s and ones’ as shown below.

```java
@override
public Integer parseResult(float result) {
    return (result < 0.5) ? 0 : 1;
}
```
For simplicity we also wrap the previous method with another wrapper to restrict the output to true and false corresponding to the ones and zeros concluded in the previous method. The true indicates a “deadlock” classification and false otherwise.

```java
public boolean classify(double CPUUtilization, double Heap) {
    float[] valueToPredict = new float[] { (float)CPUUtilization, (float)Heap };
    if (result.classifyValue(valueToPredict)==1)
        return true;
    else return false;
}
```

### 3.6. Neural Network Kernel

The below code is the heart of the learning engine. It initializes the neural network, sets the number of neuron and the number of the hidden layers, and the activation method. It also randomly assigns values for its input and output weights and bias. Then loops number of times relative to the number of epochs parameter and calls both analyzer and the learner classes. The “analyzer” sums the weights and the input values, and then calculates the expected output values using the activation method. The “learner” calculates the error by subtracting the actual output from the expected output and updates the weights and bias using the delta rule. The code is shown below.

```java
public class NeuralNetworkThread implements Runnable {
    @Override
    public void run() {
        float quadraticError = 0;
        float MSE = 0;
        float[] f;
        int success = 0;
        for (int i = 0; i<maxEpochs; i++) {
            success = 0;
            for (int z = 0; z<nElements; z++) {
                analyzer = new Analyzer (getRowElements(z), inputWeights, bias,
                                        outWeights, bOut, neurons, activateFunction, dimension);
                f = analyzer.getFOutArray();
                fOut = analyzer.getFOut();
                learner = new Learner (outputs[z], fOut, f, outWeights,
                                        inputWeights,
                                        bias, bOut, neurons, getRowElements(z), dimension);
                outWeights = learner.getVWeights();
            }
        }
    }
}
```
3.7. Mean Squared error

To show how accurate the neural network becomes, we plotted the below diagram using the MSE data over the complete number of iterations. It is easy to note that MSE decreases as the number of epoch’s increases to almost near a zero value.

![MSE per Epoch](image)

*Figure 16 – MSE per Epoch*
3.8. Learning Engine Class Diagram

The below graph a UML class diagram summarizing the structure of the learning engine.

Figure 17 – LEARNING ENGINE CLASS DIAGRAM
The second component of the Cognitive Performance Application (CPA) is the Analytics engine. The Analytics Engine is a core component of the system that will act primarily as an early alert mechanism that will predict the situation leading to a system deadlock. This component will work continuously in the backend and will keep autonomously correcting its predictions results with the ability to adapt to the different changes in the system specifications. The system will collect a library of factors, which feed through the application in order to establish a pattern of behavior pertaining to the overall performance. At this point, the application usually depends on a human interference for configuring and integrating the feeds of this library of factors into the applications. To establish such a pattern, the input of domain experts usually are needed to decide on the input parameters of the system performance predictive model in regard to forecasting the online commerce store future demand. In real world the demand of a particular eCommerce store could be a subject of multiple correlated relations with metrics that can be collected externally or internally from:

- Current user on the store
- Commerce stores database, such as the trending of daily average of order transactions number
- Weather company, such as feeds of daily temperatures, or the mark and end of weather seasons
- Social media buzz, such as twitter
  - We can also extract other factors such as political stability, social rest…and other metrics to predict global demand.
The previously mentioned feeds can be integrated in many ways such as be implementing a scheduled service-oriented architecture (SOA) calls, where services are provided to the other components through a communication protocol over a network, or alternatively through an Extract, Transform, and Load (ETL) architecture. The architecture for the CPA will be flexible to be extended to different patterns of feeds i.e. either SOA or ETL.

In our implementation, we will use the internal user load to predict the future demand in terms of the CPU utilization. If the future CPU utilization if above certain threshold the analytics engine will trigger a signal, in combination of the learning engine signal, to the queue system to take the necessary rectifying methods.
4.1. Factory Design Pattern

Learning engine will be based on a Factory design pattern. Factory design pattern is used to create objects or Class in Java and it provides loose coupling and high cohesion. Factory pattern encapsulate object creation logic which makes it easy to change it later when you change how object gets created or you can even introduce new object with just change in one class. The obvious reason for this architecture decision is that we will have different analytics models that we will choose among them to initialize our learning engine. These analytics models are

1- Linear Regression Model
2- Multiple Linear Regression Model
3- Polynomial Linear Regression Model

In our learning engine we will implement a parent interface for the previous sub models and we use declaration method to initialize the best fit model after processing the input data as shown in the code below.

```java
PredictingModel forecaster = Predictor.getBestForecast(observedData);
```

Using the factory pattern will enable the analytic engine to choose among the different predictive models. Likewise, will make it a “scalable” to add new patterns in the future

4.2. Best Fit Model Criteria

The choice among the predictive models will be based on how a good fit they are determined by the following calculation criteria
• Akaike Information Criteria (AIC)

• Arithmetic mean of the errors (bias)

• Mean Absolute Deviation (MAD)

• Mean Absolute Percentage Error (MAPE)

• Mean square of the errors (MSE)

• Sum of Absolute Errors (SAE)

The calculation and the implementation of earlier analytical criteria are shown below

```java
protected void calculateAccuracyIndicators( DataSet dataSet )
{
    // Note that the model has been initialized
    initialized = true;

    // Reset various helper summations
    double sumErr = 0.0;
    double sumAbsErr = 0.0;
    double sumAbsPercentErr = 0.0;
    double sumErrSquared = 0.0;

    // Obtain the forecast values for this model
    DataSet forecastValues = new DataSet( dataSet );
    predict( forecastValues );

    // Calculate the Sum of the Absolute Errors
    Iterator<DataPoint> it = dataSet.iterator();
    Iterator<DataPoint> itForecast = forecastValues.iterator();
    while ( it.hasNext() )
    {
        // Get next data point
        DataPoint dp = it.next();
        double x = dp.getDependentValue();

        // Get next forecast value
        DataPoint dpForecast = itForecast.next();
        double forecastValue = dpForecast.getDependentValue();

        // Calculate error in forecast, and update sums appropriately
        double error = forecastValue - x;
        sumErr += error;
        sumAbsErr += Math.abs( error );
        sumAbsPercentErr += Math.abs( error / x );
        sumErrSquared += error*error;
    }

    // Initialize the accuracy indicators
    int n = dataSet.size();
    int p = getNumberOfPredictors();

    accuracyIndicators.setAIC( n*Math.log(2*Math.PI) + Math.log(sumErrSquared/n) + 2 * ( p+2 ) );
    accuracyIndicators.setBias( sumErr / n );
}
```
4.3. Predictive Analytics Models

Once the feeds and the means of integration are established, the predictive analytic model needs to be activated to forecast the future demand. Again, there are many techniques or methods that can be used to establish such model. Many machine learning techniques can be utilized to establish the system demand predictive model such as regression, clustering, classifications, decision trees/rules, k-nearest neighbors…etc [9]. This document will focus only on three solid predictive models. These analytics models are

- Linear Regression Model
- Multiple Linear Regression Model
- Polynomial Linear Regression Model

The detail of each will be discussed in the subsequent sections.

4.3.1. Linear Regression Model

An example of these techniques is a regression predictive model. Linear regression was developed in the field of statistics and is studied as a model for understanding the relationship between input and output numerical variables, and later it has been brought on to be a branch machine learning. The idea of regression model is to find the best-fitting line among historic data points. [10] This will be done through calculating the slope and the intercept as in the below equation.
\[ Y = \text{intercept} + \text{slope} \times X \]

Where \( Y \) in our application would be the CPU utilization, and \( X \) is the number of the user session. \( X \) is the independent variable that will be used to predict the value of \( Y \). The intercept and the slope are calculated as

\[
\text{intercept} = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{(n(\Sigma x^2) - (\Sigma x)^2)}
\]

\[
\text{slope} = \frac{(n(\Sigma xy) - (\Sigma x)(\Sigma y))}{(n(\Sigma x^2) - (\Sigma x)^2)}
\]

We could use other examples of the independent variable such as in time units, such as a day or an hour, average temperature, competitor price and so on, but we will stick to the earlier definition for simplicity. The given representation is a linear equation, making predictions as simple as solving the equation for a specific set of inputs.

The implementation of regression predictive model can be programmed using a Java program (class) that will be part of the Cognitive Performance Application’s analytics engine. The calculation of the linear regression is carried out in the init method as shown in the below code snippet, which is part of the LinearRegressionModel class.

```java
public void init( DataSet dataSet )
{
    int n = dataSet.size();
    double sumX = 0.0;
    double sumY = 0.0;
    double sumXX = 0.0;
    double sumXY = 0.0;

    Iterator<DataPoint> it = dataSet.iterator();
    while ( it.hasNext() )
    {
        DataPoint dp = it.next();
        double x = dp.getIndependentValue( independentVariable );
        double y = dp.getDependentValue();
        sumX += x;
        sumY += y;
        sumXX += x*x;
        sumXY += x*y;
    }
}
```
double xMean = sumX / n;
double yMean = sumY / n;
slope = (n*sumXY - sumX*sumY) / (n*sumXX - sumX*sumX);
intercept = yMean - slope * xMean;
}

Once the model has been established, the calculation for the slope and the intercept has been completed we use the below method to predict the value for CPU utilization of the next future user session.

```java
public double predict( DataPoint dataPoint )
{
    double x = dataPoint.getIndependentValue( independentVariable );
    double forecastValue = intercept + slope*x;
    dataPoint.setDependentValue( forecastValue );
    return forecastValue;
}
```

### 4.3.2. Multiple Linear Regression Model

A multiple variable linear regression model essentially attempts to put a hyperplane through the data points. Mathematically, assuming the independent variables are \( x_i \) and the dependent variable is \( Y \). This hyperplane can be represented as in the below equation:

\[
Y = b_0 + b_1 * x_1 + b_2 * x_2 + b_3 * x_3 + \ldots + b_i * x_i
\]

- Where the \( b_i \) are the coefficients of the regression. \( b_0 \) is called the intercept.

Let each of the \( i \) predictor variables, \( x_1, x_2, \ldots, x_i \) have \( n \) levels. Levels can be expressed in the following way:

\[
y_1 = b_0 + b_1 * x_{11} + b_2 * x_{12} + b_3 * x_{13} + \ldots + b_i * x_{1i}
\]
\[
y_2 = b_0 + b_1 * x_{21} + b_2 * x_{22} + b_3 * x_{23} + \ldots + b_i * x_{2i}
\]
\[
\ldots
\]
\[
y_n = b_0 + b_1 * x_{n1} + b_2 * x_{n2} + b_3 * x_{n3} + \ldots + b_i * x_{ni}
\]
The system of \( n \) equations shown previously can be represented in matrix notation as follows:

\[
Y = Xb
\]

For the special case of one variable the above matrix can be expressed as below:

\[
\begin{bmatrix}
n & \Sigma x \\
\Sigma x & \Sigma x^2
\end{bmatrix}
\begin{bmatrix}
b_0 \\
b_1
\end{bmatrix}
= 
\begin{bmatrix}
\Sigma y
\end{bmatrix}
\]

For two variables the above matrix can be expressed as below:

\[
\begin{bmatrix}
n & \Sigma x_1 & \Sigma x_2 \\
\Sigma x_1 & \Sigma x_1^2 & \Sigma x_1x_2 \\
\Sigma x_2 & \Sigma x_1x_2 & \Sigma x_2^2
\end{bmatrix}
\begin{bmatrix}
b_0 \\
b_1 \\
b_2
\end{bmatrix}
= 
\begin{bmatrix}
\Sigma y \\
\Sigma x_1y \\
\Sigma x_2y
\end{bmatrix}
\]

and so on. The previous illustrated calculation for multi regression model is carried out in the init method as shown in the below code snippet, which is part of the MultipleLinearRegressionModel class.

```java
public void init( DataSet dataSet ){
    String varNames[] = dataSet.getIndependentVariables();
    // If no coefficients have been defined for this model,
    // use all that exist in this data set
    if ( coefficient == null )
        setIndependentVariables( varNames );

    int n = varNames.length;
    double a[][] = new double[n+1][n+2];

    // Iterate through dataSet
    Iterator<DataPoint> it = dataSet.iterator();
    while ( it.hasNext() )
    {
        // Get next data point
        DataPoint dp = it.next();

        // For each row in the matrix, a
        for ( int row=0; row<n+1; row++ )
        {
            double rowMult = 1.0;
            if ( row != 0 )
            {
                // Get value of independent variable, row
                String rowVarName = varNames[row-1];
                rowMult = dp.getIndependentValue(rowVarName);
            }

            // For each column in the matrix, a
            for ( int col=0; col<n+2; col++ )
            {
                if ( col != n+1 )
```
Once the multi regression problem is expressed in a matrix form we can implement a Gaussian elimination on the given matrix to find the values of the coefficients $b_0$, $b_1$, $b_2$ … $b_n$, as shown in the code snippet below, which is part of a common utility class.

```java
static double[] GaussElimination( int n, double a[][]) {
    // Forward elimination
    for ( int k=0; k<n-1; k++ ) {
        for ( int i=k+1; i<n; i++ ) {
            double qt = a[i][k] / a[k][k];
            for ( int j=k+1; j<n+1; j++ )
                a[i][j] -= qt * a[k][j];
            a[i][k] = 0.0;
        }
    }

    double x[] = new double[n];
    // Back-substitution
    x[n-1] = a[n-1][n] / a[n-1][n-1];
    for ( int k=n-2; k>=0; k-- ) {
        double sum = 0.0;
        for ( int j=k+1; j<n+1; j++ )
            sum += a[k][j]*x[j];
        x[k] = ( a[k][n] - sum ) / a[k][k];
    }
    return x;
}
```
Once the model has been established, or that the y-intercept and the other coefficients have been calculated, we can use the below method to predict the value for CPU utilization, or the independent value, of the next future iteration.

```java
public double predict(DataPoint dataPoint) {
    double forecastValue = intercept;
    Iterator<Map.Entry<String, Double>> it = coefficient.entrySet().iterator();
    while (it.hasNext()) {
        Map.Entry<String, Double> entry = it.next();
        // Get value of independent variable
        double x = dataPoint.getIndependentValue((String)entry.getKey());
        // Get coefficient for this variable
        double coeff = ((Double)entry.getValue()).doubleValue();
        forecastValue += coeff * x;
    }
    dataPoint.setDependentValue(forecastValue);
    return forecastValue;
}
```

### 4.3.3. Polynomial Regression Model

When the relation between the independent variable \(x\) and the dependent variable \(y\) is not linear but rather show as “curve” we can use a single variable polynomial regression model, which essentially attempts to put a polynomial line through the data points. Mathematically, assuming the independent variable is \(x\) and the dependent variable is \(y\), then this line can be represented as:

\[
y = b_0 + b_1 x^1 + b_2 x^2 + b_3 x^3 + \ldots + b_m x^m
\]

*Where “\(m\)” is order of the polynomial equation. [11]*
Without getting into much details of how to drive the mathematical equation for the polynomial regression, which is beyond the scope of this document, the problem can represent itself in a matrix form as shown below.

\[
\begin{bmatrix}
  n & \Sigma x_i & \ldots & \Sigma x_i^m \\
  \Sigma x_1 & \Sigma x_1^2 & \ldots & \Sigma x_1^{m+1} \\
  \ldots & \ldots & \ldots & \ldots \\
  \Sigma x_i^m & \Sigma x_i^{m+1} & \ldots & \Sigma x_i^{2m} \\
\end{bmatrix}
\begin{bmatrix}
  b_0 \\
  b_1 \\
  \ldots \\
  b_m \\
\end{bmatrix}
= 
\begin{bmatrix}
  \Sigma y_i \\
  \Sigma x_i y_i \\
  \ldots \\
  \Sigma x_i^m y_i \\
\end{bmatrix}
\]

This matrix presentation is carried out in the init method as shown in the below code snippet, which is part of the PolynomialRegressionModel class.

```java
public void init( DataSet dataSet )
{
    double a[][] = new double[order][order+1];
    for ( int i=0; i<order; i++ ) {
        for ( int j=0; j<order; j++ ) {
            int k = i + j;
            Iterator<DataPoint> it = dataSet.iterator();
            while ( it.hasNext() )
            {
                DataPoint dp = it.next();
                double x = dp.getIndependentValue( independentVariable );
                a[i][j] = a[i][j] + Math.pow(x,k);
            }
        }
    }
    Iterator<DataPoint> it = dataSet.iterator();
    while ( it.hasNext() )
    {
        DataPoint dp = it.next();
        double x = dp.getIndependentValue( independentVariable );
        double y = dp.getDependentValue();
        a[i][order] += y*Math.pow(x,i);
    }
    coefficient = Utils.GaussElimination( order, a );
    // Calculate the accuracy indicators
    calculateAccuracyIndicators( dataSet );
}
```
Once the single polynomial regression problem is expressed in a matrix form we can implement a Gaussian elimination on the given matrix to find the values of the coefficients \(a_0, a_1, a_2 \ldots a_n\), as shown in the previous section. And once the polynomial regression model calculation has been completed, we can use the below method to predict the value for CPU utilization, or the independent value, of the next future iteration.

```java
public double predict( DataPoint dataPoint ) {
    double x = dataPoint.getIndependentValue( independentVariable );
    double forecastValue = 0.0;
    for ( int i=0; i<order; i++ )
        forecastValue += coefficient[i] * Math.pow(x,i);
    dataPoint.setDependentValue( forecastValue );
    return forecastValue;
}
```

### 4.3.4. Models Comparison

Once the above models are established and calculated, we use an evaluation criteria class to specify how to compare the different models based on the values of Akaike Information Criteria (AIC), Arithmetic mean of the errors (bias), Mean Absolute Deviation (MAD), Mean Absolute Percentage Error (MAPE), Mean square of the errors (MSE), Sum of Absolute Errors (SAE.) we can also use them all in our comparison as shown in the code below which is part of the Predictor class.

```java
static boolean compare( PredictingModel model1,
                        PredictingModel model2,
                        EvaluationCriteria evalMethod )
{
    // Special case. Any model is better than no model!
    if ( model2 == null )
        return true;
    double tolerance = 0.00000001;
    // Use evaluation method as requested by user
    if ( evalMethod == EvaluationCriteria.BIAS )
        return ( model1.getBias() <= model2.getBias() );
    else if ( evalMethod == EvaluationCriteria.MAD )
        return ( model1.getMAD() <= model2.getMAD() );
    else if ( evalMethod == EvaluationCriteria.MAPE )
        return ( model1.getMAPE() <= model2.getMAPE() );
    else
        return false;
}
```
```java
else if ( evalMethod == EvaluationCriteria.MSE )
    return ( model1.getMSE() <= model2.getMSE() );
else if ( evalMethod == EvaluationCriteria.SAE )
    return ( model1.getSAE() <= model2.getSAE() );
else if ( evalMethod == EvaluationCriteria.AIC )
    return ( model1.getAIC() <= model2.getAIC() );

// Default evaluation method is a combination
int score = 0;
if ( model1.getAIC()-model2.getAIC() <= tolerance )
    score++;
else if ( model1.getAIC()-model2.getAIC() >= tolerance )
    score--;
if ( model1.getBias()-model2.getBias() <= tolerance )
    score++;
else if ( model1.getBias()-model2.getBias() >= tolerance )
    score--;
if ( model1.getMAD()-model2.getMAD() <= tolerance )
    score++;
else if ( model1.getMAD()-model2.getMAD() >= tolerance )
    score--;
if ( model1.getMAPE()-model2.getMAPE() <= tolerance )
    score++;
else if ( model1.getMAPE()-model2.getMAPE() >= tolerance )
    score--;
if ( model1.getMSE()-model2.getMSE() <= tolerance )
    score++;
else if ( model1.getMSE()-model2.getMSE() >= tolerance )
    score--;
if ( model1.getSAE()-model2.getSAE() <= tolerance )
    score++;
else if ( model1.getSAE()-model2.getSAE() >= tolerance )
    score--;
    return ( score > 0 );
}

4.4. Run Example

The below example will show how the analytic system predict the value of the CPU utilization for the next User session based on the CPU utilization values recorded for the immediate preceding five user sessions as shown in the below table.

<table>
<thead>
<tr>
<th>User Session #</th>
<th>CPU Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>51.6504854368932</td>
</tr>
<tr>
<td>1</td>
<td>24.27184460194176</td>
</tr>
<tr>
<td>2</td>
<td>54.16666666666664</td>
</tr>
<tr>
<td>3</td>
<td>92.63657957244655</td>
</tr>
<tr>
<td>4</td>
<td>78.33622183708839</td>
</tr>
<tr>
<td>5</td>
<td>89.0</td>
</tr>
</tbody>
</table>

*Table 9 – CPU Per User Session*
The analytical engine decided that the best model is the Analytics Engine is linear regression model with the following equation:

\[ y = 35.86511809212924 + 12.173620771264279 \times x \]

, and the predicted value of the CPU utilization for the next run will be predicted value:

96.73322194845063

as shown in the system logs captured below.

```
Data set: 
(x=0.0, dependentValue=51.6504854368932) 
(x=1.0, dependentValue=24.271844660194176) 
(x=2.0, dependentValue=54.166666666666664) 
(x=3.0, dependentValue=92.63657957244655) 
(x=4.0, dependentValue=78.33622183708839) 

bestModel: linear regression model with the following equation:
  y = 35.86511809212924 + 12.173620771264279 * x

count---5
predicted value: 96.73322194845063
```
4.5. Analytics Engine Class Diagram

The below graph a UML class diagram summarizing the structure of the learning engine.

Figure 20 – ANALYTICS ENGINE UML CLASS DIAGRAM
4.6. Summary

The Analytics Engine is a component of the Cognitive Performance Application that establishes a pattern between library of factors and the system performance gauges to act as an early alert system signaling a potential deadlock. In this chapter we explained the architecture decision of using a factory design pattern to make the model scalable for future updates. We also have shown the three different models that constitute the core of our engine, namely

- Linear Regression Model
- Multiple Linear Regression Model
- Polynomial Linear Regression Model

Moreover, we have explained how the system will compare and choose between these model based on the accuracy of the their calculation using the following criteria

- Akaike Information Criteria (AIC)
- Arithmetic mean of the errors (bias)
- Mean Absolute Deviation (MAD)
- Mean Absolute Percentage Error (MAPE)
- Mean square of the errors (MSE)
- Sum of Absolute Errors (SAE)
The Queuing engine is the last component of the Cognitive Performance Application. The engine will have two execution modes.

- A pass-through
- Queuing mode.

The learning engine or the analytics engine will signal the mode of execution for the queuing engine. When the pass-through signal is flagged, all database transactions will go uninterrupted from the JVM to the DBMS. When the queuing mode is signaled, the database queries will join a First-Come-First-Serviced (FCFS) queue structure. The database queries in the queue structure will be concurrently executed after adding a trifling delay to each in order to stream down the flow and reduce the performance load on the JVM and DBMS. The queuing engine will iterate between the two execution modes based on the signal given.
We will use native library called `BlockingQueue.Java.util.concurrent.BlockingQueue`, which is a Java Queue that supports operations that wait for the queue to become non-empty when retrieving and removing an element, and wait for space to become available in the queue when adding an element. Java BlockingQueue doesn’t accept null values and throw `NullPointerException` when storing null value in the queue. Java BlockingQueue implementations are thread-safe. All queuing methods are atomic in nature and use internal locks or other forms of concurrency control. Java BlockingQueue interface is part of Java collections framework and it’s primarily used for implementing producer consumer problem. We don’t need to worry about waiting for the space to be available for producer or object to be available for consumer in BlockingQueue because it is handled by implementation classes of BlockingQueue. Java provides several BlockingQueue implementations such as `ArrayBlockingQueue`, `LinkedBlockingQueue`, `PriorityBlockingQueue`, `SynchronousQueue` etc. While implementing producer consumer problem in BlockingQueue, we will use `LinkedBlockingQueue` implementation.
package cpu.cs.louisville.edu;

import Java.sql.Connection;
import Java.sql.Statement;
import Java.text.NumberFormat;
import Java.util.Random;
import Java.util.concurrent.BlockingQueue;
import Java.util.concurrent.LinkedBlockingDeque;

public class UserSession implements Runnable {
    static Connection connect = null;
    static Statement statement = null;
    private int threadNumber;
    private boolean runTimeMode = true;
    private int counterRunRound;

    public UserSession(int threadNum, boolean runTime, int counterRunNum) {
        this.threadNumber = threadNum;
        this.runTimeMode = runTime;
        this.counterRunRound = counterRunNum;
    }

    NumberFormat format = NumberFormat.getInstance();

    static long maxMemory;
    static long allocatedMemory;
    static long heapRatio;

    private BlockingQueue<SessionQueries> concurrentLinkedQueue = new LinkedBlockingDeque<SessionQueries>(100);

    public void dequeueItem() {
        if (!concurrentLinkedQueue.isEmpty()) {
            // System.out.println("Queue size: " + concurrentLinkedQueue.size());
            try {
                concurrentLinkedQueue.take().runQueries(connect, statement, format, maxMemory, allocatedMemory, heapRatio, this.threadNumber, this.counterRunRound);
            } catch (Exception e) {} 
        } else {
            System.out.println("Queue Empty ");
        }
    }

    private void enqueueItem(SessionQueries item) throws Exception {
        // System.out.println("Enqueueing item ");
        concurrentLinkedQueue.put(item);
    }

    public int getQueueSize() {
        if (!concurrentLinkedQueue.isEmpty()) {
            return concurrentLinkedQueue.size();
        } else {
            return 0;
        }
    }

    /**
     *
     */
    public void run() {
        SessionQueries session = new SessionQueries();

        int numberOfUserSession = (new Random().nextInt(100));
        System.out.println("number Of User Session "+numberOfUserSession);
        for (int count = 0; count < numberOfUserSession; count++){
This queue structure will be used to rectify the system deadlock situation when the number of queries overwhelms the system and cause the CPU utilization to reach its peak.
RANDOM LOAD SIMULATION

In this section, a system run is described for a certain amount of duration, five minutes to be exact. During the five minutes range, the simulator program triggers a random number of threads, user sessions, at constant rate, 12 seconds to be exact. Also for this run the Java heap size is being reduced to help the system reach a bottleneck situation and hence study the recovery mechanism.

6.1. Discrete vs. Continuous

The simulator only triggers random number of threads, sampling user loads, every twelfth seconds during a five-minute range and hence, by definition, is classified as a discrete process. The choice of using a discrete run was to assemble the actual user load on an online system. Otherwise, the simulator would have triggered the number of the users in a loop structure such as the Java language “for” or “while” loops continuously for the same time range, or five minutes as in our case.

6.2. Pseudo-random Number Sampling

The simulator triggers random number of threads, every twelfth seconds during a five-minute range. The random number of threads is generated using a Java Pseudo-random generator utility. The Java number sampling or non-uniform pseudo-random variate generation is the numerical method of generating random numbers distributed according to a normal distribution of certain mean and standard deviation.
6.2.1. Normalization

In order to get to mean and the average we have studied the user load on the hp ecommerce store for four consecutive working weeks as shown below. We then normalized the data to make it is values between 0 and 1 using the following equation:

\[ X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \]

Then we calculated the mean and standard deviation as shown in the table below.

<table>
<thead>
<tr>
<th>Day</th>
<th>#</th>
<th>Number of users</th>
<th>Normalized</th>
</tr>
</thead>
<tbody>
<tr>
<td>18-Sep</td>
<td>1</td>
<td>231</td>
<td>0.401993355</td>
</tr>
<tr>
<td>19-Sep</td>
<td>2</td>
<td>197</td>
<td>0.289036545</td>
</tr>
<tr>
<td>20-Sep</td>
<td>3</td>
<td>231</td>
<td>0.401993355</td>
</tr>
<tr>
<td>21-Sep</td>
<td>4</td>
<td>131</td>
<td>0.069767442</td>
</tr>
<tr>
<td>22-Sep</td>
<td>5</td>
<td>286</td>
<td>0.584717608</td>
</tr>
<tr>
<td>25-Sep</td>
<td>6</td>
<td>374</td>
<td>0.877076412</td>
</tr>
<tr>
<td>26-Sep</td>
<td>7</td>
<td>110</td>
<td>0</td>
</tr>
<tr>
<td>27-Sep</td>
<td>8</td>
<td>282</td>
<td>0.571428571</td>
</tr>
<tr>
<td>28-Sep</td>
<td>9</td>
<td>229</td>
<td>0.395348837</td>
</tr>
<tr>
<td>2-Oct</td>
<td>10</td>
<td>411</td>
<td>1</td>
</tr>
<tr>
<td>3-Oct</td>
<td>11</td>
<td>220</td>
<td>0.365448505</td>
</tr>
<tr>
<td>4-Oct</td>
<td>12</td>
<td>411</td>
<td>1</td>
</tr>
<tr>
<td>5-Oct</td>
<td>13</td>
<td>333</td>
<td>0.740863787</td>
</tr>
<tr>
<td>6-Oct</td>
<td>14</td>
<td>284</td>
<td>0.57807309</td>
</tr>
<tr>
<td>8-Oct</td>
<td>15</td>
<td>192</td>
<td>0.272425249</td>
</tr>
<tr>
<td>9-Oct</td>
<td>16</td>
<td>218</td>
<td>0.358803987</td>
</tr>
<tr>
<td>10-Oct</td>
<td>17</td>
<td>242</td>
<td>0.438538206</td>
</tr>
<tr>
<td>11-Oct</td>
<td>18</td>
<td>269</td>
<td>0.528239203</td>
</tr>
<tr>
<td>12-Oct</td>
<td>19</td>
<td>188</td>
<td>0.259136213</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td>254.6842105</td>
<td>0.48067844</td>
</tr>
<tr>
<td>Standard deviation</td>
<td></td>
<td>83.15183744</td>
<td>0.276251952</td>
</tr>
</tbody>
</table>
The Java number sampling or non-uniform pseudo-random variate generation is the numerical method of generating random numbers distributed according to a normal distribution of mean 0.48067844 and standard deviation 0.276251952, derived from the above calculation. The choice of normal distribution is more appropriate because if the sample size is sufficiently large, as in our case, the sampling distribution of the sample mean approximates the normal distribution. This holds true despite the distribution of the population from which the sample were collected according to the central limit theorem.

6.3. Results of First Run

The first run does not entail any performance enhancement mechanism. The first run is aimed to observe and collect the as-is system performance metrics to be compared later on after the addition of the learning, analytics, and queue components. The first run resulted in the execution of a total of 1004 threads or user sessions. The first run is triggered over a five minutes time range, within which a random number of user sessions is rendered every 12 seconds using Java random number generator and normally distributed. Performance metrics such as the

- CPU Utilization percentage,
- Thread response time in seconds,
- Allocated memory in KB
- Heap ratio
are collected and recorded toward the end of each thread execution. A summary of the run data is shown in the below table and the further subsequent sections.

<table>
<thead>
<tr>
<th>CPU Usage</th>
<th>Response Time (s)</th>
<th>Start Time</th>
<th>End Time</th>
<th>Thread Number</th>
<th>Heap Ratio</th>
<th>Allocated Memory (KB)</th>
<th>Max Memory (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.00%</td>
<td>6.82298798</td>
<td>2051670.832</td>
<td>2051677.655</td>
<td>1</td>
<td>0.03</td>
<td>6.17</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.10095347</td>
<td>2051677.677</td>
<td>2051679.778</td>
<td>2</td>
<td>0.08</td>
<td>15.473</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.75671768</td>
<td>2051679.779</td>
<td>2051682.536</td>
<td>3</td>
<td>0.11</td>
<td>20.408</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>3.88841427</td>
<td>2051682.537</td>
<td>2051686.422</td>
<td>4</td>
<td>0.09</td>
<td>16.331</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>3.52589645</td>
<td>2051691.545</td>
<td>2051695.07</td>
<td>5</td>
<td>0.09</td>
<td>16.496</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.88591846</td>
<td>2051695.072</td>
<td>2051697.958</td>
<td>6</td>
<td>0.10</td>
<td>17.844</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.55195492</td>
<td>2051697.959</td>
<td>2051700.511</td>
<td>7</td>
<td>0.09</td>
<td>16.577</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>3.0836706</td>
<td>2051700.512</td>
<td>2051703.596</td>
<td>8</td>
<td>0.08</td>
<td>15.64</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.90243103</td>
<td>2051708.703</td>
<td>2051711.605</td>
<td>9</td>
<td>0.07</td>
<td>12.811</td>
<td>178.176</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.45123124</td>
<td>2054600.353</td>
<td>2054602.804</td>
<td>996</td>
<td>0.64</td>
<td>115.208</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>14.5856296</td>
<td>2054600.353</td>
<td>2054604.648</td>
<td>997</td>
<td>0.79</td>
<td>141.898</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>32.38739141</td>
<td>2054601.418</td>
<td>2054623.806</td>
<td>998</td>
<td>0.63</td>
<td>112.641</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>8.17268885</td>
<td>2054613.735</td>
<td>2054619.908</td>
<td>999</td>
<td>0.48</td>
<td>85.554</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>12.50983365</td>
<td>2054608.014</td>
<td>2054620.523</td>
<td>1000</td>
<td>0.58</td>
<td>103.689</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>11.90430417</td>
<td>2054607.993</td>
<td>2054619.898</td>
<td>1001</td>
<td>0.53</td>
<td>95.513</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>14.20815774</td>
<td>2054605.491</td>
<td>2054619.699</td>
<td>1002</td>
<td>0.52</td>
<td>93.001</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>11.01713206</td>
<td>2054608.606</td>
<td>2054619.023</td>
<td>1003</td>
<td>0.47</td>
<td>83.931</td>
<td>178.176</td>
</tr>
<tr>
<td>54.47%</td>
<td>3.48059529</td>
<td>2054627.697</td>
<td>2054631.178</td>
<td>1004</td>
<td>0.63</td>
<td>112.828</td>
<td>178.176</td>
</tr>
</tbody>
</table>

Table 11 – First Run Summary

6.3.1. CPU Utilization

The below graph and table shows the system CPU utilization recorded toward the end of the execution of each thread, or a user session, along with a data statics summary including the average, max, min, and standard deviation.
Figure 22 – System CPU Utilization Charts

<table>
<thead>
<tr>
<th>STATS Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>94.15%</td>
</tr>
<tr>
<td>Max</td>
<td>100.00%</td>
</tr>
<tr>
<td>Min</td>
<td>3.01%</td>
</tr>
<tr>
<td>StdDev</td>
<td>0.158067891</td>
</tr>
</tbody>
</table>

Table 12 – First Run CPU data Summary

6.3.2. Response Time Per Thread

The below graph records each thread response time calculated the difference between the thread start time and the thread end time. The response time of each query will be measured using the nanoTime() method to allow for max precision. This method provides nanosecond precision, but not necessarily nanosecond accuracy. For example, to measure how long some database transaction takes to execute:

```java
long startTime = System.nanoTime();  // ... the code being measured ...
long estimatedTime = System.nanoTime() - startTime;
```

The subsequent table displays a data statistics summary including the average, max, min, and standard deviation for all the user sessions involved in the run.
**Figure 23 – Response Time in Seconds per Thread**

<table>
<thead>
<tr>
<th>STATS Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>8.09</td>
</tr>
<tr>
<td>Max</td>
<td>43.63</td>
</tr>
<tr>
<td>Min</td>
<td>1.33</td>
</tr>
<tr>
<td>StdDev</td>
<td>6.35</td>
</tr>
</tbody>
</table>

*Table 13 – First run thread response time summary*

### 6.3.3. Heap Ratio

The below graph shows the JVM heap ratio recorded toward at the end of the execution of each thread, or a user session, by retrieving the JVM allocated memory divided my max memory variable. The subsequent table displays the heap ratio data statistics summary including first Quartile, second Quartile (median), and third Quartile.
Figure 24 – HEAP RATIO PER THREAD

<table>
<thead>
<tr>
<th>STATS Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Quartile</td>
<td>43</td>
</tr>
<tr>
<td>Second Quartile (median)</td>
<td>55</td>
</tr>
<tr>
<td>Third Quartile</td>
<td>63.75</td>
</tr>
</tbody>
</table>

Table 14 – First run heap ratio summary

6.4. Results of Second Run

The second run resulted the execution 940 threads or user session. The second run is similar to the first one in that it is also triggered over exactly five minutes time range, within which a random number of user sessions is rendered every 12 seconds using Java random number generator. The main difference from the first run is that it includes two extra major components to enhance the performance. These are the learning engine and the queuing engine. These two components are responsible mainly for defining and detecting the system bottleneck and introducing a recovering mechanism. The system resorts to the recovering mechanism if and only a bottleneck is detected and then alters back to normal execution mode once the bottleneck conditions are void. The details of these two extra
components will be covered on different chapters of this document. A summary of the run data is shown in the below table.

<table>
<thead>
<tr>
<th>CPU Usage</th>
<th>Response Time (s)</th>
<th>Start Time</th>
<th>End Time</th>
<th>Thread Number</th>
<th>Heap Ratio</th>
<th>Allocated Memory (KB)</th>
<th>Max Memory (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>71.45%</td>
<td>6.36287852</td>
<td>2068323.678</td>
<td>206830.041</td>
<td>1</td>
<td>4</td>
<td>8.099</td>
<td>178.176</td>
</tr>
<tr>
<td>44.42%</td>
<td>3.37656177</td>
<td>2068340.181</td>
<td>2068343.557</td>
<td>2</td>
<td>4</td>
<td>7.989</td>
<td>178.176</td>
</tr>
<tr>
<td>37.73%</td>
<td>3.18759926</td>
<td>2068348.452</td>
<td>2068351.64</td>
<td>3</td>
<td>5</td>
<td>9.113</td>
<td>178.176</td>
</tr>
<tr>
<td>41.13%</td>
<td>2.90389656</td>
<td>2068362.034</td>
<td>2068364.938</td>
<td>4</td>
<td>6</td>
<td>11.366</td>
<td>178.176</td>
</tr>
<tr>
<td>80.76%</td>
<td>4.02605219</td>
<td>2068369.837</td>
<td>2068373.863</td>
<td>5</td>
<td>8</td>
<td>14.83</td>
<td>178.176</td>
</tr>
<tr>
<td>42.29%</td>
<td>3.09899373</td>
<td>2068378.888</td>
<td>2068381.987</td>
<td>6</td>
<td>5</td>
<td>10.162</td>
<td>178.176</td>
</tr>
<tr>
<td>52.84%</td>
<td>3.21760867</td>
<td>2068386.854</td>
<td>2068390.072</td>
<td>7</td>
<td>11</td>
<td>20.212</td>
<td>178.176</td>
</tr>
<tr>
<td>31.19%</td>
<td>2.3927622</td>
<td>2068400.266</td>
<td>2068402.659</td>
<td>8</td>
<td>10</td>
<td>18.542</td>
<td>178.176</td>
</tr>
<tr>
<td>71.45%</td>
<td>6.36287852</td>
<td>2068323.678</td>
<td>2068330.041</td>
<td>1</td>
<td>4</td>
<td>8.099</td>
<td>178.176</td>
</tr>
</tbody>
</table>

...  

| 100.00%   | 9.21452238        | 2071222.498  | 2071231.713 | 931          | 58         | 101.22               | 178.176        |
| 100.00%   | 14.88215555       | 2071222.504  | 2071237.486 | 932          | 49         | 88.945               | 178.176        |
| 100.00%   | 16.7946994        | 2071222.509  | 2071239.303 | 933          | 63         | 113.543              | 178.176        |
| 0.00%     | 5.0810955         | 2071244.65   | 2071249.731 | 934          | 75         | 134.55               | 178.176        |
| 100.00%   | 39.96265167       | 2071222.531  | 2071262.494 | 935          | 61         | 109.058              | 178.176        |
| 100.00%   | 37.2201013        | 2071222.525  | 2071259.745 | 936          | 61         | 109.112              | 178.176        |
| 100.00%   | 34.5797734        | 2071222.55   | 2071257.1   | 937          | 40         | 71.511               | 178.176        |
| 100.00%   | 32.35881489       | 2071222.515  | 2071254.75  | 938          | 46         | 82.62                | 178.176        |
| 100.00%   | 6.77268899        | 2071244.659  | 2071251.432 | 939          | 88         | 158.063              | 178.176        |

**Table 15 –Second Run Summary**

The subsequent sections show more detailed analysis on the performance metrics collected through the run.
6.4.1. **CPU Utilization**

The below graph and table shows the system CPU utilization recorded toward at the end of the execution of each thread, or a user session, along with a data statics summary including the average, max, min, and standard deviation.

![System CPU Utilization Chart](image)

**Figure 25 – System CPU Utilization Charts**

<table>
<thead>
<tr>
<th>STAT Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>87.80%</td>
</tr>
<tr>
<td>Max</td>
<td>100.00%</td>
</tr>
<tr>
<td>Min</td>
<td>0.00%</td>
</tr>
<tr>
<td>StdDev</td>
<td>0.268636399</td>
</tr>
</tbody>
</table>

*Table 16 – Second run CPU data summary*

The second run shows significant improvement in the overall CPU utilization. Also it shows that the added components managed to decrease the frequency of which the system hits the bottleneck. These results are visualized and calculated in the above graph and table.
6.4.2. Response Time Per Thread

The below graph records each thread response time calculated by recording the thread start time and end time then retrieving the difference as the thread response time. The subsequent table displays a data statics summary including the average, max, min, and standard deviation for all the user sessions involved in the run.

![Response Time/Thread](image)

**Table 17 – Second run thread response time data summary**

<table>
<thead>
<tr>
<th>STATS Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>9.97</td>
</tr>
<tr>
<td>Max</td>
<td>48.09</td>
</tr>
<tr>
<td>Min</td>
<td>1.40</td>
</tr>
<tr>
<td>StdDev</td>
<td>7.58</td>
</tr>
</tbody>
</table>

The second run shows slight change in the overall response time. However, the thread response time is shown to be a little longer due to the introductory of the queuing component as a recovery mechanism. The stacking of the threads to the queuing engine, which will be detailed in subsequent sections, during the system peak times, contributed
to the increase of some of the individual thread’s response time, and hence increasing the overall response times of the user sessions as recorded in the above table.

6.4.3. Heap Ratio Per Thread

The below graph shows the JVM heap ratio recorded toward at the end of the execution of each thread, or a user session, by retrieving the JVM allocated memory divided my max memory variable. The subsequent table displays the heap ratio data statics summary including the first Quartile, second Quartile (median), and third Quartile.

---

**Figure 27 – Heap Ratio Per Thread**

<table>
<thead>
<tr>
<th>STATS Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Quartile</td>
<td>45</td>
</tr>
<tr>
<td>Second Quartile (median)</td>
<td>58</td>
</tr>
<tr>
<td>Third Quartile</td>
<td>68</td>
</tr>
</tbody>
</table>

*Table 18 – Second run heap ratio data summary*

The heap ratio also shows to be similar to the first run with a slight or insignificant increase resulted from the addition of the recovery components.
### 6.5. Results of Third Run

The third run resulted in the execution of a total of 905 threads, user sessions. The second run is similar to the first one in that it is also triggered over a five minutes time range, within which a random number of user sessions is rendered every 12 seconds using Java random number generator. The main difference from the first and second runs is that it includes, along with the learning and the queuing components, a third component. This new component is the analytic engine, which is mainly responsible for predicting the possibility of the system reaching a bottleneck based on the historic runs. The details of all these components; learning, queuing and analytics, will be covered in subsequent sections.

A summary of the run data is shown in the below table.

<table>
<thead>
<tr>
<th>CPU Usage</th>
<th>Response Time (s)</th>
<th>Start Time</th>
<th>End Time</th>
<th>Thread Number</th>
<th>Heap Ratio</th>
<th>Allocated Memory (KB)</th>
<th>Max Memory (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.64%</td>
<td>5.6798846</td>
<td>2082177.806</td>
<td>2082183.486</td>
<td>1</td>
<td>2</td>
<td>5.301</td>
<td>178.176</td>
</tr>
<tr>
<td>40.05%</td>
<td>3.51932159</td>
<td>2082198.698</td>
<td>2082202.217</td>
<td>2</td>
<td>8</td>
<td>14.28</td>
<td>178.176</td>
</tr>
<tr>
<td>50.00%</td>
<td>3.64850281</td>
<td>2082212.432</td>
<td>2082216.081</td>
<td>3</td>
<td>9</td>
<td>17.44</td>
<td>178.176</td>
</tr>
<tr>
<td>53.82%</td>
<td>3.09170858</td>
<td>2082221.019</td>
<td>2082224.111</td>
<td>4</td>
<td>13</td>
<td>23.777</td>
<td>178.176</td>
</tr>
<tr>
<td>39.59%</td>
<td>3.13410433</td>
<td>2082229.13</td>
<td>2082232.265</td>
<td>5</td>
<td>12</td>
<td>21.952</td>
<td>178.176</td>
</tr>
<tr>
<td>38.55%</td>
<td>3.14383253</td>
<td>2082237.258</td>
<td>2082240.402</td>
<td>6</td>
<td>8</td>
<td>15.233</td>
<td>178.176</td>
</tr>
<tr>
<td>36.50%</td>
<td>3.01136807</td>
<td>2082250.672</td>
<td>2082253.684</td>
<td>7</td>
<td>8</td>
<td>15.481</td>
<td>178.176</td>
</tr>
<tr>
<td>61.89%</td>
<td>3.59677626</td>
<td>2082258.63</td>
<td>2082262.227</td>
<td>8</td>
<td>6</td>
<td>11.363</td>
<td>178.176</td>
</tr>
<tr>
<td>42.49%</td>
<td>2.19742104</td>
<td>2082267.364</td>
<td>2082269.561</td>
<td>9</td>
<td>14</td>
<td>25.196</td>
<td>178.176</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>96.80%</td>
<td>9.71586053</td>
<td>2085075.284</td>
<td>2085085</td>
<td>896</td>
<td>62</td>
<td>111.818</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.90107402</td>
<td>2085089.062</td>
<td>2085091.963</td>
<td>897</td>
<td>76</td>
<td>136.73</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>7.89125783</td>
<td>2085089.532</td>
<td>2085097.423</td>
<td>898</td>
<td>43</td>
<td>76.935</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>2.7990394</td>
<td>2085100.152</td>
<td>2085102.951</td>
<td>899</td>
<td>57</td>
<td>101.806</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>7.5139068</td>
<td>2085091.722</td>
<td>2085099.236</td>
<td>900</td>
<td>56</td>
<td>101.097</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>17.28572346</td>
<td>2085097.036</td>
<td>2085114.322</td>
<td>901</td>
<td>69</td>
<td>123.635</td>
<td>178.176</td>
</tr>
<tr>
<td>100.00%</td>
<td>15.42726171</td>
<td>2085097.027</td>
<td>2085112.455</td>
<td>902</td>
<td>78</td>
<td>140.132</td>
<td>178.176</td>
</tr>
</tbody>
</table>
The subsequent sections show more detailed analysis on the performance metrics collected through the run.

### 6.5.1. CPU Utilization

The below graph and table shows the system CPU utilization recorded toward at the end of the execution of each thread, or a user session, along with a data statics summary including the average, max, min, and standard deviation.

<table>
<thead>
<tr>
<th>Thread #</th>
<th>CPU Utilization %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>52</td>
</tr>
<tr>
<td>103</td>
<td>103</td>
</tr>
<tr>
<td>154</td>
<td>154</td>
</tr>
<tr>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>307</td>
<td>307</td>
</tr>
<tr>
<td>358</td>
<td>358</td>
</tr>
<tr>
<td>400</td>
<td>400</td>
</tr>
<tr>
<td>451</td>
<td>451</td>
</tr>
<tr>
<td>502</td>
<td>502</td>
</tr>
<tr>
<td>553</td>
<td>553</td>
</tr>
<tr>
<td>604</td>
<td>604</td>
</tr>
<tr>
<td>655</td>
<td>655</td>
</tr>
<tr>
<td>706</td>
<td>706</td>
</tr>
<tr>
<td>757</td>
<td>757</td>
</tr>
<tr>
<td>808</td>
<td>808</td>
</tr>
</tbody>
</table>

**System CPU Utilization**

![System CPU Utilization Chart](image)

<table>
<thead>
<tr>
<th>Stats Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>84.60%</td>
</tr>
<tr>
<td>Max</td>
<td>100.00%</td>
</tr>
<tr>
<td>Min</td>
<td>0.00%</td>
</tr>
<tr>
<td>StdDev</td>
<td>0.293183864</td>
</tr>
</tbody>
</table>

**Table 19 – Third run data summary**

**Table 20 – Third run CPU utilization data summary**
The second run shows meaningful another improvement in the overall CPU utilization. Also, it shows that the added component managed to further decrease the frequency of which the system hits the bottleneck. These results can be inspected visually or mathematically from the above graph and table.

6.5.2. Response Time Per Threads

The below graph records each thread response time calculated by recording the thread start time and end time then retrieving the difference as the thread response time. The subsequent table displays a data statics summary including the average, max, min, and standard deviation for all the user sessions involved in the run.

![Response Time/Thread](image)

*Figure 29 – Response Time in Seconds Per Thread*

<table>
<thead>
<tr>
<th>STATS Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>9.02</td>
</tr>
<tr>
<td>Max</td>
<td>49.14</td>
</tr>
<tr>
<td>Min</td>
<td>1.38</td>
</tr>
<tr>
<td>StdDev</td>
<td>7.22</td>
</tr>
</tbody>
</table>

*Table 21 – Third run response time data summary*
The third run shows a better overall response time from the second run. However, the thread response time shows to be little longer than the first run.

### 6.5.3. Heap Ratio Per Thread

The below graph shows the JVM heap ratio recorded toward at the end of the execution of each thread, or a user session, by retrieving the JVM allocated memory divided by max memory variable. The subsequent table displays the heap ratio data statics summary including the first Quartile, second Quartile (median), and third Quartile.

![Heap Ratio Per Thread](image)

**Figure 30 – HEAP RATIO PER THREAD**

<table>
<thead>
<tr>
<th>STATS Analysis</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>First Quartile</td>
<td>42</td>
</tr>
<tr>
<td>Second Quartile (median)</td>
<td>58</td>
</tr>
<tr>
<td>Third Quartile</td>
<td>70</td>
</tr>
</tbody>
</table>

**Table 22 – Third run heap ratio data summary**
The heap ratio also shows to be more similar to the first run with a slight or insignificant increase resulted from the addition of the recovery components.
CONCLUSION

We have built an API that can be attached to a JVM or multiple JVM’s hosting web applications heavy on the integration with DMBS, such as an ecommerce or an online store application. The API, Cognitive Performance Application, monitors the JVM performance and learns to detect its bottleneck. The application also has predictive engine that can forecast the next bottleneck occurrence. Once a bottleneck is detected or forecasted, the application uses a temp queue structure to stream down the demand intelligently and only when needed in order to avoid the system from bottlenecking. The application utilizes feed forward ANN, to learn from the performance parameters how to detect the bottleneck. The use of ANN in our case was to emphasize generalization on the design and to demo how the ANN can weight the most determining factors leading to bottleneck in case of ambiguity or in case of absence of domain expert. The application also uses regression models to predict the next bottleneck in order to help the system become a proactive one. We have run number of test to examine the different components of the performance API. We ran a test without invoking the different components of the API with certain hardware specifications managing to bottleneck the system very frequently with an average CPU utilization of 94.15%. Then we ran a second test on the same conditions, but with the addition of two components of the API, the learning engine and the queue engine. We noticed a significant improvement in terms of the system not bottlenecking as frequently as before and with an overall CPU utilization of 87.80%. We concluded with a third test running on the same conditions as the first and the seconds tests, and fully deployed the
three components of the application; learning, analytics, and queuing engines. The test shows further improvements with a better average CPU utilization of 84.60%
REFERENCES


[34] Shuigeng Zhou; Songmao Zhang; George Karypis, “Advanced Data Mining and Applications” 8th International Conference, ADMA 2012, Nanjing, China, December 15-18, 2012


[38] Peifeng Niu; Yunpeng Ma; Mengning Li; Shanshan Yan; Guoqiang Li," A Kind of Parameters Self-adjusting Extreme Learning Machine", Neural Processing Letters, DOI: 10.1007/s11063-016-9496-z, December 2016


72
APPENDIX A

A.1. Estore Schema Create Statement
CREATE DATABASE `estore` /*!40100 DEFAULT CHARACTER SET utf8 */;

A.2. Address Table Create Statement

A.2.1. Description
This table stores the addresses of users or organizations in the WebSphere Commerce system. The addresses can be the members' own addresses or for their friends, associates, or clients, and so on. Some columns here replace columns used in previous versions.

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS_ID</td>
<td>BIGINT NOT NULL</td>
<td>Unique reference number, internally generated. This is a primary key, replacing the SASHNBR column in the SHADDR table used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>ADDRESSTYPE</td>
<td>CHAR(5)</td>
<td>The purpose of the address. Valid values are: S (shipto), B (billto), and SB (both shipto and billto). If this is unspecified when creating a new address, the business logic will default to SB.</td>
</tr>
<tr>
<td>MEMBER_ID</td>
<td>BIGINT NOT NULL</td>
<td>Foreign key to the MEMBER table for the member who owns this address.</td>
</tr>
<tr>
<td>ADDRBOOK_ID</td>
<td>BIGINT NOT NULL</td>
<td>Foreign key to the ADDRBOOK table for the address book to which this address belongs, replacing the SASHNBR column in the SHADDR table used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ORGUNITNAME</td>
<td>VARCHAR(128)</td>
<td>Name of the organizational unit to which this address information applies, if this is a business address. See comment on ADDRESS.ORGNAME column.</td>
</tr>
<tr>
<td>FIELD3</td>
<td>VARCHAR(64)</td>
<td>Compare with the BCFIELD3 column in the BUCONT table.</td>
</tr>
<tr>
<td>BILLINGCODE</td>
<td>VARCHAR(17)</td>
<td>A code to identify the shipping or billing information.</td>
</tr>
<tr>
<td>BILLINGCODETYPE</td>
<td>CHAR(2)</td>
<td>The code designating the system or method of code structure used for billing.</td>
</tr>
<tr>
<td>STATUS</td>
<td>CHAR(1)</td>
<td>Specifies the status of the address. Valid values are P (permanent or current) and T (temporary or historical). This column replaces SAADRFLAG used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>ORGNAME</td>
<td>VARCHAR(128)</td>
<td>If this is a business address, the name of the organization for this address. This is not the name of the organization that owns the address book with this address. This column replaces SAREPCOM used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>ISPRIMARY</td>
<td>INTEGER</td>
<td>A user or organization can have multiple addresses of each ADDRESSTYPE and one of them to be designated as primary. For example, one of the shipping addresses can be designated as the primary shipping address. Valid values are 1 (primary address) or 0 (non-primary address).</td>
</tr>
<tr>
<td>LASTNAME</td>
<td>VARCHAR(128)</td>
<td>Last name of the person to which this address applies, replacing SALNAME used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PERSONTITLE</td>
<td>VARCHAR(50)</td>
<td>Title of the person to which this address applies, replacing SATITLE used in previous versions of WebSphere Commerce or WebSphere Commerce Suite. LDAP uses a length of 50. Valid values are Dr, Prof, Rev, Mr, Mrs, Ms, and N (not provided). The default is N.</td>
</tr>
<tr>
<td>FIRSTNAME</td>
<td>VARCHAR(128)</td>
<td>First name of the person to which this address applies, replacing SAFNAME used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>MIDDLENAME</td>
<td>VARCHAR(128)</td>
<td>Middle name or initials of the person to which this address applies, replacing SAMNAME used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>BUSINESSTITLE</td>
<td>VARCHAR(128)</td>
<td>The business title. For example, Manager or Chief Executive Officer. LDAP uses a length of 128 characters for title attribute in ePerson. Compare with the BUCONT.BCTITLE column.</td>
</tr>
<tr>
<td>PHONE1</td>
<td>VARCHAR(32)</td>
<td>The primary phone number, replacing SAPHONE1 used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>FAX1</td>
<td>VARCHAR(32)</td>
<td>The primary fax number 1, replacing SAFAX used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>PHONE2</td>
<td>VARCHAR(32)</td>
<td>The secondary phone number, replacing SAPHONE2 used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>Column</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ADDRESS1</td>
<td>VARCHAR(100)</td>
<td>Address line 1, replacing SAADDR1 used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>FAX2</td>
<td>VARCHAR(32)</td>
<td>The secondary fax number.</td>
</tr>
<tr>
<td>NICKNAME</td>
<td>VARCHAR(254)</td>
<td>The nickname or identifier of the address, replacing SANICK used in previous versions of WebSphere Commerce or WebSphere Commerce Suite. Among all the addresses owned by a member, their nicknames must be unique.</td>
</tr>
<tr>
<td>ADDRESS2</td>
<td>VARCHAR(50)</td>
<td>Address line 2, replacing SAADDR2 used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>ADDRESS3</td>
<td>VARCHAR(50)</td>
<td>Address line 3, replacing SAADDR3 used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>CITY</td>
<td>VARCHAR(128)</td>
<td>The city, replacing SACITY used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>STATE</td>
<td>VARCHAR(128)</td>
<td>State or province, replacing SASTATE used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>COUNTRY</td>
<td>VARCHAR(128)</td>
<td>The country or region, replacing SACNTRY used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>ZIPCODE</td>
<td>VARCHAR(40)</td>
<td>ZIP or postal code, replacing SAZIPC used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>EMAIL1</td>
<td>VARCHAR(256)</td>
<td>The primary e-mail address, replacing SAEMAIL1 used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EMAIL2</td>
<td>VARCHAR(256)</td>
<td>The secondary e-mail address.</td>
</tr>
<tr>
<td>PHONE1TYPE</td>
<td>CHAR(3)</td>
<td>Phone type for the primary phone number, such as <strong>TTY</strong> for a teletypewriter for people who have a hearing impairment or <strong>PHN</strong> for a standard phone. This column replaces <strong>SADPHTYP</strong>.</td>
</tr>
<tr>
<td>PHONE2TYPE</td>
<td>CHAR(3)</td>
<td>Phone type for the secondary phone number, such as <strong>TTY</strong> for a teletypewriter for people who have a hearing impairment or <strong>PHN</strong> for a standard phone. This column replaces <strong>SAEPHTYP</strong>.</td>
</tr>
<tr>
<td>PUBLISHPHONE1</td>
<td>INTEGER</td>
<td>Specifies whether or not the primary phone number is listed. Valid values are 1 (listed) or 0 (unlisted).</td>
</tr>
<tr>
<td>PUBLISHPHONE2</td>
<td>INTEGER</td>
<td>Specifies whether or not the secondary phone number is listed. Valid values are 1 (listed) or 0 (unlisted).</td>
</tr>
<tr>
<td>BESTCALLINGTIME</td>
<td>CHAR(1)</td>
<td>The best time to call, replacing <strong>SABTCALL</strong> used in previous versions of WebSphere Commerce or WebSphere Commerce Suite. Valid values are D (daytime) and E (evening).</td>
</tr>
<tr>
<td>PACKAGESUPPRESSION</td>
<td>INTEGER</td>
<td>Specifies whether or not to include package inserts. Valid values are 1 (include) or 0 (do not include).</td>
</tr>
<tr>
<td>LASTCREATE</td>
<td>TIMESTAMP</td>
<td>The date and time the row was created, replacing <strong>SASTMP</strong> used in previous versions of WebSphere Commerce or WebSphere Commerce Suite.</td>
</tr>
<tr>
<td>OFFICEADDRESS</td>
<td>VARCHAR(128)</td>
<td>Desktop delivery or office address associated with a shipping address.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SELFADDRESS</td>
<td>INTEGER NOT NULL DEFAULT 0</td>
<td>Specifies whether or not the address belongs to the member (user or organization). The address that belongs to a member is the one that was entered as part of registration. Valid values are 1 (the address belongs to the member) or 0 (the address does not belong to the user; it may belong to a friend, associate, or relative). A member can have only one permanent self address.</td>
</tr>
<tr>
<td>FIELD1</td>
<td>VARCHAR(64)</td>
<td>Customizable. This column replaces the SAFIELD1 column in the SHADDR table (used in previous versions of WebSphere Commerce or WebSphere Commerce Suite) and accepts up to 3 characters.</td>
</tr>
<tr>
<td>FIELD2</td>
<td>VARCHAR(64)</td>
<td>Customizable. This column replaces the SAFIELD2 column in the SHADDR table (used in previous versions of WebSphere Commerce or WebSphere Commerce Suite) and accepts up to one character.</td>
</tr>
<tr>
<td>TAXGEOCODE</td>
<td>VARCHAR(254)</td>
<td>Tax code based on geographical region, used for integration with Taxware.</td>
</tr>
<tr>
<td>SHIPPINGGEOCODE</td>
<td>VARCHAR(254)</td>
<td>Shipping code based on geographical region. This value is used for integration with Taxware.</td>
</tr>
</tbody>
</table>

**Address Table**

**A.2.2. SQL Statement**

```sql
DROP TABLE IF EXISTS `estore`.`address`;
CREATE TABLE  `estore`.`address` (
    `ADDRESS_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `ADDRESSTYPE` varchar(5) NOT NULL,
    `ISPRIMARY` int(1) unsigned NOT NULL,
    `MEMBER_ID` int(10) unsigned NOT NULL,
    `STATUS` char(1) NOT NULL,
    `LASTNAME` varchar(45) NOT NULL,
);```
A.3. Inventory Table Create Statement

A.3.1. Description

Each row of this table contains a quantity amount representing the inventory for a particular CatalogEntry. The CatalogEntry is available to be shipped from a FulfillmentCenter on behalf of a Store. This table cannot be used in conjunction with Available To Promise (ATP) inventory allocation. It is used only when ATP inventory is not enabled (refer to the INVENTORYSYSTEM column of the STORE table).

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Column Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATENTRY_ID</td>
<td>BIGINT NOT NULL</td>
<td>The CatalogEntry.</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>DOUBLE NOT NULL DEFAULT 0</td>
<td>The quantity amount, in units indicated by QUANTITYMEASURE.</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------</td>
<td>------------------------------------------------------------</td>
</tr>
<tr>
<td>FFMCENTER_ID</td>
<td>INTEGER NOT NULL</td>
<td>The FulfillmentCenter.</td>
</tr>
<tr>
<td>STORE_ID</td>
<td>INTEGER NOT NULL</td>
<td>The Store.</td>
</tr>
<tr>
<td>QUANTITYMEASURE</td>
<td>CHAR(16) NOT NULL DEFAULT 'C62'</td>
<td>The unit of measurement for QUANTITY.</td>
</tr>
<tr>
<td>INVENTORYFLAGS</td>
<td>INTEGER NOT NULL DEFAULT 0</td>
<td>Bit flags, from low to high order, indicating how QUANTITY is used: 1 = noUpdate. The default UpdateInventory task command does not update QUANTITY. 2 = noCheck. The default CheckInventory and UpdateInventory task commands do not check QUANTITY.</td>
</tr>
<tr>
<td>OPTCOUNTER</td>
<td>SMALLINT</td>
<td>Reserved for IBM internal use.</td>
</tr>
</tbody>
</table>

Inventory Table

A.3.2. SQL Statement

```sql
DROP TABLE IF EXISTS `estore`.`inventory`;
CREATE TABLE `estore`.`inventory` (
    `PRODUCT_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `QUANTITY` double NOT NULL,
    PRIMARY KEY (`PRODUCT_ID`),
    CONSTRAINT `FK_inventory_1` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product` (`PRODUCT_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

A.4. Language Table Create Statement

A.4.1. Description

Each row of this table represents a language. Our Commerce system supports multiple languages and is translated into ten languages by default. Using the predefined ISO codes users can add other supported languages.
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
</table>
| LANGUAGE_ID | INTEGER NOT NULL | The language ID (primary key of this table) for each language. Foreign key to the tables that contain language-dependent information. The following is the list of language components (language ID codes) of the locale:  
- 1 = English (US)  
- 2 = French  
- 3 = German  
- 4 = Italian  
- 5 = Spanish  
- 6 = Brazilian Portuguese  
- 7 = Simplified Chinese  
- 8 = Traditional Chinese  
- 9 = Korean |
| LOCALENAME  | CHAR (16) NOT NULL | A Java locale used to represent a political, geographical, or cultural region that has a distinct language and customs. The following is the list of locale used for formatting:  
- en_US = United States  
- fr_FR = France  
- de_DE = Germany  
- it_IT = Italy  
- es_ES = Spain  
- pt_BR = Brazil  
- zh_CN = China  
- zh_TW = Taiwan  
- ko_KR = Korea  
- ja_JP = Japan  
- ru_RU = Russian  
- ro_RO = Romanian  
- pl_PL = Polish |
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LANGUAGE</td>
<td>CHAR (5)</td>
<td>Language component of the locale. NOTE: A value must be entered for this column: en = English (US) fr = French de = German it = Italian es = Spanish pt = Brazilian Portuguese zh = Simplified Chinese zh = Traditional Chinese ko = Korean ja = Japanese ru = Russian ro = Romanian pl = Polish To choose another language component, refer to the existing ISO codes.</td>
</tr>
<tr>
<td>COUNTRY</td>
<td>CHAR (5)</td>
<td>Country or region component of the locale. NOTE: A value must be entered for this column: US = United States FR = France DE = Germany IT = Italy ES = Spain BR = Brazil CN = China TW = Taiwan KR = Korea JP = Japan RU = Russia RO = Romania PL = Poland To choose another country or region, refer to the existing ISO codes.</td>
</tr>
<tr>
<td>VARIANT</td>
<td>CHAR (10)</td>
<td>Variant component of the locale. Used to specify the locale encoding character set</td>
</tr>
<tr>
<td>ENCODING</td>
<td>VARCHAR (32)</td>
<td>The character encoding value that the browser uses to display the page in the selected language.</td>
</tr>
</tbody>
</table>
### Language Table

**A.4.2. SQL Statement**

```sql
DROP TABLE IF EXISTS `estore`.`language`;
CREATE TABLE `estore`.`language` (
  `LANGUAGE_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `LOCALENAME` varchar(45) NOT NULL,
  `LANGUAGE` varchar(45) NOT NULL,
  `COUNTRY` varchar(45) NOT NULL,
  `VARIANT` varchar(45) NOT NULL,
  PRIMARY KEY (`LANGUAGE_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

### Listprice Table Create Statement

**A.5. Listprice Table Create Statement**

**A.5.1. Description**

Each row of this table represents a ListPrice in a particular currency for each Catalog Entry.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATENTRY_ID</td>
<td>BIGINT NOT NULL</td>
<td>The CatalogEntry.</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>CHAR (3) NOT NULL</td>
<td>The Currency of the ListPrice. This is a currency code as per ISO 4217 standards.</td>
</tr>
<tr>
<td>LISTPRICE</td>
<td>DECIMAL (20,5) NOT NULL</td>
<td>The amount of the ListPrice.</td>
</tr>
<tr>
<td>OID</td>
<td>VARCHAR (64)</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td>OPTCOUNTER</td>
<td>SMALLINT</td>
<td>The optimistic concurrency control counter for the table.</td>
</tr>
</tbody>
</table>

**A.5.2. SQL Statement**

```sql
DROP TABLE IF EXISTS `estore`.`listprice`;
CREATE TABLE `estore`.`listprice` (
  `PRODUCT_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `PRODUCT_CURRENCY` CHAR (3) NOT NULL,  
  `PRODUCT_LISTPRICE` DECIMAL (20,5) NOT NULL, 
  `PRODUCT_OID` VARCHAR (64),  
  `PRODUCT_OPTCOUNTER` SMALLINT,  
  PRIMARY KEY (`PRODUCT_ID`),
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```
A.6. **Member Table Create Statement**

A.6.1. **Description**
Stores the list of members (participants) of the WebSphere Commerce system. A member is either a user, an organizational entity or a member group.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMBER_ID</td>
<td>BIGINT NOT NULL</td>
<td>ID for the member, a unique primary key for this table.</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR (3) NOT NULL</td>
<td>The type of member as follows: O = OrgEntity, U = User, G = MemberGroup</td>
</tr>
<tr>
<td>STATE</td>
<td>INTEGER</td>
<td>The registration approval status as follows: 0 = pending approval, 1 = approved, 2 = rejected, 3 = pending email activation, Null = approval is not applicable (approved)</td>
</tr>
<tr>
<td>OPTCOUNTER</td>
<td>SMALLINT</td>
<td>The optimistic concurrency control counter for the table.</td>
</tr>
</tbody>
</table>

**Member Table**

A.6.2. **SQL Statement**

```sql
DROP TABLE IF EXISTS `estore`.`member`;
CREATE TABLE `estore`.`member` (
    `MEMBER_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `TYPE` varchar(45) NOT NULL,
    `state` varchar(45) NOT NULL,
    PRIMARY KEY (`MEMBER_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```
### A.7. OrderItems Table Create Statement

#### A.7.1. Description

Each row of this table represents an order item in an order.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDERITEMS_ID</td>
<td>BIGINT NOT NULL</td>
<td>Generated unique key.</td>
</tr>
<tr>
<td>STOREENT_ID</td>
<td>INTEGER NOT NULL</td>
<td>The store entity the order (this order item is part of) is part of. This is normally a store unless STATUS is Q, in which case it is normally a store group.</td>
</tr>
<tr>
<td>ORDERS_ID</td>
<td>BIGINT NOT NULL</td>
<td>The order of which this order item is part.</td>
</tr>
<tr>
<td>TERMCOND_ID</td>
<td>BIGINT</td>
<td>The TermAndCondition, if known, that determined the price for this order item.</td>
</tr>
<tr>
<td>TRADING_ID</td>
<td>BIGINT</td>
<td>The TradingAgreement, if known, that determines the TermAndCondition objects (including how the price is determined) that apply to this order item.</td>
</tr>
<tr>
<td>ITEMSPC_ID</td>
<td>BIGINT</td>
<td>The specified item to be allocated from available inventory and shipped to the customer.</td>
</tr>
<tr>
<td>CATENTRY_ID</td>
<td>BIGINT</td>
<td>The catalog entry, if any, of the product being purchased.</td>
</tr>
<tr>
<td>PARTNUM</td>
<td>VARCHAR (64)</td>
<td>The part number of the catalog entry(CATENTRY.PARTNUMBER) for the product.</td>
</tr>
<tr>
<td>SHIPMODE_ID</td>
<td>INTEGER</td>
<td>The shipping mode, if still known.</td>
</tr>
<tr>
<td>FFMCENTER_ID</td>
<td>INTEGER</td>
<td>The fulfillment center, if known, from which the product will ship.</td>
</tr>
<tr>
<td>MEMBER_ID</td>
<td>BIGINT NOT NULL</td>
<td>The customer of the order item (which is the same as the customer of the order).</td>
</tr>
<tr>
<td>ADDRESS_ID</td>
<td>BIGINT</td>
<td>The shipping address, if any, for this order item.</td>
</tr>
<tr>
<td>ALLOCADDRESS_ID</td>
<td>BIGINT</td>
<td>The shipping address used when inventory for this order item was allocated or backordered.</td>
</tr>
<tr>
<td><strong>PRICE</strong></td>
<td>DECIMAL (20,5)</td>
<td>The price for the nominal quantity of the product (CATENTSHIP.NOMINALQUANTITY).</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>LINEITEMTYPE</strong></td>
<td>CHAR (4)</td>
<td>If specified, indicates the type of the order item. ALT = the order item represents an alternative item (might not be exactly what the customer requested).</td>
</tr>
<tr>
<td><strong>STATUS</strong></td>
<td>CHAR (1) NOT NULL</td>
<td>The status for the order item. It may not be the same as the status in the order.</td>
</tr>
<tr>
<td><strong>OUTPUTQ_ID</strong></td>
<td>BIGINT</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td><strong>INVENTORYSTATUS</strong></td>
<td>CHAR (4) NOT NULL DEFAULT 'NALC'</td>
<td>The allocation status of inventory for this order item: NALC: Inventory is not allocated nor on back-order. BO: Inventory is on back-order. ALLC: Inventory is allocated. FUL: Inventory has been released for fulfillment. AVL: Inventory is available.</td>
</tr>
<tr>
<td><strong>LASTCREATE</strong></td>
<td>TIMESTAMP</td>
<td>The time this order item was created.</td>
</tr>
<tr>
<td><strong>LASTUPDATE</strong></td>
<td>TIMESTAMP</td>
<td>The most recent time this order item was updated. Changing inventory allocation related information does not cause this timestamp to be updated (refer to the LASTALLOCUPDATE column).</td>
</tr>
<tr>
<td><strong>FULFILLMENTSTATUS</strong></td>
<td>CHAR (4) NOT NULL DEFAULT 'INT'</td>
<td>The fulfillment status of the order item: INT = not yet released for fulfillment. OUT = released for fulfillment. SHIP = shipment confirmed. HOLD = a temporary status between INT and OUT status.</td>
</tr>
<tr>
<td>LASTALLOCUPDATE</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------</td>
<td></td>
</tr>
<tr>
<td>The most recent time inventory was checked (for unallocated order items), allocated, or backordered, for this order item.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OFFER_ID</th>
<th>BIGINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>The offer, if any, and if it still exists, from which PRICE was obtained.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIMERELEASED</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>The time this order item was released for fulfillment.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TIMESHIPTED</th>
<th>TIMESTAMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>The time this order item was manifested for shipment.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CURRENCY</th>
<th>CHAR (10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The currency of order item monetary amounts other than BASEPRICE. This is the same as the currency of the order, ORDERS.CURRENCY. This is a currency code according to ISO 4217 standards.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COMMENTS</th>
<th>VARCHAR (254)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comments from the customer, such as a greeting for a gift.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOTALPRODUCT</th>
<th>DECIMAL (20,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRICE times QUANTITY.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QUANTITY</th>
<th>DOUBLE NOT NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>The result of multiplying QUANTITY by CATENTSHIP.NOMINALQUANTITY must be a multiple of CATENTSHIP.QUANTITYMULTIPLE. And it represents the actual quantity being purchased, in the unit of measurement specified by CATENTSHIP.QUANTITYMEASURE.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAXAMOUNT</th>
<th>DECIMAL (20,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The total sales taxes associated with this order item, in the currency specified by CURRENCY.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TOTALADJUSTMENT</th>
<th>DECIMAL (20,5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The total of the monetary amounts of the order item adjustments for this order item, in the currency specified by CURRENCY. This column also includes all kinds of shipping charge adjustments like discount, coupon, shipping adjustment and surcharge.</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------</td>
</tr>
<tr>
<td>SHIPTAXAMOUNT</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>ESTAVAILTIME</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>FIELD1</td>
<td>INTEGER</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>FIELD2</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>ALLOCATIONGROUP</td>
<td>BIGINT</td>
</tr>
<tr>
<td>SHIPCHARGE</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>BASEPRICE</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>BASECURRENCY</td>
<td>CHAR (3)</td>
</tr>
<tr>
<td>TRACKNUMBER</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TRACKDATE</td>
<td>TIMESTAMP</td>
</tr>
</tbody>
</table>
**PREPAREFLAGS**

<table>
<thead>
<tr>
<th>INTEGER</th>
<th>NOT NULL</th>
<th>DEFAULT 0</th>
</tr>
</thead>
</table>

Contains the following bit flags indicating special processing associated with this order item:

1. **gene**
   - The order item was generated during a previous execution of the OrderPrepare command. The next time the OrderPrepare command is run, it first removes all generated order items so that they can be re-generated if and as applicable.

2. **priceOverride**
   - The price of the order item has been manually entered and will not be changed by customer commands.

4. **fulfillmentCenterOverride**
   - The fulfillment center has been manually specified and will not be changed by customer commands.

8. **directCalculationCodeAttachment**
   - CalculationCodes may be directly attached to the order item. The default CalculationCodeCombineMethod will not look for direct attachments unless this flag is true.
shippingChargeByCarrier

The contract for this order item indicates that no shipping charge will be calculated by WebSphere Commerce. It may be calculated and charged by the carrier on fulfillment.

quotation

The order item was obtained from a quotation. The price will not be automatically refreshed by customer commands.

notConfigured

Price lookup and inventory allocation for this order item is not done using the component items found in the OICOMPddST table. This flag does not need to be set for order items whose CONFIGURATIONID column value is null.

autoAdd

This order item, such as a free gift, was automatically added to the order by the OrderCalculate command. This flag is for information-display purposes only and does not affect price calculations.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>256</td>
<td>hasPersonalizationAttributes</td>
</tr>
<tr>
<td></td>
<td>The order item contains personalization attributes.</td>
</tr>
<tr>
<td>512</td>
<td>skipRepricing</td>
</tr>
<tr>
<td></td>
<td>The order item was marked as not to be repriced. For example, it is set as skipRepricing when a quote is generated into an order if the quote has not expired.</td>
</tr>
<tr>
<td>1024</td>
<td>shippingAdjusted</td>
</tr>
<tr>
<td></td>
<td>The order items shipping charge is manually adjusted.</td>
</tr>
<tr>
<td>2048</td>
<td>priceRefreshed</td>
</tr>
<tr>
<td></td>
<td>The order items price has already been refreshed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CORRELATIONGROUP</th>
<th>BIGINT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normally this is the same as ORDERITEMS_ID, except:</td>
<td></td>
</tr>
<tr>
<td>1. When an order item is split by the AllocateInventory task command, the newly created OrderItem inherits the CORRELATIONGROUP value from the original order item.</td>
<td></td>
</tr>
<tr>
<td>2. When the PREPAREFLAGS column indicates &quot;quotation&quot;, the OrderItem inherits the CORRELATIONGROUP value from the corresponding OrderItem in the parent order.</td>
<td></td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>PROMISEDAVAILTIME</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>SHIPPINGOFFSET</td>
<td>INTEGER NOT NULL DEFAULT 0</td>
</tr>
<tr>
<td>NEEDEDQUANTITY</td>
<td>INTEGER NOT NULL DEFAULT 0</td>
</tr>
<tr>
<td>ALLOCQUANTITY</td>
<td>INTEGER NOT NULL DEFAULT 0</td>
</tr>
<tr>
<td>ALLOCFFMC_ID</td>
<td>INTEGER</td>
</tr>
<tr>
<td>ORDRERELEASENUM</td>
<td>INTEGER</td>
</tr>
<tr>
<td>CONFIGURATIONID</td>
<td>VARCHAR (128)</td>
</tr>
</tbody>
</table>
### A.7.2. SQL Statement

```sql
DROP TABLE IF EXISTS `estore`.`orderitems`;
```
CREATE TABLE `estore`.`orderitems` (  
`ORDERITEMS_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,  
`ORDERS_ID` int(10) unsigned NOT NULL,  
`PRODUCT_ID` int(10) unsigned NOT NULL,  
`MEMBER_ID` int(10) unsigned NOT NULL,  
`ADDRESS_ID` int(10) unsigned NOT NULL,  
`PRICE` decimal(20,5) NOT NULL,  
`QUANTITY` double NOT NULL,  
`TOTALPRODUCT` decimal(20,5) NOT NULL,  
`DESCRIPTION` varchar(45) NOT NULL,  
PRIMARY KEY (`ORDERITEMS_ID`),  
KEY `FK_ORDERITEMS_1` (`ORDERS_ID`),  
KEY `FK_orderitems_2` (`PRODUCT_ID`),  
KEY `FK_orderitems_3` (`MEMBER_ID`),  
KEY `FK_orderitems_4` (`ADDRESS_ID`),  
CONSTRAINT `FK_ORDERITEMS_1` FOREIGN KEY (`ORDERS_ID`) REFERENCES `orders` (`ORDERS_ID`),  
CONSTRAINT `FK_orderitems_2` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product` (`PRODUCT_ID`),  
CONSTRAINT `FK_orderitems_3` FOREIGN KEY (`MEMBER_ID`) REFERENCES `member` (`MEMBER_ID`),  
CONSTRAINT `FK_orderitems_4` FOREIGN KEY (`ADDRESS_ID`) REFERENCES `address` (`ADDRESS_ID`)  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

A.8. Order Table Create Statement

A.8.1. Description

Each row in this table represents an order in a store

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORDERS_ID</td>
<td>BIGINT NOT NULL</td>
<td>Generated unique key.</td>
</tr>
<tr>
<td>ORMORDER</td>
<td>CHAR (30)</td>
<td>A merchant-assigned order reference number, if any.</td>
</tr>
<tr>
<td>ORGENTITY_ID</td>
<td>BIGINT</td>
<td>The immediate parent organization ID of the creator.</td>
</tr>
<tr>
<td>TOTALPRODUCT</td>
<td>DECIMAL (20,5) DEFAULT 0</td>
<td>The sum of ORDERITEMS.TOTALPRODUCT for the OrderItems in the Order.</td>
</tr>
<tr>
<td>Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TOTALTAX</td>
<td>DECIMAL (20,5)</td>
<td>The sum of ORDERITEMS.TAXAMOUNT for the OrderItems in the Order.</td>
</tr>
<tr>
<td>TOTALSHIPPING</td>
<td>DECIMAL (20,5)</td>
<td>The sum of ORDERITEMS.SHIPCHARGE for the OrderItems in the Order.</td>
</tr>
<tr>
<td>TOTALTAXSHIPPING</td>
<td>DECIMAL (20,5)</td>
<td>The sum of ORDERITEMS.SHIPTAXAMOUNT for the OrderItems in the Order.</td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR (254)</td>
<td>A mnemonic description of the order, entered by the customer, suitable for display to the customer.</td>
</tr>
<tr>
<td>STOREENT_ID</td>
<td>INTEGER NOT NULL</td>
<td>The store entity the order is part of. This is normally a store unless STATUS is Q, in which case it is normally a store group.</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>CHAR (10)</td>
<td>The currency for monetary amounts associated with this order. This is the currency code according to ISO 4217 standards.</td>
</tr>
<tr>
<td>LOCKED</td>
<td>CHAR (1)</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td>TIMEPLACED</td>
<td>TIMESTAMP</td>
<td>The time this order was processed by the OrderProcess command.</td>
</tr>
<tr>
<td>LASTUPDATE</td>
<td>TIMESTAMP</td>
<td>The time this order was most recently updated.</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td>DOUBLE NOT NULL</td>
<td>Can be used by a user interface to control the sequence of orders in a list.</td>
</tr>
<tr>
<td>STATUS</td>
<td>VARCHAR (3)</td>
<td>The status of the order. This is similar to order states. For specific values of status and their descriptions, refer to the order states page, but omit states 'Y' and 'Z'.</td>
</tr>
<tr>
<td>MEMBER_ID</td>
<td>BIGINT NOT NULL</td>
<td>The customer that placed the order.</td>
</tr>
<tr>
<td>FIELD1</td>
<td>INTEGER</td>
<td>Customizable.</td>
</tr>
<tr>
<td>ADDRESS_ID</td>
<td>BIGINT</td>
<td>This column is deprecated. In versions WC 5.6.1 and earlier, this column contained the billing address. In version WC 6.0 and later, the billing address is part of payment data and is stored in PPCEXTDATA table as encrypted data.</td>
</tr>
<tr>
<td>Field/Column</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FIELD2</td>
<td>DECIMAL (20,5)</td>
<td>Customizable.</td>
</tr>
<tr>
<td>PROVIDERORDERNUM</td>
<td>INTEGER</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td>SHIPASCOMPLETE</td>
<td>CHAR (1) NOT NULL DEFAULT 'Y'</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td>FIELD3</td>
<td>VARCHAR (254)</td>
<td>Customizable.</td>
</tr>
<tr>
<td>TOTALADJUSTMENT</td>
<td>DECIMAL (20,5) DEFAULT 0</td>
<td>The sum of ORDERITEMS.TOTALADJUSTMENT for the order items in the order. This column also includes all kinds of shipping charge adjustments like discount, coupon, shipping adjustment and surcharge.</td>
</tr>
<tr>
<td>ORDCHNLTYP_ID</td>
<td>BIGINT</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td>COMMENTS</td>
<td>VARCHAR (254)</td>
<td>Comments from the customer.</td>
</tr>
<tr>
<td>NOTIFICATIONID</td>
<td>BIGINT</td>
<td>Notification identifier referring to the rows in the NOTIFY table that store notification attributes. These attributes override the defaults for notifications related to this order.</td>
</tr>
<tr>
<td>TYPE</td>
<td>CHAR (3)</td>
<td>For an order, TYPE indicates whether it is a regular order (ORD), private requisition list order (PRL), shareable requisition list order (SRL), standing order (STD), quotation order (QUT), quote (QOT), profile order (QUK), recurring order (REC), or subscription (SUB).</td>
</tr>
<tr>
<td>OPTCOUNTER</td>
<td>SMALLINT</td>
<td>The optimistic concurrency control counter for the table.</td>
</tr>
<tr>
<td>EDITOR_ID</td>
<td>BIGINT</td>
<td>Stores the ID of the person editing the order.</td>
</tr>
<tr>
<td>BUSCHN_ID</td>
<td>INTEGER</td>
<td>The Business Channel ID of the order.</td>
</tr>
<tr>
<td>SOURCEID</td>
<td>BIGINT</td>
<td>This field indicates the sources from which this order came, if any. This field is relevant for orders that were generated from a previously saved quote, for example. In that situation, the Quotetld would be saved here.</td>
</tr>
<tr>
<td>Field</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EXPIREDATE</td>
<td>TIMESTAMP</td>
<td>This entry is only relevant when the row refers to a quote, type &quot;QOT&quot;. This field then indicates the expiration date for that quote.</td>
</tr>
<tr>
<td>BLOCKED</td>
<td>SMALLINT</td>
<td>Indicates if this order has a block placed against it or not.</td>
</tr>
<tr>
<td>OPSYSTEM_ID</td>
<td>INTEGER</td>
<td>This field indicates the Order Processing system for the order.</td>
</tr>
<tr>
<td>TRANSFERSTATUS</td>
<td>SMALLINT</td>
<td>Order transferring status: 0, it is in the initial status. 1, it is in the transferring status. 2, it is in the transferred status. null, the same as 0.</td>
</tr>
<tr>
<td>BUYERPO_ID</td>
<td>BIGINT</td>
<td>The Buyer PO Number Id related to an order</td>
</tr>
</tbody>
</table>

**A.8.2. SQL Statement**

```sql
DROP TABLE IF EXISTS `estore`.`orders`;
CREATE TABLE `estore`.`orders` (
    `ORDERS_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `TOTALPRODUCT` decimal(20,5) NOT NULL,
    `TOTALTAX` decimal(20,5) NOT NULL,
    `TOTALSHIPPING` decimal(20,5) NOT NULL,
    `DESCRIPTION` varchar(45) NOT NULL,
    `CURRENCY` char(10) NOT NULL,
    `TIMEPLACED` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    `STATUS` char(3) NOT NULL,
    `BILLING_ADDRESS_ID` int(10) unsigned NOT NULL,
    `TOTALADJUSTMENT` decimal(20,5) NOT NULL,
    `MEMBER_ID` int(10) unsigned NOT NULL,
    PRIMARY KEY (`ORDERS_ID`),
    KEY `FK_ORDERS_1` (`MEMBER_ID`),
    KEY `FK_orders_2` (`BILLING_ADDRESS_ID`),
    CONSTRAINT `FK_ORDERS_1` FOREIGN KEY (`MEMBER_ID`) REFERENCES `member` (`MEMBER_ID`),
    CONSTRAINT `FK_orders_2` FOREIGN KEY (`BILLING_ADDRESS_ID`) REFERENCES `address` (`ADDRESS_ID`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;```
A.9. Product Table Create Statement

A.9.1. Description
This table holds the information related to a catalog entry. Examples of catalog entries include products, items, packages, and bundles.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATENTRY_ID</td>
<td>BIGINT NOT NULL</td>
<td>The internal reference number of the catalog entry.</td>
</tr>
<tr>
<td>MEMBER_ID</td>
<td>BIGINT NOT NULL</td>
<td>The reference number that identifies the owner of the catalog entry. Along with the PARTNUMBER, these columns are a unique index.</td>
</tr>
<tr>
<td>ITEMSPC_ID</td>
<td>BIGINT</td>
<td>The specified item that this catalog entry relates to. This column should only be populated for catalog entries that are of type &quot;ItemBean&quot;, &quot;PackageBean&quot;, or &quot;DynamicKitBean&quot;. Specified items are used for fulfillment.</td>
</tr>
<tr>
<td>CATENTTYPE_ID</td>
<td>CHAR (16) NOT NULL</td>
<td>Identifies the type of catalog entry. Foreign key to the CATENTTYPE table. The supported default types are: ProductBean, ItemBean, PackageBean, BundleBean and DynamicKitBean.</td>
</tr>
<tr>
<td>PARTNUMBER</td>
<td>VARCHAR (64) NOT NULL</td>
<td>The reference number that identifies the part number of the catalog entry. Along with the MEMBER_ID, these columns are a unique index.</td>
</tr>
<tr>
<td>MFPARTNUMBER</td>
<td>VARCHAR (64)</td>
<td>The part number used by the manufacturer to identify this catalog entry.</td>
</tr>
<tr>
<td>MFNAME</td>
<td>VARCHAR (64)</td>
<td>The name of the manufacturer of this catalog entry.</td>
</tr>
<tr>
<td>MARKFORDELETE</td>
<td>INTEGER NOT NULL</td>
<td>Indicates if this catalog entry has been marked for deletion: 0 = No. 1 = Yes.</td>
</tr>
<tr>
<td>URL</td>
<td>VARCHAR (254)</td>
<td>The URL to this catalog entry, which can be used as a download URL for soft goods.</td>
</tr>
<tr>
<td>FIELD1</td>
<td>INTEGER</td>
<td>Customizable.</td>
</tr>
<tr>
<td>FIELD</td>
<td>DATA_TYPE</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FIELD2</td>
<td>INTEGER</td>
<td>Customizable.</td>
</tr>
<tr>
<td>LASTUPDATE</td>
<td>TIMESTAMP</td>
<td>Indicates the last time the catalog entry was updated.</td>
</tr>
<tr>
<td>FIELD3</td>
<td>DECIMAL (20,5)</td>
<td>Customizable.</td>
</tr>
<tr>
<td>ONSPECIAL</td>
<td>INTEGER</td>
<td>This flag identifies if this catalog entry is on special.</td>
</tr>
<tr>
<td>ONAUCTION</td>
<td>INTEGER</td>
<td>This flag identifies if this catalog entry is on auction.</td>
</tr>
<tr>
<td>FIELD4</td>
<td>VARCHAR (254)</td>
<td>Customizable.</td>
</tr>
<tr>
<td>FIELD5</td>
<td>VARCHAR (254)</td>
<td>Customizable.</td>
</tr>
<tr>
<td>BUYABLE</td>
<td>INTEGER</td>
<td>Indicates whether this catalog entry can be purchased individually: 1=yes and 0=no.</td>
</tr>
<tr>
<td>OID</td>
<td>VARCHAR (64)</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td>BASEITEM_ID</td>
<td>BIGINT</td>
<td>The base item to which this catalog entry relates.</td>
</tr>
<tr>
<td>STATE</td>
<td>CHAR (1)</td>
<td>DEFAULT '1' Reserved for IBM internal use.</td>
</tr>
<tr>
<td>STARTDATE</td>
<td>TIMESTAMP</td>
<td>The date when this catalog entry is introduced. This column is for your interpretation and information only.</td>
</tr>
<tr>
<td>ENDDATE</td>
<td>TIMESTAMP</td>
<td>The date when this catalog entry is withdrawn. This column is for your interpretation and information only.</td>
</tr>
<tr>
<td>RANK</td>
<td>DOUBLE</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td>OPTCOUNTER</td>
<td>SMALLINT</td>
<td>The optimistic concurrency control counter for the table.</td>
</tr>
<tr>
<td>AVAILABILITYDATE</td>
<td>TIMESTAMP</td>
<td>The date that the product is available to customers for purchase. This column is for your interpretation and information only.</td>
</tr>
<tr>
<td>Column Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LASTORDERDATE</td>
<td>TIMESTAMP</td>
<td>The final date that the product is available for order. This is the latest date a customer can order the product. This column is for your interpretation and information only.</td>
</tr>
<tr>
<td>ENDOFSERVICEDATE</td>
<td>TIMESTAMP</td>
<td>The date when the product is no longer in service. For example, this date can be the date that a software manufacturer stops providing upgrades for a version of their product. This column is for your interpretation and information only.</td>
</tr>
<tr>
<td>DISCONTINUEDATE</td>
<td>TIMESTAMP</td>
<td>The date when the product is discontinued. For example, this date can be the date the manufacturer stops producing the product, or the date your store stops selling the product. It is recommended that this column be used consistently to avoid confusion. This column is for your interpretation and information only.</td>
</tr>
<tr>
<td>UP_MFNAME</td>
<td>VARCHAR (64)</td>
<td>The equivalent value of the MFNAME column in upper case characters. This column is used only for DB2 (LUW) database-types to enhance performance of text-based searches issued from Management Center.</td>
</tr>
<tr>
<td>UP_MFPARTNUMBER</td>
<td>VARCHAR (64)</td>
<td>The equivalent value of the MFPARTNUMBER column in upper case characters. This column is used only for DB2 (LUW) database-types to enhance performance of text-based searches issued from Management Center.</td>
</tr>
<tr>
<td>UP_PARTNUMBER</td>
<td>VARCHAR (64)</td>
<td>The equivalent value of the PARTNUMBER column in upper case characters. This column is used only for DB2 (LUW) database-types to enhance performance of text-based searches issued from Management Center.</td>
</tr>
</tbody>
</table>

### A.9.2. SQL Statement

```
DROP TABLE IF EXISTS `estore`.`product`;
```
### A.10. ProductDesc Table Create Statement

**A.10.1. Description**

This table holds language-dependent information related to a catalog entry.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATENTRY_ID</td>
<td>BIGINT NOT NULL</td>
<td>The internal reference number that indicates to which catalog entry this language-specific information relates.</td>
</tr>
<tr>
<td>LANGUAGE_ID</td>
<td>INTEGER NOT NULL</td>
<td>The identifier of the language. For a list of language components, see the LANGUAGE table.</td>
</tr>
<tr>
<td>NAME</td>
<td>CHAR (128)</td>
<td>The language-dependent name of this catalog entry.</td>
</tr>
<tr>
<td>SHORTDESCRIPTION</td>
<td>VARCHAR (254)</td>
<td>A short description of this catalog entry.</td>
</tr>
<tr>
<td>LONGDESCRIPTION</td>
<td>CLOB (1000000)</td>
<td>A long description of this catalog entry.</td>
</tr>
<tr>
<td>THUMBNAIL</td>
<td>VARCHAR (254)</td>
<td>The thumbnail image path of this catalog entry.</td>
</tr>
<tr>
<td>AUXDESCRIPTION1</td>
<td>VARCHAR (4000)</td>
<td>Additional description for this catalog entry.</td>
</tr>
<tr>
<td>FULLIMAGE</td>
<td>VARCHAR (254)</td>
<td>The full image path of this catalog entry.</td>
</tr>
<tr>
<td>AUXDESCRIPTION2</td>
<td>VARCHAR (4000)</td>
<td>Additional description for this catalog entry.</td>
</tr>
<tr>
<td>XMLDETAIL</td>
<td>CLOB (1000000)</td>
<td>Reserved for IBM internal use.</td>
</tr>
<tr>
<td><strong>AVAILABLE</strong></td>
<td><strong>INTEGER</strong></td>
<td>NOT NULL</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>PUBLISHED</strong></td>
<td><strong>INTEGER</strong></td>
<td>NOT NULL</td>
</tr>
<tr>
<td><strong>AVAILABILITYDATE</strong></td>
<td><strong>TIMESTAMP</strong></td>
<td></td>
</tr>
<tr>
<td><strong>KEYWORD</strong></td>
<td><strong>VARCHAR</strong></td>
<td>(254)</td>
</tr>
<tr>
<td><strong>OPTCOUNTER</strong></td>
<td><strong>SMALLINT</strong></td>
<td></td>
</tr>
<tr>
<td><strong>UP_NAME</strong></td>
<td><strong>CHAR</strong></td>
<td>(128)</td>
</tr>
</tbody>
</table>

**ProductDesc Table**

**A.10.2. SQL Statement**

```sql
DROP TABLE IF EXISTS `estore`.`productdesc`;
CREATE TABLE  `estore`.`productdesc` (
    `PRODUCT_ID` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `LANGUAGE_ID` varchar(45) NOT NULL,
    `NAME` varchar(128) NOT NULL,
    `SHORTDESCRIPTION` varchar(254) NOT NULL,
    `LONGDESCRIPTION` varchar(400) NOT NULL,
    `THUMBNAIL` varchar(254) NOT NULL,
    `FULLIMAGE` varchar(254) NOT NULL,
    `KEYWORD` varchar(254) NOT NULL,
    PRIMARY KEY (`PRODUCT_ID`),
    CONSTRAINT `FK_productdesc_1` FOREIGN KEY (`PRODUCT_ID`) REFERENCES `product` (`PRODUCT_ID`) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```
APPENDIX B

B.1. Abbreviation

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>JVM</td>
<td>A Java virtual machine (JVM) is an abstract computing machine that enables a computer to run a Java program</td>
</tr>
<tr>
<td>DBMS</td>
<td>A database-management system (DBMS) is a computer-software application that interacts with end-users, other applications, and the database itself to capture and analyze data. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases.</td>
</tr>
<tr>
<td>CPU</td>
<td>is the abbreviation for central processing unit. Sometimes referred to simply as the central processor, but more commonly called processor, the CPU is the brains of the computer where most calculations take place.</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial intelligence (AI, also machine intelligence, MI) is intelligent behavior by machines, rather than the natural intelligence (NI) of humans and other animals.</td>
</tr>
<tr>
<td>FCFS</td>
<td>First-come, first-served (FCFS), this principle states that customers are served one at a time and that the customer that has been waiting the longest is served first.</td>
</tr>
<tr>
<td>ANN</td>
<td>An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one neuron to the input of another.</td>
</tr>
<tr>
<td>CPA</td>
<td>Cognitive Performance Application</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity (JDBC) is an application-programming interface (API) for the programming language Java, which defines how a client may access a database. It is Java based data access technology and used for Java database connectivity. It is part of the Java Standard Edition platform, from Oracle Corporation.</td>
</tr>
<tr>
<td><strong>SQL</strong></td>
<td>a standard language for storing, manipulating and retrieving data in databases</td>
</tr>
<tr>
<td><strong>UML</strong></td>
<td>The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system</td>
</tr>
</tbody>
</table>
CURRICULUM VITA

Name: Shade El-Hadik

Address:

• 1618 Grayson Lakes Blvd.
  Katy, TX. 77494

Education:

• B.S. Computer Science
  American University in Cairo
  1999-2003
• M.S. Computer Engineering and Computer Science
  University of Louisville
  2004-2004

Work Experience:

• Accenture
  o IBM Watson eCommerce Architect       2017-Present
• IBM
  o IBM eCommerce Architect                2015-2017
• Accenture
  o IBM eCommerce Architect                2012-2015
• Academy Outdoors & Sports
  o IBM eCommerce Architect                2011-2012
• IBM
  o IT Architect                            2007-2011