**Please answer the following 2 technical questions:**

**Question #1 (SQL)**

We work with large datasets,and are always performance conscious since extended processing times will impact our time to market. Keep this in mind as you answer the following questions:

There is a table defined as:

CREATE TABLE [Positions](

            [load\_id]           [int]                  NOT NULL,

            [acct\_cd]         [varchar](20)   NOT NULL,

            [acct\_num]      [varchar](255)             NULL,

            [sec\_id]            [varchar](50)   NOT NULL,

            [long\_sht\_cd]   [varchar](3)     NOT NULL,

            [sedol]              [varchar](15)   NULL,

            [isin]                 [varchar](15)   NULL,

            [cusip]              [varchar](9)     NULL,

            [sec\_type]       [varchar](8)     NULL,

            [sec\_name]     [varchar](100) NULL,

            [currency\_cd] [varchar](3)     NULL,

            [total\_holding] [decimal](18, 4) NULL,

            [mkt\_price]      [float]               NULL,

            [datetime\_stamp] [datetime]   NULL,

CONSTRAINT [pk\_Positions] PRIMARY KEY CLUSTERED (

[load\_id] ASC,

            [acct\_cd] ASC,

            [sec\_id] ASC,

            [long\_sht\_cd] ASC )

)

This table holds account positions data that are appended to multiple times a day

There are currently some 24 million rows in the table. Every time we append additional positions we add approximately 32,000 entries to this table, and all 32,000 entries will have the same load\_id. The load\_id is incremented by one each time we load a batch of 32,000 entries (i.e. the first 32K entries have load\_id=1, the next 32K has load\_id=2, etc...). The datetime\_stamp field shows the time at which the entries were loaded and is the same for all 32K entries in a single load.

**How would you efficiently retrieve the first set of positions for the current day given the above table definition?**

Example:

Today, positions were loaded into this table at 8am, 10am and 3pm. At 5pm today we want to know what positions were loaded at 8am since that is the first load that occurred today. Note that for any given day, there can be different number of loads and the times that the loads occur will vary.

**Answer Question #1 Here:**

The following MySQL query efficiently retrieves the first set of positions for the current day.

Steps:

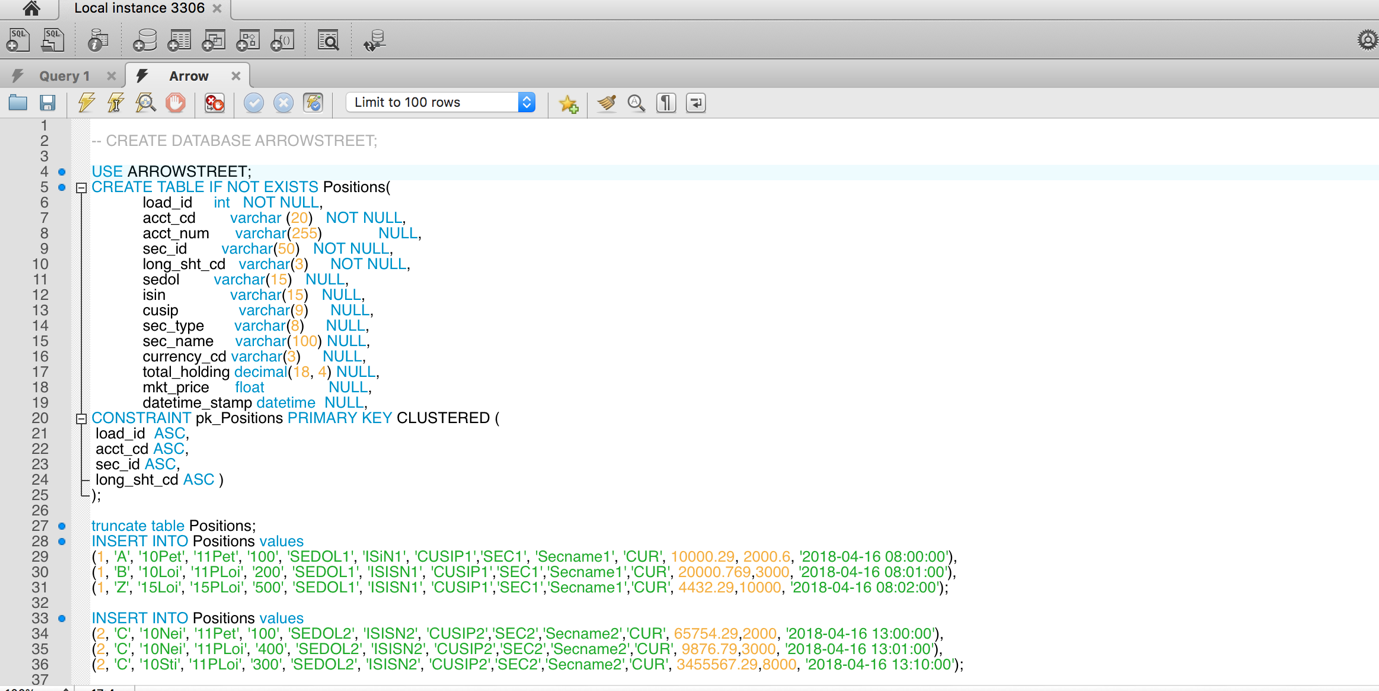
1. First find the load\_id for the earliest load of the current day.
2. Get all rows for that particular load\_id

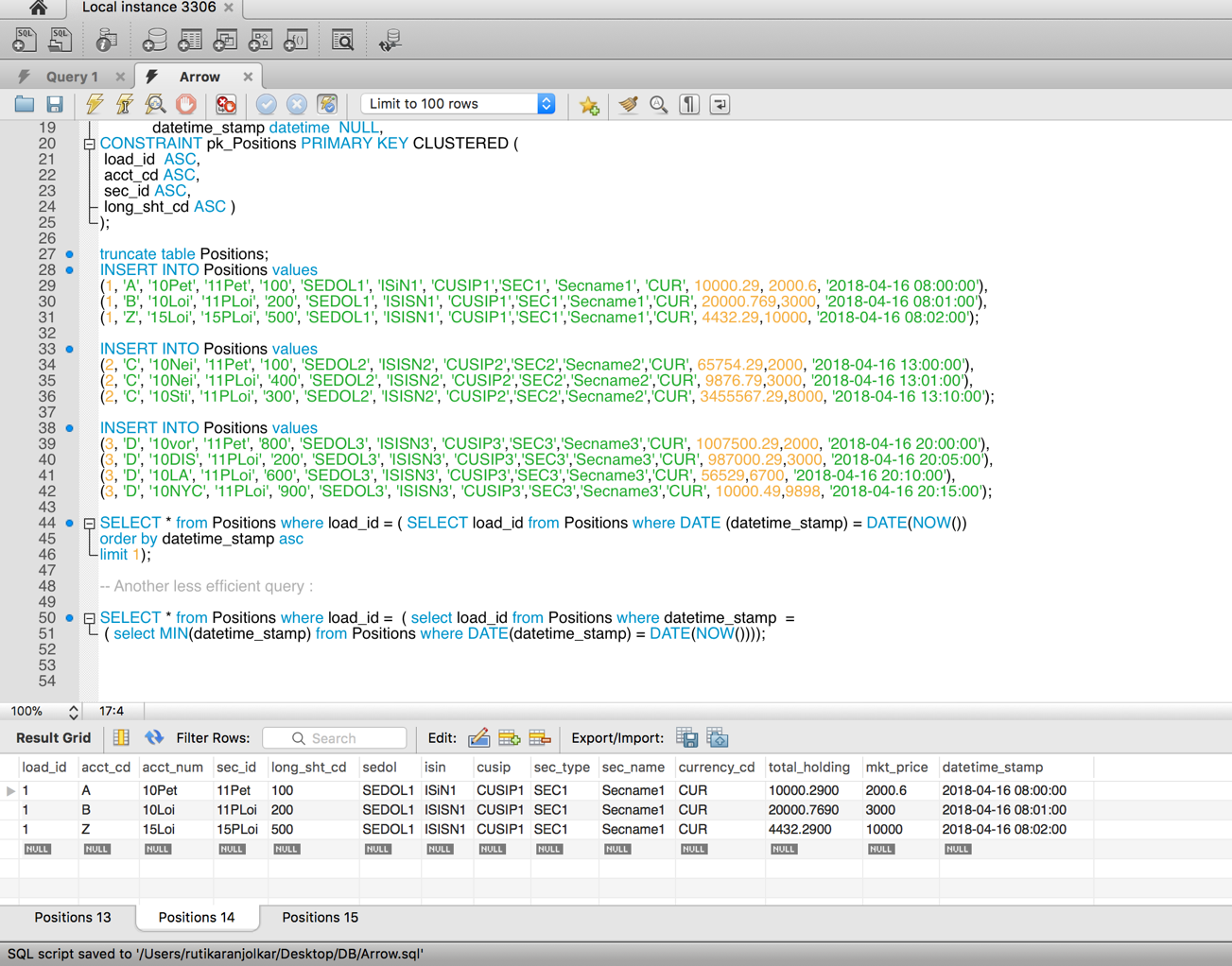
SELECT \* from Positions where load\_id = ( SELECT load\_id FROM Positions WHERE DATE (datetime\_stamp) = DATE(NOW())

ORDER BY datetime\_stamp ASC

LIMIT 1);

**Approach**: Used MySQL to replicate the challenge and find the optimum solution





**Question #2 (Analyzing C# /Code)**

Given the table [tSWIFT\_History\_Trades] , the C# structure TradeNAK and the C# method UpdateTradeACKNAKDB as defined below:

CREATE TABLE [tSWIFT\_History\_Trades](

[Id] [bigint] IDENTITY(1,1) NOT NULL,

[BatchId] [varchar](25) NOT NULL,

[TradeId] [varchar](25) NOT NULL,

[Account] [varchar](25) NULL,

[TranType] [varchar](50) NULL,

[TradeDate] [int] NULL,

[SettleDate] [int] NULL,

[Exch] [varchar](50) NULL,

[DateSent] [datetime] NOT NULL

CONSTRAINT [DF\_tSWIFT\_History\_Trades\_Sent\_DateSent] DEFAULT (getdate()),

[Status] [varchar](20) NOT NULL,

[AckNak] [varchar](7) NULL

CONSTRAINT [DF\_tSWIFT\_History\_Trades\_AckNak] DEFAULT ('Pending'),

[MachineIdReceived] [varchar](50) NULL,

[FileNameReceived] [varchar](100) NULL,

[DateReceived] [datetime] NULL,

[DirectoryArchivedReceived] [varchar](300) NULL,

[Error] [varchar](500) NULL,

[Price] [float] NULL CONSTRAINT [DF\_tSWIFT\_History\_Trades\_Price] DEFAULT (0),

CONSTRAINT [PK\_tSWIFT\_History\_Trades] PRIMARY KEY NONCLUSTERED ([Id] ASC )

GO

select count (\*) from dbo.tSWIFT\_History\_Trades

--returns 568655

----------------------------------------------------------------

public struct TradeNAK

{ public string BatchId;

public string TradeId;

public bool AckNakSuccess;

public string ErrorText; }

--The number of trade acknowledgements (TradeNAKs) per batch (i.e. \_lANT) can reach 2000.

public int UpdateTradeACKNAKDB(List<TradeNAK> \_lANT, string strFileNameReceived, string strDirectoryArchivedReceived)

{

try

{

int j = 0;

for (int i = 0; i < \_lANT.Count; i++)

{

if (\_lANT[i].AckNakSuccess == true)

{

if (\_lANT[i].ErrorText == null)

{

// success = true & error = null

j += Convert.ToInt16(dba.ExecuteSQL("UPDATE tSWIFT\_History\_Trades SET MachineIdReceived = '" + System.Environment.UserName.ToUpper() + "', FileNameReceived = '" + strFileNameReceived + "', DirectoryArchivedReceived = '" + strDirectoryArchivedReceived + "', DateReceived = GetDate(), AckNak = 'ACK' WHERE (BatchId = '" + \_lANT[i].BatchId + "' AND TradeId = '" + \_lANT[i].TradeId + "')"));

}

else

{

// success = true & error = not null

j += Convert.ToInt16(dba.ExecuteSQL("UPDATE tSWIFT\_History\_Trades SET MachineIdReceived = '" + System.Environment.UserName.ToUpper() + "', FileNameReceived = '" + strFileNameReceived + "', DirectoryArchivedReceived = '" + strDirectoryArchivedReceived + "', DateReceived = GetDate(), AckNak = 'ACK', [Error] = '" + \_lANT[i].ErrorText.Replace("'", "''") + "' WHERE (BatchId = '" + \_lANT[i].BatchId + "' AND TradeId = '" + \_lANT[i].TradeId + "')"));

}

}

else

{

if (\_lANT[i].ErrorText == null) // success = false & error = null

{

j += Convert.ToInt16(dba.ExecuteSQL("UPDATE tSWIFT\_History\_Trades SET MachineIdReceived = '" + System.Environment.UserName.ToUpper() + "', FileNameReceived = '" + strFileNameReceived + "', DirectoryArchivedReceived = '" + strDirectoryArchivedReceived + "', DateReceived = GetDate(), AckNak = 'NAK' WHERE (BatchId = '" + \_lANT[i].BatchId + "' AND TradeId = '" + \_lANT[i].TradeId + "')"));

}

else

{ // success = false & error = not null

j += Convert.ToInt16(dba.ExecuteSQL("UPDATE tSWIFT\_History\_Trades SET MachineIdReceived = '" + System.Environment.UserName.ToUpper() + "', FileNameReceived = '" + strFileNameReceived + "', DirectoryArchivedReceived = '" + strDirectoryArchivedReceived + "', DateReceived = GetDate(), AckNak = 'NAK', [Error] = '" + \_lANT[i].ErrorText.Replace("'", "''") + "' WHERE (BatchId = '" + \_lANT[i].BatchId + "' AND TradeId = '" + \_lANT[i].TradeId + "')"));

}

}

}

return j;

}

catch (Exception ex)

{

throw ex;

}

}

**First briefly describe this method does?**

1. **Describe how the method can be changed, improved or written better.**
2. **Include an explanation for how and why your solution is better.**
3. **Rewrite UpdateTradeACKNAKDB function**

**Question #2 Answers Here:**

The method iterates over a list of TradeNAK objects and updates the columns of tSWIFT\_History\_Trades. The values of the updated columns depend on the following logic:

|  |  |  |  |
| --- | --- | --- | --- |
| lANT[i].AckNakSuccess | lANT[i].ErrorText | AckNak | Error |
| Success = True | Null | ACK |  |
| Success = True | Not Null | ACK | Update error in DB |
| Success = False | Null | NAK |  |
| Success = False | Not Null | NAK | Update error in DB |

The total number of successful updates is returned by the function.

1. **The method can be improved in the following manner:**
   1. **Remove code repetition:** The SQL statement can be extracted into a String just once, instead of defining it 4 times. A further improvement can be to store the SQL statement in a properties file.
   2. **Naming conventions**: Instead of using variable names such as ‘j’, use meaningful names such as ‘total updates’ that make the code readable.
   3. **Parameterize the SQL Statement**: To avoid SQL injection, the ‘dba.ExecuteSQL’ function must be parameterized.
   4. **Batch SQL updates for performance**: Ideally, we should have dba.ExecuteBatchSQL function, that will take a list of queries and run them as a batch. This will reduce the roundtrip time for the SQL statement to reach the Database.
2. **How and why my solution is better:**
   1. **Remove code repetition**: The code is cleaner, more readable and less time consuming to write when the code is not repeated. This also reduces the chance of error caused by refactoring, since there is less code to write and maintain.
   2. **Naming conventions:** Providing meaningful names to the variables enhances code readability, especially when the code is read by other developers.
   3. **Parameterize the SQL Statement**: Without parameterization, SQL statements are vulnerable to SQL injection attacks. This is a technique frequently used by hackers to get into the system. Parameterization also removes the need to escape single quotes.
   4. **Batch SQL updates for performance:** Batch updates can be used to execute repetitive SQL statements together to save time and thus, make the process more efficient.
3. **Rewriting UpdateTradeACKNAKDB method:**

**Approach:**

* 1. The SQL statement that was repeated 4 times in the previous version of the **UpdateTradeACKNAKDB** method has been extracted as a parameterized statement called ‘updateStatement’.
  2. In the previous version, the ‘Error’ column of the table tSWIFT\_History\_Trades was not being updated when the value of ErrorText was Null. The last updated value of Error was still displayed in the column. In the updated version of **UpdateTradeACKNAKDB,** the value of Error column becomes null, if the value of ErrorText is null.
  3. The multiple if-else statements are substituted by simple ternary operators and the SQL statement is executed based on the result of ErrorText and AckNakVal
  4. The individual values of the parameters is extracted by using updateCommand.Parameters.AddWithValue
  5. The variable ‘j’ is replaced with ‘totalUpdates’ that gives a context of what the variable does.

**The modified method is as follows:**

public int UpdateTradeACKNAKDB(List < TradeNAK > \_lANT, string strFileNameReceived, string strDirectoryArchivedReceived)

{

// The 'using' ensures that the connection object is closed safely at the end

using(SqlConnection connection = getConnection()) {

// getConnection is a function defined elsewhere

// The SQL text can be defined in an external file as well

string updateStatement = "UPDATE tSWIFT\_History\_Trades SET MachineIdReceived =

@username, FileNameReceived = @filename, DirectoryArchivedReceived = @directoryName, DateReceived = GetDate(), AckNak = @acknakVal WHERE (BatchId = @batchId AND TradeId = @tradeId)";

try

{

int totalUpdates = 0;

for (int i = 0; i < \_lANT.Count; i++)

{

// Decide what the values should be

string acknakVal = \_lANT[i].AckNakSuccess == true? 'ACK': 'NAK';

string errorText = \_lANT[i].ErrorText;

// We don't need to replace single quotes since we are using parameterized SQL

// Create an SqlCommand and add parameters for it

SqlCommand updateCommand = new SqlCommand(updateStatement, connection);

updateCommand.Parameters.AddWithValue("@acknakVal", acknakVal);

updateCommand.Parameters.AddWithValue("@username", System.Environment.UserName.ToUpper());

updateCommand.Parameters.AddWithValue("@filename", strFileNameReceived);

updateCommand.Parameters.AddWithValue("@directoryName", strDirectoryArchivedReceived);

updateCommand.Parameters.AddWithValue("@batchId", \_lANT[i].BatchId);

updateCommand.Parameters.AddWithValue("@tradeId", \_lANT[i].TradeId);

// Execute SQL and increase the total updates count

totalUpdates += updateCommand.ExecuteNonQuery();

}

return totalUpdates;

}

catch (Exception ex)

{

throw ex;

}

}

}

I also worked out the challenge using Java and MySQL JDBC connection to update the table using the conditions. Github link of the code:

