## Welcome to Oracle haCpp!

Tag your social activity and music requests with #haCpp

Here are the house rules and we appreciate your cooperation.

1) Please do not to plug in your device/laptops to the Oracle network port

2 ) Please co operate with the security personnel and produce the invite letter at the oracle Main entry gate when asked by Security.

3) Please refrain from venturing into other floors of the building apart from where the event is organized.

4) Please handle office equipments/assets with utmost responsibility (vending machines, water dispensers, chairs/tables etc.)

5) Please refrain from getting into any altercations with the RE&F team on site. For any concerns, please out to the Oracle rep.

6) Please use the Office amenities responsibly. Avoid washing hands/legs etc in the restrooms.

7) In case of any emergency, please reach out to the Oracle representative.

**SAMPLE DATA (to all 4 Problem Statements)**

SampleData.zip

# Problem Statements (Choose only two)

# CHALLENGE 1

## PROBLEM STATEMENT

Imagine two set of tables in a Database. The tables are named - Table_Delivery and Table_Customer. The structure of both the tables is same and looks like this –

| Key 1 | Key 2 | Key 3 | Field 1 | Field 2 | Field 3 | Field 4 | Field 5 …. |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

1. There might any number of key fields and normal fields for the table. The set of Key fields together form a unique constraint for a row in the table.
2. Both the key fields and other fields can be of any of the following type – "Number", "VarChar" and "Date"
3. The number of rows in the table is also not limited.

There is an Application that populates data into these tables. The Application instance on the delivery side populates the "Table_Delivery" whereas the one on the customer side populates the "Table_Customer". Additionally, everyday there is a copy process which copies the data from the "Delivery" table to the "Customer" table.

The copy process has the following properties –

1. Copy process first compares the data between the two tables.
2. If a row of data from the "Delivery" is not present on the "Customer", it is copied directly.
3. If a row of data from the "Delivery" is present on the "Customer" (identified by Key values) -
    a. If the field value is changed in the "Customer" then it is preserved.

    b. If the field value is changed in the "Delivery" then it is copied.
4. If a row of data is present only on the "Customer" side, it is preserved.

Using these tables, design a framework for the above mentioned "Copy" process which should work for any kind of "Delivery" and "Customer" tables over multiple iterations of the Copy Process.

The expected output of this challenge is as follows –

1. A process simulating data population into the two tables. It is not necessary to have a UI to populate the tables. You could use file input as well (file with comma separated values).
2. Simulation of the compare process and displaying the results (at least 2 iterations).
3. Simulation of the copy process (at least 2 iterations).
4. A test client for testing all these aspects of the problem. We will provide some test data files at the end to check the correctness of the solution.
5. Good to have **(Brownie Points)** – If you can develop the compare UI in such a way that the user can pick and choose which fields to copy, that would be great.
6. A Technical Design document describing the following –
    a. Overall solution Architecture
    b. Object Model
    c. Sequence Diagrams
    d. Interaction Patterns

    (It will be nice if you can use UML notations for all of these. This technical design document should be used for final presentations.)

You can develop a library or an executable for this challenge.

**Example**

1. **Table_Delivery Initial Data**

| Key1 | Key2 | Key3 | Field1 | Field2 | Field3 | Field4 |
|---|---|---|---|---|---|---|
| A | B | C | 10 | Test | 1/1/1900 | Test1 |
| A1 | B1 | C1 | 20 | Test2 | 10/10/2000 | Test3 |
| A2 | B2 | C2 | 30 | Test4 | 11/11/2001 | Test6 |

2. **Data populated into the Table_Customer. So, Table_Customer initially looks like Table_Delivery.**

3. **Table_Delivery is altered (New rows are added and Existing rows are updated)**

| Key1 | Key2 | Key3 | Field1 | Field2 | Field3 | Field4 |
|---|---|---|---|---|---|---|
| A | B | C | 10 | Test | 1/1/1900 | Test45 |
| A1 | B1 | C1 | 60 | Test2 | 1/2/2003 | Test3 |
| A2 | B2 | C2 | 30 | Test21 | 11/11/2001 | Test6 |
| A3 | B3 | C3 | 40 | Test10 | 12/1/2004 | Test67 |

4. **Table_Customer is also altered.**

| Key1 | Key2 | Key3 | Field1 | Field2 | Field3 | Field4 |
|---|---|---|---|---|---|---|
| A | B | C | 10 | Test12345 | 9/9/2005 | Test1 |
| A1 | B1 | C1 | 80 | Test2 | 10/10/2000 | Test3 |
| A2 | B2 | C2 | 30 | Test4567 | 9/8/2013 | Test6 |
| A5 | B5 | C5 | 40 | Test56 | 9/8/2012 | Test88 |

5. **At the compare time, differences are shown like this –**

   **Row 1 – Field2, Field3 and Field4 are altered**

   **Row2 – Field1 (modified in both tables) and Field3 modified**

   **….**

   **Row4 – new by Table_Delivery**

   **Row5 – new by Table_Customer**

6. **After the copy and applying the rules, the Table_Customer looks like this –**

| Key1 | Key2 | Key3 | Field1 | Field2 | Field3 | Field4 |
|---|---|---|---|---|---|---|
| A | B | C | 10 | Test12345 | 9/9/2005 | Test45 |
| A1 | B1 | C1 | 80 | Test2 | 1/2/2003 | Test3 |
| A2 | B2 | C2 | 30 | Test4567 | 9/8/2013 | Test6 |
| A5 | B5 | C5 | 40 | Test56 | 9/8/2012 | Test88 |
| A3 | B3 | C3 | 40 | Test10 | 12/1/2004 | Test67 |

**Fields highlighted in yellow are changed because only Table_Delivery had changes to these fields. Fields highlighted in green are preserved because Table_Customer had changes to those fields.**

# Technology

1. C++ for actual business logic
2. You would probably require Database access. If you don't have that, you can use flat files to store and display data.
3. You would also probably need a library to build your UI (MFC or equivalent). You can also develop the UI as a web application.

# Judging Criteria

## Judging Weightage

1. **Choice of Architecture – 10%**
2. **User Interface – If they have a compare UI – 5%**
3. **Efficiency/Performance – for huge amount of data – 20%**
4. **Design Patterns being used – 5%**
5. **Completeness of the solution – 30%**
6. **Scaling up of the solution (based on different inputs) – 10%**
7. **Error Handling (Invalid Inputs, parameters, Buffer checks, Invalid File Format etc.) – 10%**
8. **Profiling Tool – 10%**

# CHALLENGE 2

## PROBLEM STATEMENT

Elastic Search is an open source end-to-end search and analytics platform (www.elastic.co)
Download the elastic search from here - https://www.elastic.co/downloads/elasticsearch

API Information:

https://www.elastic.co/guide/en/elasticsearch/reference/current/docs.html

The challenge is to build a Framework which utilizes the Elastic Search to index historical Sensex data. Once the data is indexed, build a small Application on top of this Framework which will perform a search based on a "Query String" and show results to the user. Your solution should

be scalable. We should be able to test your data with 1, 5, 10 and 15 years of Sensex data. We will provide some sample data for you to build and code the application. The final testing will be done based on the test data that we provide to test the performance of the application.

The expected output of this challenge is as follows –

1. An application that is able to index and query sensex data using elastic search.
2. A configurable and scalable elastic search installation on your laptop.
3. Good to have **(Brownie Points)** – If you can develop a small web application on top of the actual search to take search inputs and display the results.
4. A Technical Design document describing the following –
    a. Overall solution Architecture
    b. Object Model
    c. Sequence Diagrams
    d. Interaction Patterns
    e. How have the elastic search APIs been used?

    (It will be nice if you can use UML notations for all of these. This technical design document should be used for final presentations.)

You can develop a library or an executable for this challenge.

## Technology

1. C++ for actual business logic
2. You can use the JSON APIs that elastic search provides to index and query the data.
3. If you are building a web application on top, then you might need to use HTML, CSS, JAVASCRIPT, JQUERY etc. for the front end. A web server to process the web client requests and JNI for JAVA to C++ conversion and vice-versa.

## Judging Criteria

1. **Architecture and Design – 20%**
2. **Usage of Elastic Search APIs – 15%**
3. **Efficiency/Performance – 15%**

4. **Completeness of the solution – 30%**
5. **User Interface – 10%**
6. **Profiling Tool – 10%**

# CHALLENGE 3

**PROBLEM STATEMENT**

**Star gazer**

You are an apprentice at a premier Astronomical society. Your job is to work with an extremely efficient automated astronomer system. This system detects millions of new stars and planets in a day (24 hours) via the James Webb and Hubble telescopes and feeds them to you with a distance with respect to Earth along with other details. You need to build a very efficient and fast software that outputs the distance between any two stars.

You need to keep in mind that you are getting data for millions of stars every new day. And you would need to calculate the distances between the existing set and new set of stars daily. So over the period of time the data accumulated is going to be huge. But the design of your software should be able to withstand that. I.e. no reduction in the performance.

You have resource constraints too. You are only allowed to use 200 MB of memory. Any more and your process will be killed by a ruthless system monitor process. But on Sundays you are granted 100% of the resources.

For simplicity we shall say that the 3D spatial co-ordinates (X,Y,Z) of the stars are made available to you. The distances X,Y,Z are relative to the Earth on the respective axis. Given the 3D spatial co-ordinates for any two stars ($X_1$, $Y_1$, $Z_1$) & ($X_2$, $Y_2$, $Z_2$) we give the below the formula to compute the distance between these two stars.

$$dX = X_2 - X_1$$
$$dY = Y_2 - Y_1$$
$$dZ = Z_2 - Z_1$$

$$D = \sqrt{(dX)^2 + (dY)^2 + (dZ)^2}$$

You are the main engineer of the Starship Enterprise. You have been asked to work with Spock to chart a course from one star to another in the galaxy.

You would need to stop at some stars on the way to refuel after every 10 light years, as the fuel in your space ship does not last beyond 10 light years.

You also need to avoid stars with hostile civilizations on planets circling it.

You can have a file system storing the following data for each star.

| Star ID | X | Y | Z | CAN_REFUEL | IS_HOSTILE |
|---|---|---|---|---|---|
| | | | | | |

The CAN_REFUEL column can be  Y / N value based on whether the star has the gas to refuel your spaceship. The IS_HOSTILE column can be Y / N value based on whether the star is considered Hostile to visit because of the planets encircling it.

Given any source and destination stars as input, you will need to find an optimum path to traverse from source to destination stars considering all the above constraints.

## Technology

This is a pure C++ problem and does not require anything else to code the business logic. You can use files for storage and retrieval.

## Judging Criteria

1. **Architecture and Design – 20%**
2. **Error Handling (For incorrect inputs, wrong parameters etc.) – 10%**
3. **Efficiency/Performance – 20%**
4. **Completeness of the solution – 30%**
5. **Constraint Handling – 10%**
6. **Profiling Tool – 10%**

# CHALLENGE 4

**PROBLEM STATEMENT**

Pivot Tables in Excel are used to display data in a multidimensional format. The data in the Pivot table shows data based on different dimensions and facts at different levels. An example is shown below –

| Month | Region | Sum of Unit Cost | Sum of Sales | Sum of Product Sales |
|---|---|---|---|---|
| 01/01/2004 | EAST_COAST | 4270.11 | 1982 | 681894.67 |
| | HAWAII | 4270.11 | 4015 | 1354178.92 |
| | MIDWEST | 4270.11 | 815 | 827281.86 |
| | WEST_COAST | 4270.11 | 2877 | 775259.1 |
| 02/01/2004 | EAST_COAST | 4183.84 | 1982 | 668119.02 |
| | HAWAII | 4183.84 | 4015 | 1326821.77 |
| | MIDWEST | 4183.84 | 815 | 810569.1 |
| | WEST_COAST | 4183.84 | 2877 | 759597.3 |
| 03/01/2004 | EAST_COAST | 4173.73 | 1982 | 666505.18 |
| | HAWAII | 4173.73 | 4015 | 1323616.87 |
| | MIDWEST | 4173.73 | 815 | 808611.18 |
| | WEST_COAST | 4173.73 | 2877 | 757762.5 |
| 04/01/2004 | EAST_COAST | 3925.05 | 1982 | 626792.07 |
| | HAWAII | 4701.43 | 4015 | 1490964.67 |
| | MIDWEST | 3925.05 | 815 | 760430.8 |
| | WEST_COAST | 4172.82 | 2877 | 756127.93 |
| 05/01/2004 | EAST_COAST | 4701.43 | 1982 | 750772.92 |
| | HAWAII | 4356.37 | 4015 | 1381536.07 |
| | MIDWEST | 4529.21 | 815 | 859553.91 |
| | WEST_COAST | 4701.43 | 2877 | 853568.1 |
| 06/01/2004 | EAST_COAST | 4356.37 | 1982 | 695670.32 |
| | HAWAII | 4356.37 | 4015 | 1381536.07 |
| | MIDWEST | 4356.37 | 815 | 843994.63 |

| | | | | |
|---|---|---|---|---|
| | WEST_COAST | 4356.37 | 2877 | 790920.9 |
| 07/04/2004 | EAST_COAST | 4313.25 | 1982 | 688782.5 |
| | HAWAII | 4313.25 | 4015 | 1367857.5 |
| | MIDWEST | 4313.25 | 815 | 835638.25 |
| | WEST_COAST | 4313.25 | 2877 | 783090 |

In this example, the fields Month and Region are on the Row Axis of the Pivot Table and the Facts "Unit Cost", "Sales" and "Product Sales" are on the Column Axis. This is just an example of the layout. It is possible to have the fields being plotted on any of the axis e.g. the Month could have been plotted on the Column Axis.

The data for the n-dimensional pivot table comes from a two dimensional grid e.g. for the above mentioned example the data for the pivot table comes from the attached file "TestData.xls". There are two inputs for creating the pivot table – the two dimensional grid data and the pivot table layout i.e. which field lies on which axis (Row or Column). Also, the dimensional fields can be of any type – strings, numbers or dates but the Facts are always numeric in nature.

The challenge is two-fold here –

1. Implement –
    a. Data structures for representing data in the pivot table.
    b. Given a two dimensional grid and the layout, construct the pivot table and display it

       **Note** – The input for the two dimensional grid can be from a flat file or in any other fashion that you desire. That is not the core part of the challenge.
2. Implement a sorting algorithm for such a Pivot Table. There are three inputs which the Sort Function will take – the Dimension field on which the Sorting should be done, the Fact value on which the sort should happen and the Sort order. As an example, in the above mentioned example, the input could be to sort the data by Month Field, Product Sales and ascending order. Then the data would be sorted and shown for the lowest to highest Product Sales values for the Month Field.
3. Additional Challenge (Extra item to earn **brownie points**) – In addition to the n-Dimensional layout if you can compute row and column totals, it will be great e.g. for the above mentioned example compute the total values (highlighted rows) –

| **Row Labels** | **Sum of Unit Cost** | **Sum of Sales** | **Sum of Prd Sales** |
|---|---|---|---|
| **1/1/2004** | **17080.44** | **9689** | **3638614.55** |

| | | | |
|---|---|---|---|
| EAST_COAST | 4270.11 | 1982 | 681894.67 |
| HAWAII | 4270.11 | 4015 | 1354178.92 |
| MIDWEST | 4270.11 | 815 | 827281.86 |
| WEST_COAST | 4270.11 | 2877 | 775259.1 |
| **2/1/2004** | **16735.36** | **9689** | **3565107.19** |
| EAST_COAST | 4183.84 | 1982 | 668119.02 |
| HAWAII | 4183.84 | 4015 | 1326821.77 |
| MIDWEST | 4183.84 | 815 | 810569.1 |
| WEST_COAST | 4183.84 | 2877 | 759597.3 |
| **3/1/2004** | **16694.92** | **9689** | **3556495.73** |
| EAST_COAST | 4173.73 | 1982 | 666505.18 |
| HAWAII | 4173.73 | 4015 | 1323616.87 |
| MIDWEST | 4173.73 | 815 | 808611.18 |
| WEST_COAST | 4173.73 | 2877 | 757762.5 |
| **4/1/2004** | **16724.35** | **9689** | **3634315.47** |
| EAST_COAST | 3925.05 | 1982 | 626792.07 |
| HAWAII | 4701.43 | 4015 | 1490964.67 |
| MIDWEST | 3925.05 | 815 | 760430.8 |
| WEST_COAST | 4172.82 | 2877 | 756127.93 |
| **5/1/2004** | **18288.44** | **9689** | **3845431** |
| EAST_COAST | 4701.43 | 1982 | 750772.92 |
| HAWAII | 4356.37 | 4015 | 1381536.07 |
| MIDWEST | 4529.21 | 815 | 859553.91 |
| WEST_COAST | 4701.43 | 2877 | 853568.1 |
| **6/1/2004** | **17425.48** | **9689** | **3712121.92** |
| EAST_COAST | 4356.37 | 1982 | 695670.32 |
| HAWAII | 4356.37 | 4015 | 1381536.07 |
| MIDWEST | 4356.37 | 815 | 843994.63 |
| WEST_COAST | 4356.37 | 2877 | 790920.9 |
| **7/4/2004** | **17253** | **9689** | **3675368.25** |
| EAST_COAST | 4313.25 | 1982 | 688782.5 |
| HAWAII | 4313.25 | 4015 | 1367857.5 |
| MIDWEST | 4313.25 | 815 | 835638.25 |
| WEST_COAST | 4313.25 | 2877 | 783090 |
| **Grand Total** | **120201.99** | **67823** | **25627454.11** |

The end product would also have three items. The first item would be a library which exposes two functions –

1. The first function will create the pivot table. It will take two inputs – the two dimensional data (can be in a file format) and the layout. The output of the function can also be a file

displaying the pivot table. The input will also specify which of the fields in the two dimensional data are dimensions and which one of them are Facts.

**Note**:

a. In the solution mention the format of the file that the function can take. An example file would also help. An example file which can be used for the input format is attached at the end (TestData.csv)

b. Also, handle all the error scenarios like invalid file format, incorrect layout etc.

2. The second function will do the sorting.

The second item would be a test client which loads the library, takes inputs and performs the different functions.

We would also like a Technical Design document describing the object model, sequence diagrams, interactions etc. (UML notations preferred)

# Technology

We love C++ so the solution should be purely in C++ and preferably on Windows. You are free to use any design patterns, Templates, STL etc. Actually, the more these paradigms are used, more are the number of brownie points that you get.

# Judging Criteria

1. **Architecture & Object Design – 20%**
2. **Efficiency/Performance – 10%**
3. **Design Patterns being used – 5%**
4. **Completeness of the solution – 30%**
5. **Extra part of the challenge – 10%**
6. **Efficient Sorting algorithm – 5%**
7. **Error Handling – 10%**
8. **Code Profiling – 10%**

# Submission Guidelines

Direct your projects to submissions@venturesity.com

**We can't wait to see what you create!**

---

VH Education Services Pvt. Ltd.

Ground Floor, No. 46, 11th Cross

Indiranagar 1st Stage

Bangalore - 560038

IN: +91 959 009 1584

US: +1 202 558 2075

Support: help@venturesity.com