Advanced Database Systems

Report of Tuning Project

By: Pratik Daga

Student ID: N18669576

DEDICATED TO MY FAMILY, SYLPHY AND JACKY

Index

Server Information .................................................................................................................. 2

Case 1: Remove cursor from procedure
  • Procedure which calculate & store date for last trading date (30 times Improvement)......... 4
  • Procedure which calculate and store data for all the trading dates (10 times) ...................... 7

Case 2: Logical Tuning
  • Avoiding Dead Index during Peak Hours (Performance varies) ........................................ 13
  • Debugging and other modifications .................................................................................. 16

Case 3: Correlated Sub Queries (logical reads) .................................................................. 18

Case 4: Reducing Complicated Sub Queries into simpler (from 6 min to 3 sec) ...................... 21

Appendix: DDL statement of tables and views .................................................................... 25
**Server Information**

<table>
<thead>
<tr>
<th>Common</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Network name</td>
<td><strong>Investarsd6800</strong></td>
</tr>
<tr>
<td>(alias)</td>
<td></td>
</tr>
<tr>
<td>IP External</td>
<td><strong>XXX.XXX.XXX</strong></td>
</tr>
<tr>
<td>IP Internal</td>
<td><strong>XXX.XXX.XX</strong></td>
</tr>
<tr>
<td>CPU</td>
<td>Xeon MP, 3Ghz X 4, 2CORE, 8MB</td>
</tr>
<tr>
<td>Memory</td>
<td>32 Gb</td>
</tr>
<tr>
<td>Slot 8X PCIe</td>
<td>PERC 5/E Adapter</td>
</tr>
<tr>
<td></td>
<td>Connected to MD1000</td>
</tr>
</tbody>
</table>

**Disks**

<table>
<thead>
<tr>
<th>C:</th>
<th>36Gb X 2, RAID 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Internal Drives</td>
</tr>
<tr>
<td></td>
<td>Disk Size: 33GB</td>
</tr>
<tr>
<td></td>
<td>Read Policy: Adaptive Read Ahead</td>
</tr>
<tr>
<td></td>
<td>Write Policy: Write Back</td>
</tr>
<tr>
<td></td>
<td>Stripe Size: 16 KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E:</th>
<th>72Gb X 10, RAID 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Internal Drives</td>
</tr>
<tr>
<td></td>
<td>Disk Size: 272GB</td>
</tr>
<tr>
<td></td>
<td>Read Policy: Adaptive Read Ahead</td>
</tr>
<tr>
<td></td>
<td>Write Policy: Write Back</td>
</tr>
<tr>
<td></td>
<td>Stripe Size: 64 KB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>J:</th>
<th>136Gb X 15, RAID 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MD1000</td>
</tr>
<tr>
<td></td>
<td>Disk Size: 952.88GB</td>
</tr>
<tr>
<td></td>
<td>Read Policy: Adaptive Read Ahead</td>
</tr>
<tr>
<td></td>
<td>Write Policy: Write Back</td>
</tr>
<tr>
<td></td>
<td>Stripe Size: 64 KB</td>
</tr>
</tbody>
</table>
The database Server used is SQL SERVER 2005. The front end for the Website is ASP.net. Since it’s a financial Database, it has lot of procedures and most of the them run every day or as per client requirements.

**CASE 1:**

a) Removing Cursor from procedure which calculates and store data for latest trading date:-
There is no need to store calculation about all the trading dates. Simply, a last trade date calculation at the end of the cursor is needed.

```
Before:--

DECLARE cur_dates_1 CURSOR FOR
SELECT DISTINCT date - tradedate FROM @dates WHERE tradedate is not null
ORDER BY date
OPEN cur_dates_1
FETCH FROM cur_dates_1 INTO @date

WHILE @@FETCH_STATUS = 0
BEGIN
  PRINT 'rebalance date: ' + CONVERT(VARCHAR(10), @date, 1)
  IF @TotalReturn = 'Y'
  BEGIN
    UPDATE @positions SET
      Dividend = ISNULL(Dividend, 0) + (SELECT ISNULL(SUM(Dividend), 0) FROM @dividends as d WHERE d.StockID = p.StockID AND d.Date > @PrevDate AND d.Date <= @Date)
    FROM @positions as p
  END
  IF @RatingType = 'U'
  BEGIN
    DELETE FROM #ratings WHERE termid = 9
    SET @RatingType = NULL
  END

INSERT INTO @newPositions(stockid, rating, date)
SELECT DISTINCT ar.stockid, ar.rating, @date
FROM #ratings as ar
INNER JOIN (
  SELECT r.stockid, date = MAX(r.date)
FROM #ratings as r
```
INNER JOIN @portfolio_history as p ON p.date = (SELECT MAX(date) FROM @portfolio_history WHERE date <= @date AND p.stockid = r.stockid)
INNER JOIN stocks_local as s ON s.StockID = r.stockid
WHERE r.date <= @date AND s.FirstQuoteDate <= @date AND (s.LastQuoteDate >= @date OR s.DeadFlag = 'N')
AND ISNULL((SELECT TOP 1 Exchange FROM stock_exchange_history WHERE StockID = r.StockID AND Date <= @date ORDER BY Date DESC), s.Exchange) NOT IN (SELECT Exchange FROM #excluded_exchanges) GROUP BY r.stockid
) as l ON l.stockid = ar.stockid AND l.date = ar.date
WHERE ar.Rating = ISNULL(@RatingType, ar.Rating)

TRUNCATE TABLE #intra_dates
SELECT IntraDate = d.date, @MktCapValue, @EqualValue, @NumOfStocks
FROM trading_dates d
WHERE d.date > @PrevComponentDate AND d.Date > @StartDate AND d.Date < @Date AND d.Date NOT IN(SELECT Date FROM stock_holidays)

--select Date = @date, prevdate = @prevdate, PrevComponentDate = @PrevComponentDate, * from #intra_dates

TRUNCATE TABLE #prices
INSERT INTO #prices (IntraDate, StockID, MktCapShares, EqualShares, OldPrice, Price, MktCapValue, EqulValue)
SELECT d.intradate, c.stockid, c.MktCapShares, c.EqualShares,
oldprice - c.price, price = h.closeat, d.MktCapValue, d.EqualValue
FROM #intra_dates as d
INNER JOIN @positions c ON 1 = 1
LEFT JOIN stock_history_safe as h (nolock) ON h.stockid = c.stockid AND h.date = d.intradate

UPDATE #prices SET price = (SELECT TOP 1 closeat FROM stock_history_safe WHERE stockid = #prices.stockid
AND date <= intradate ORDER BY date DESC)
WHERE price IS NULL

INSERT INTO #performance_history(Date, MktCapValue, EqualValue, NumOfStocks, [Type])
SELECT distinct h.IntraDate,
MktCapValue = ISNULL(SUM(MktCapShares * Price), h.MktCapValue),
EqualValue = ISNULL(SUM(EqualShares * Price), h.EqualValue),
NumOfStocks, 'I'
FROM #intra_dates as h
LEFT JOIN #prices as p ON h.IntraDate = p.IntraDate
WHERE NOT EXISTS(SELECT * FROM #performance_history WHERE date = p.intradate)
GROUP BY h.IntraDate, h.NumOfStocks, h.MktCapValue, h.EqualValue

UPDATE @positions SET price = NULL, marketcap = NULL
UPDATE @positions set price = closeat, marketcap = h.marketcap
FROM @positions as c
INNER JOIN stock_history_safe as h ON h.stockid = c.stockid
WHERE h.date = @date

UPDATE @positions SET
price = (SELECT TOP 1 closeat FROM stock_history_safe WHERE stockid = s.stockid AND date <= @date ORDER BY date DESC),
marketcap = (SELECT TOP 1 marketcap FROM stock_history_safe WHERE stockid = s.stockid AND date <= @date AND marketcap IS NOT NULL ORDER BY date DESC)
FROM @positions as s
WHERE price IS NULL OR marketcap IS NULL

SELECT @PrevDate = @Date

FETCH FROM cur_dates_1 INTO @date
END
CLOSE cur_dates_1
DEALLOCATE cur_dates_1
After :--

select tradedate into #t_dates from @dates order by tradedate asc  // make one trading date table
called #t_dates

print 'rebalanceFrequencyID' + CONVERT(VARCHAR(20), @RebalanceFrequencyID)

IF @TotalReturn = 'Y'
BEGIN
    UPDATE @positions SET
    Dividend = ISNULL(Dividend, 0) + (SELECT ISNULL(SUM(Dividend), 0) FROM @dividends as d inner
    JOIN #t_dates td on 1=1
    FROM @positions as p
    END

IF @RatingType = 'U'
BEGIN
    DELETE FROM #ratings WHERE termid = 9
    SET @RatingType = NULL
END

-- grab the current positions

SELECT r.stockid, date - max(r.date) into #temp
FROM #ratings as r
INNER JOIN #stocks_local as s ON s.StockID = r.stockid
INNER JOIN #t_dates td on 1=1
INNER JOIN @portfolio_history WHERE p.date - (SELECT MAX(date) FROM @portfolio_history WHERE date <= td.tradedate) AND p.stockid = r.stockid
WHERE r.date <= td.tradeDate AND s.FirstQuoteDate <= td.tradeDate AND
(s.LastQuoteDate > td.tradedate OR s.DeadFlag = 'N')
AND ISNULL((SELECT TOP 1 Exchange FROM stock_exchange_history WHERE StockID = r.StockID AND Date <= td.tradedate ORDER BY Date DESC), s.Exchange) NOT IN (SELECT Exchange FROM
#excluded_exchanges)
GROUP BY r.stockid

INSERT INTO @newPositions(stockid, rating)
SELECT DISTINCT ar.stockid, ar.rating
FROM #ratings as ar
INNER JOIN #temp 1
ON ar.stockid = 1.stockid AND ar.date = 1.date
--INNER JOIN #t_dates td on 1=1
WHERE ar.Rating = ISNULL(@RatingType, ar.Rating)

-- compare current positions to the old positions

IF (SELECT COUNT(*) FROM @newPositions as p WHERE NOT EXISTS(SELECT * FROM @positions WHERE
stockid = p.stockid and date=p.date)) <> 0 OR
(SELECT COUNT(*) FROM @positions as p WHERE NOT EXISTS(SELECT * FROM @newPositions WHERE
stockid = p.stockid and date=p.date)) <> 0
SELECT @rebalancing = 'Y'

INSERT INTO #intra_dates (IntraDate, MktCapValue, EqualValue, NumOfStocks)
SELECT distinct IntraDate - d.date, @MktCapValue, @EqualValue, @NumOfStocks
FROM trading_dates d
INNER JOIN #t_dates td on 1=1
WHERE d.date > @PrevComponentDate AND d.Date > @StartDate AND d.Date < td.tradedate AND
d.Date NOT IN(SELECT Date FROM stock_holidays)

INSERT INTO #prices (IntraDate, StockID, MktCapShares, EqualShares, OldPrice, Price,
MktCapValue, EqaulValue)
SELECT d.intradate, c.stockid, c.MktCapShares, c.EqualShares,
oldprice = c.price, price = h.closeat, d.MktCapValue, d.EqualValue
FROM #intra_dates as d
INNER JOIN @positions c ON h.stockid = c.stockid AND h.date = d.intradate
LEFT JOIN stock_history_safe as h (nolock) ON h.stockid = c.stockid
AND h.date = d.intradate
UPDATE #prices SET price = (SELECT TOP 1 closeat FROM stock_history_safe WHERE stockid = #prices.stockid
AND date <= intradate ORDER BY date DESC)
WHERE price IS NULL
INSERT INTO #performance_history (Date, MktCapValue, EqualValue, NumOfStocks, [Type])
SELECT h.IntraDate, MktCapValue = ISNULL(SUM(MktCapShares * Price), h.MktCapValue),
EqualValue = ISNULL(SUM(EqualShares * Price), h.EqualValue), h.NumOfStocks, 'I'
FROM #intra_dates as h
LEFT JOIN #prices as p ON h.IntraDate = p.IntraDate
WHERE NOT EXISTS(SELECT * FROM #performance_history WHERE date = p.intradate)
GROUP BY h.IntraDate, h.NumOfStocks, h.MktCapValue, h.EqualValue
UPDATE @positions SET price = NULL, marketcap = NULL
UPDATE @positions set price = closeat, marketcap = h.marketcap
FROM @positions as c
INNER JOIN stock_history_safe as h ON h.stockid = c.stockid
WHERE h.date = (SELECT max(tradedate) from #t_dates)
UPDATE @positions SET price = (SELECT TOP 1 closeat FROM stock_history_safe, #t_dates td WHERE stockid = s.stockid
AND date <= td.tradedate ORDER BY date DESC),
marketcap = (SELECT TOP 1 marketcap FROM stock_history_safe, #t_dates td WHERE stockid = s.stockid
AND marketcap IS NOT NULL ORDER BY date DESC)
FROM @positions as s
WHERE price IS NULL OR marketcap IS NULL

This procedure is run for indices which have stocks between 100 to 800
My parameters are
exec calc_iip_index_history1
    @UserID = NULL,
    @IndexID = 206803, // it have 699 stocks
    @StartDate = '2002-01-01',
    @TotalReturn = 'Y',
    @IsCountUp = 'Y',
    @IsDebug = 'N'
Conclusion: --

There are about 200 active indices which take an entire day (About 20 hrs) to compute all of them. The code is still on testing. If it is a success, the computational time will reduce to an hour.

Reference: -- tuning case study from a travel application. On course web Page.
b) Removing Cursor from Procedure which calculates and store data for all the trading dates:

Before:

```
DECLARE cur_dates_ CURSOR FOR SELECT Date FROM trading_dates WHERE Date >= @StartDate ORDER BY Date
OPEN cur_dates_

FETCH FROM cur_dates_ INTO @Date
WHILE @@FETCH_STATUS = 0 AND i < @MaxSteps
BEGIN
  PRINT ' Calculating Date: ' + CONVERT(VARCHAR(10), @Date, 101) + '; Time: ' + CONVERT(VARCHAR(20), GETDATE(), 14)
  /* *********************** New Portfolio Initialization *****************************/
  /* find all new portfolios for this date */
  TRUNCATE TABLE #new_portfolios
  INSERT INTO #new_portfolios (PortfolioID)
  SELECT PortfolioID FROM #recalculation_list l
  WHERE l.LastIndexDate IS NULL AND l.InitialPortfolioDate < @Date
  AND NOT EXISTS (SELECT * FROM #portfolio_values WHERE PortfolioID = l.PortfolioID)
  /*
   if (select count(*) from #new_portfolios) > 0
   begin
     select [new portfolios date] = @date, * from #new_portfolios
     select * from #portfolios
     select * from #portfolio_changes
   end
  */
  /* initialize new portfolios */
  INSERT INTO #portfolios (PortfolioID, StockID, Status, NewStatus)
  SELECT DISTINCT h.PortfolioID, h.StockID, 'C', 'C'
  FROM iip_user_portfolio_history_usa h
  INNER JOIN (
    SELECT h.PortfolioID, h.StockID, Date = MAX(h.Date)
    FROM #new_portfolios l
    INNER JOIN iip_user_portfolio_history_usa h ON h.PortfolioID = l.PortfolioID
    INNER JOIN stocks s ON s.ID = h.StockID AND s.LastQuoteDate > @StartDate --
    include only the stocks which were traded on this date
    WHERE h.Date <= @Date
  ) l
  ON l.PortfolioID = h.PortfolioID AND l.StockID = h.StockID AND l.Date = h.Date
  WHERE h.ActionID = 1
  /* initialize values of the new portfolios */
  INSERT INTO #portfolio_values (PortfolioID, Val)
  SELECT PortfolioID, @StartValue
  FROM #new_portfolios
```
/******************** Resume calculation of the existing portfolios
************************/
/* adjust the resumed positions for stock splits */

```sql
INSERT INTO #portfolio_values (PortfolioID, Val) 
SELECT l.P PortfolioID, c.CustomValue 
FROM #recalculation_list l 
INNER JOIN index_performance_history c ON c.IndexID = l.IndexID AND c.Date = l.LastIndexDate 
WHERE l.NextCalcDate = @Date
```

```sql
/* find all changes since the previous date */
TRUNCATE TABLE #portfolio_changes 
INSERT INTO #portfolio_changes (PortfolioID, StockID, ActionID) 
SELECT DISTINCT h.PortfolioID, h.StockID, h.ActionID 
FROM iip_user_portfolio_history_usa h 
INNER JOIN ( 
SELECT ID = MAX(p.ID) -- p.PortfolioID, p.StockID, Date = MAX(p.Date) 
FROM #recalculation_list l 
INNER JOIN iip_user_portfolio_history_usa p ON p.PortfolioID = l.PortfolioID 
WHERE p.Date > @PrevDate AND p.Date <= @Date 
GROUP BY p.PortfolioID, p.StockID ) l ON l.ID = h.ID -- l.PortfolioID = h.PortfolioID AND l.StockID = h.StockID 
AND l.Date = h.Date
```

```sql
/* mark the positions which have to be removed from the portfolio because we got a remove signal */
UPDATE #portfolios SET NewStatus = 'D' 
FROM #portfolios p 
INNER JOIN #portfolio_changes c ON c.PortfolioID = p.PortfolioID AND c.StockID = p.StockID 
WHERE c.ActionID = 2
```

```sql
/* add new positions to portfolios */
INSERT INTO #portfolios (PortfolioID, StockID, NewStatus) 
SELECT PortfolioID, StockID, 'N' 
FROM #portfolio_changes c 
WHERE ActionID = 1 
AND NOT EXISTS (SELECT * FROM #portfolios WHERE PortfolioID = c.PortfolioID 
AND StockID = c.StockID) -- i don't like this. we don't need this line
```

```sql
/* mark the positions which have to be removed from the portfolio because the stock is dead now */
UPDATE #portfolios SET NewStatus = 'D' 
FROM #portfolios p 
INNER JOIN stocks s ON s.ID = p.StockID 
WHERE s.LastQuoteDate <= @Date AND s.DeadFlag = 'Y'
```

```sql
TRUNCATE TABLE #stocks 
INSERT INTO #stocks SELECT DISTINCT StockID FROM #portfolios
```
UPDATE #portfolios SET NewPrice = ISNULL(s.CloseAt, Price)
FROM #portfolios p
INNER JOIN (  
SELECT l.StockID, h.CloseAt  
FROM #stocks l  
INNER JOIN stock_history_safe h ON h.StockID = l.StockID AND h.Date = @Date  
) s ON s.StockID = p.StockID
TRUNCATE TABLE #dividends
INSERT INTO #dividends (StockID, Dividend)
SELECT d.StockID, SUM(Dividend)  
FROM stock_dividends d  
INNER JOIN #stocks s ON s.StockID = d.StockID  
WHERE d.Date > @PrevDate AND d.Date <= @Date  
GROUP BY d.Stockid
INSERT INTO #dividend_history (PortfolioID, Date, StockID, Dividend, Shares)
SELECT PortfolioID, @Date, p.StockID, d.Dividend, p.Shares  
FROM #portfolios p  
INNER JOIN #dividends d ON d.StockID = p.Stockid  
WHERE p.Status = 'C'
SELECT @PrevDate = @Date  
SELECT @I = @I + 1  
FETCH FROM cur_dates_ INTO @Date
END
CLOSE cur_dates_
DEALLOCATE cur_dates_

After:--

INSERT INTO #positions (PortfolioID, StockID, Shares, Date, Status, NewStatus)
SELECT DISTINCT 1PortfolioID, c.StockID, c.CustomShares, c.Date, 'C', 'C'
FROM #recalculation_list l
INNER JOIN index_component_history c ON c.IndexID = l.IndexID AND c.Date = l.LastRebalanceDate

update #positions set newprice = sh.CloseAt from #positions p
inner join stock_history sh on sh.StockID=p.StockID
and sh.Date=(select max(date) from stock_history where date<=p.date)
update #positions set shares=(price/newprice where (price/newprice)<1

// Above highlighted code is extra code added in procedure in order to insert values in #portfolios table which is very complicated in cursor

SELECT Date into #t_dates FROM trading_dates WHERE Date >= @StartDate ORDER BY Date
INSERT INTO #portfolios (PortfolioID, StockID, date, Shares, Status, NewStatus, td.date)
select PortfolioID, StockID, td.date, Shares, Status, NewStatus, td.date from #positions
Inner join #t_dates td on 1=1 order by date

INSERT INTO #portfolio_values (PortfolioID, Val)
SELECT l.PortfolioID, c.CustomValue FROM #recalculation_list l
INNER JOIN index_performance_history c ON c.IndexID = l.IndexID AND c.Date = l.LastIndexDate

INSERT INTO #portfolio_changes (PortfolioID, StockID, ActionID, Date)
SELECT h.PortfolioID, h.StockID, h.ActionID, l.date 
FROM iip_user_portfolio_history_usa h
INNER JOIN ( 
    SELECT ID = MAX(p.ID),
    td.date
    FROM #recalculation_list l
    INNER JOIN iip_user_portfolio_history_usa p ON p.PortfolioID = l.PortfolioID
    inner join #t_dates td on 1=1
    WHERE p.date > td.date and p.Date <= td.date
    GROUP BY p.PortfolioID, p.StockID, td.date
    ) l ON l.ID = h.ID --l.PortfolioID = h.PortfolioID AND l.StockID = h.StockID AND l.Date = h.Date

UPDATE #portfolios SET NewStatus = 'D'
FROM #portfolios p
INNER JOIN #portfolio_changes c ON c.PortfolioID = p.PortfolioID AND c.StockID = p.StockID AND p.date=c.date
WHERE c.ActionID = 2

UPDATE #portfolios SET NewStatus = 'D'
FROM #portfolios p
INNER JOIN stocks s ON s.ID = p.StockID
inner join #t_dates td on td.date-p.date
WHERE s.LastQuoteDate <= td.Date AND s.DeadFlag = 'Y'

DELETE FROM #portfolios WHERE NewStatus = 'D'

INSERT INTO #stocks(Stockid, Date, price) SELECT StockID, Date, NULL FROM #portfolios

update #portfolios set price = h.closeat
FROM #portfolios l
inner join stock_history_safe h ON l.date = h.date
and h.StockID = l.StockID
select max(s.date) date, s.stockid, s.price into #temp10 from #portfolios s
inner join #portfolios p on s.stockid=p.stockid
where s.date<=p.date group by s.stockid, s.date, s.price

update #portfolios set NewPrice = sh.price from #portfolios p
inner join #temp10 sh on sh.stockid=p.stockid
and p.date=sh.date

insert into #dividends (StockID,Dividend,Date)
select td.stockid, td.dividend, sd.date from stock_dividends sd
inner join #stocks st on st.stockid=sd.stockid and st.date=sd.date

insert into #dividend_history (PortfolioID,Date,StockID,Dividend,Shares)
select PortfolioID, p.Date, p.StockID, d.Dividend, p.Shares
from #portfolios p
inner join #dividends d on d.StockID = p.Stockid and d.date=p.date
WHERE p.Status = 'C'

// above highlighted code store all the values from first trading date to last trading date

Performance Gain for 4 day calculation (23, 24, 25 and 27 November)

TIMEtaken (in seconds)
Conclusion: -- For 1 portfolio, the time performance gain is tremendous, other than all portfolios together whose performance degrades.

Reason:--

a) Single Portfolio:
   4 days calculation (time period) = 170 stocks on an average.

   So, #portfolios table contains 170 * 4 = 608 records entries which are not much in numbers.

b) All Portfolios:

   Total numbers of portfolios together = 250

   So #portfolios table contains 250 * 170 (stocks) * 4 (days) which gives 1, 70,000 records.

Reference: -- tuning case study from a travel application. On course web Page.
CASE 2

a) Avoiding Dead Index during peak hours:

This is more like a logical change to gain performance. There are about 50 portfolios used by the cursor. Among which, 4 of them do not remain active for long and are called as dead portfolios. They haven’t entered active state since early 2009. There are other 3 to 4 portfolios which are in least use and not often updated. Apart from these (7-8) portfolios, rests are updated mostly often.

Procedure is deliberated in such a way that even if there are not enough parameters available, the process will reflect on all the portfolios along with building control from last calculation date. In the current case, last calculation date will be 19/Jan/2009 because of dead indices. Calculations will be done for all the portfolios from 2009, even if most of the portfolios are already up to date.

Before: -- Yellow Highlighted code below shows how it picks @startdate

```sql
SELECT IndexID = l.ID, PortfolioID = p.ID,
       InitialPortfolioDate = (SELECT MIN(Date) FROM iip_user_portfolio_history_usa
                                WHERE PortfolioID = p.ID),
       LastIndexDate = (SELECT MAX(Date) FROM index_performance_history WHERE
                        IndexID = l.ID),
       NextCalcDate = CONVERT(SMALLDATETIME, NULL),
       LastRebalanceDate = CONVERT(SMALLDATETIME, NULL)
INTO #recalculation_list
FROM iip_user_portfolios p
INNER JOIN index_list l ON l.ID = p.IndexID
where(@PortfolioID IS NOT NULL AND p.ID = @PortfolioID AND p.UserID = ISNULL(@UserID, p.UserID))
OR (@PortfolioID IS NULL AND p.TypeID IN (1,4) AND p.UserID = ISNULL(@UserID, p.UserID) AND p.SourceID IN (4))

DELETE FROM #recalculation_list WHERE InitialPortfolioDate IS NULL

IF @Recalculate = 'N'
BEGIN
/* this is the date where we should resume reclaculation of the existing indices */
UPDATE #recalculation_list SET NextCalcDate = (SELECT MIN(Date) FROM trading_dates WHERE Date > LastIndexDate)

/* this is were we'll pick up the latest components to resume our calculations */
UPDATE #recalculation_list SET LastRebalanceDate = (SELECT MAX(Date) FROM index_component_history WHERE IndexID = l.IndexID AND Date <= LastIndexDate)
FROM #recalculation_list l
END
```
IF @MaxAge IS NOT NULL
    DELETE FROM #recalculation_list WHERE ISNULL(NextCalcDate,'1900/01/01') < DATEADD(DAY, -@MaxAge - 1, GETDATE())

DECLARE @MinPortfolioDate SMALLDATETIME, @MinIndexDate SMALLDATETIME

/* find the very first last calculated date of an index */
IF @Recalculate = 'Y'
BEGIN
    UPDATE #recalculation_list SET LastIndexDate = NULL
    SELECT @MinIndexDate = NULL
END
ELSE
    SELECT @MinIndexDate = MIN(NextCalcDate) FROM #recalculation_list
/* find the very first date of the not-yet-calculated portfolio */
SELECT @MinPortfolioDate = MIN(InitialPortfolioDate) FROM #recalculation_list WHERE LastIndexDate IS NULL

SELECT @StartDate = ISNULL(CASE WHEN ISNULL(@MinPortfolioDate, @MinIndexDate) < [ISNULL](@MinIndexDate, @MinPortfolioDate) THEN ISNULL(@MinIndexDate, @MinPortfolioDate) ELSE ISNULL(@MinIndexDate, @MinPortfolioDate) END, @StartDate)
// Value in @startdate will be used as a first trading date in cursor

After :

Modifications have been made to first SQL part which is highlighted in blue above.

select id, Name, IndexID, UserID, SourceID, last_date = (select max(date) into #indices from index_performance_history where indexid = p.indexid)
from iip_user_portfolios p
WHERE --p.id = 523
(@PortfolioID IS NOT NULL AND p.ID = @PortfolioID AND p.UserID = ISNULL(@UserID, p.UserID))
OR (@PortfolioID IS NULL AND p.TypeID IN (1,4) AND p.UserID = ISNULL(@UserID, p.UserID) AND p.SourceID IN (4))

select p.ID 'PortfolioID', p.IndexID, p.last_date 'LastIndexDate',
InitialPortfolioDate = (SELECT MIN(Date) FROM iip_user_portfolio_history_usa
WHERE PortfolioID = p.ID),
NextCalcDate=convert(SMALLDATETIME, NULL),
LastRebalanceDate=convert(SMALLDATETIME, NULL)
into #recalculation_list from #indices p where p.last_date =

    select top 1 last_date from #indices where last_date IS NOT NULL
    group by last_date order by count(*) desc

    // the top 1 last_date will take the latest date portfolios and follows the same code.
Performance Gain (In Seconds) for one day calculation

Conclusion: -- It is tough to remove Cursor from 100’s of procedures but one can improve the performance by just providing simple changes. As mentioned above, time taken by one day calculation should be 3 sec, but the same is consuming 111 sec since some portfolios are not up to date.

Solution to the above problem can be given by removing all portfolios which are not active. But at the same time, the database which is being used has financial data and doing something might be able to influence other data. For instance, when a company merges with some other external source, every time the source name will be altered. Deletion of old records could diverse the analysis terms in determining company’s history and could be misinterpreted.
B) Debugging:

Performance in terms of speed is a critical factor. But the problem arises, if one procedure when runs for the first time calculate result quickly in comparison with the same when used to run for the second time takes a while.

Reason: -- For the first run, calculations begin from last trading date that is today’s date. But in second run, it starts from root 2002/01/01. This can be overcome by modifying the logic and printing “up—to—date”, if procedure is already calculating today’s date.

Before: --

```sql
DECLARE @StartDate SMALLDATETIME, @StartValue FLOAT
SELECT @StartDate = '2002/01/01', @StartValue = 1000, @MaxSteps = ISNULL(@MaxSteps, 20000)

IF @Recalculate = 'N'
BEGIN
   /* this is the date where we should resume reclaculation of the existing indices */
   UPDATE #recalculation_list SET NextCalcDate = (SELECT MIN(Date) FROM trading_dates WHERE Date > LastIndexDate)
   /* this is were we'll pick up the latest components to resume our calculations */
   UPDATE #recalculation_list SET LastRebalanceDate = (SELECT MAX(Date) FROM index_component_history WHERE IndexID = l.IndexID AND Date <= LastIndexDate)
   FROM #recalculation_list l
END

IF @MaxAge IS NOT NULL
   DELETE FROM #recalculation_list WHERE ISNULL(NextCalcDate,'1900/01/01') < DATEADD(DAY, -@MaxAge - 1, GETDATE())

DECLARE @MinPortfolioDate SMALLDATETIME, @MinIndexDate SMALLDATETIME

/* find the very first last calculated date of an index */
IF @Recalculate = 'Y'
BEGIN
   UPDATE #recalculation_list SET LastIndexDate = NULL
   SELECT @MinIndexDate = NULL
END
ELSE
   SELECT @MinIndexDate = MIN(NextCalcDate) FROM #recalculation_list

/* find the very first date of the not-yet-calculated portfolio */
SELECT @MinPortfolioDate = MIN(InitialPortfolioDate) FROM #recalculation_list
WHERE LastIndexDate IS NULL
```
SELECT @StartDate = ISNULL(CASE WHEN ISNULL(@MinPortfolioDate, @MinIndexDate) > ISNULL(@MinIndexDate, @MinPortfolioDate) THEN ISNULL(@MinPortfolioDate, @MinIndexDate) ELSE ISNULL(@MinIndexDate, @MinPortfolioDate) END, @StartDate)

// If calculation is up to date @MinIndexDate and @MinPortfolioDate will be Null and it will take default @StartDate which is 2002/01/01

After: --

DECLARE @StartDate SMALLDATETIME, @StartValue FLOAT
SELECT @StartDate = '2002/01/01', @StartValue = 1000, @MaxSteps = ISNULL(@MaxSteps, 20000)

IF @Recalculate = 'N'
BEGIN

/* this is the date where we should resume recalculation of the existing indices */
UPDATE #recalculation_list SET NextCalcDate = (SELECT MIN(Date) FROM trading_dates WHERE Date > LastIndexDate)

/* this is were we'll pick up the latest components to resume our calculations */
UPDATE #recalculation_list SET LastRebalanceDate = (SELECT MAX(Date) FROM index_component_history WHERE IndexID = l.IndexID AND Date <= LastIndexDate)
FROM #recalculation_list l
END

IF @MaxAge IS NOT NULL
DELETE FROM #recalculation_list WHERE ISNULL(NextCalcDate,'1900/01/01') < DATEADD(DAY, -@MaxAge - 1, GETDATE())

DECLARE @MinPortfolioDate SMALLDATETIME, @MinIndexDate SMALLDATETIME

/* find the very first last calculated date of an index */
IF @Recalculate = 'Y'
BEGIN
UPDATE #recalculation_list SET LastIndexDate = NULL
SELECT @MinIndexDate = NULL
END
ELSE
SELECT @MinIndexDate = MIN(NextCalcDate) FROM #recalculation_list

// NextCalcDate will be NULL if everything is Up to date

/* find the very first date of the not-yet-calculated portfolio */
SELECT @MinPortfolioDate = MIN(InitialPortfolioDate) FROM #recalculation_list WHERE LastIndexDate IS NULL

IF @MinIndexDate IS NULL -- Important because if calculation are up to date calc starts from 2002/01/01 which doesn’t make any sense
BEGIN
print 'calculation up--to--date'
SELECT @StartDate = @MinIndexDate // this will make @Startdate NULL
END
ELSE
BEGIN
SELECT @StartDate = ISNULL(CASE WHEN ISNULL(@MinPortfolioDate, @MinIndexDate) < ISNULL(@MinIndexDate, @MinPortfolioDate) THEN ISNULL(@MinPortfolioDate, @MinIndexDate) ELSE ISNULL(@MinIndexDate, @MinPortfolioDate) END, @StartDate)
END

Performance Gain (If Procedure runs for second time in a Day) in Minutes: --

Conclusion: -- A total of 18 minutes of running time has been replaced by a NULL value by just adding one “if-else” statement to the logic and the procedure won’t run again if calculations are already being done.
Case 3) Correlated Sub Queries Problem

Number of rows in #portfolios varies largely. The case is for 45,000 rows.

Before (OLD SQL)

update #portfolios set NewPrice = sh.price from #portfolios p
inner join (select max(s.date) date, s.stockid, s.price from #portfolios s
inner join #portfolios p on s.stockid = p.stockid
where s.date <= p.date group by s.stockid, s.date, s.price) sh on
sh.stockid = p.stockid
and p.date = sh.date

Here are the reads done by above SQL

Table '#portfolios_000000001B2D6'. Scan count 2, logical reads 5030, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table '#portfolios_000000001B2D6'. Scan count 3, logical reads 368465, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'Worktable'. Scan count 212695, logical reads 2633843, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

After (Modified SQL)

select max(s.date) date, s.stockid, s.price into #temp10 from #portfolios s
inner join #portfolios p on s.stockid = p.stockid
where s.date <= p.date group by s.stockid, s.date, s.price

update #portfolios set NewPrice = sh.price from #portfolios p
inner join #temp10 sh on sh.stockid = p.stockid
and p.date = sh.date

Here are the reads done by modified SQL

Table '#portfolios_00000001B303'. Scan count 2, logical reads 5030, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table '#temp10_00000001B317'. Scan count 5, logical reads 685, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table '#portfolios_00000001B303'. Scan count 5, logical reads 363435, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Performance Gain:

In the worktable at the end of output, modified SQL has 0 logical reads where as old SQL has 2633843 logical read. Moreover, old SQL will have slow performance (additional 10 secs) than modified SQL.

Reference: -- database tuning slides at course web page slide No 102 and MS programming in SQL

---

CASE 4) Convert Complicated Queries into simpler form:

The code below is to compute stocks value, 7 days before and after ratings. Calculations for 5 business days (7 days) before and after ratings seems very measured and increase time by 12 minutes.

```sql
CREATE TABLE #impact(rating_date smalldatetime, stockid int, symbol varchar(100),
stock_name varchar(100), analyst_name varchar(100), term varchar(100),
standing int, id int, alias varchar(100), coverage char(20),
weekafter_closeat decimal(20,4));

CREATE TABLE #impact2(rating_date smalldatetime, stockid int, symbol varchar(100),
stock_name varchar(100), analyst_name varchar(100), term varchar(100),
standing int, id int, alias varchar(100), coverage char(20),
weekafter_closeat decimal(20,4));

CREATE TABLE #impact3(rating_date smalldatetime, stockid int, symbol varchar(100),
stock_name varchar(100), analyst_name varchar(100), term varchar(100),
standing int, id int, alias varchar(100), coverage char(20),
weekafter_closeat decimal(20,4));

select date, next_date = (select max(date) from trading_dates d where (select count(*) from trading_dates where date < t.date and date >= d.date) = 5)
into #mydates from trading_dates t

select date, next_date = (select max(date) from trading_dates d where (select count(*) from trading_dates where date > t.date and date <= d.date) = 5)
into #mydates3 from trading_dates t
```
select r.date, r.stockid, s.symbol, s.name stock_name, a.name, t.term, t.standing, t.id, ata.alias, r.coverage
into #templ from analyst_ratings r
inner join analysts a on a.id = r.analystid
inner join alpha_firm_classification_data d on d.valueid = a.id
inner join alpha_firm_classification_items c on c.id = d.itemid
inner join stocks s on s.id = r.stockid
inner join analyst_terms t on t.id = r.terminid
inner join analyst_terms_aliases ata on ata.alias = r.terminaliasid
--inner join #mydates md on r.date=md.date
--left join stock_history h on h.stockid = r.stockid and h.date=md.next_date
where c.name='major fundamental / bulge bracket' and c.classificationid=2 and s.countrycode='USA' and t.standing <> 0 and coverage <> 're'
order by r.date desc

insert into #impact3 select t1.date,t1.stockid, t1.symbol, t1.stock_name, t1.name, t1.term, t1.standing, t1.id, t1.aliass, t1.coverage, h.closeat from #templ t1
left join stock_history h on h.stockid = t1.stockid and h.date=(select min(date) from stock_history where stockid = h.stockid and date >= t1.date)

insert into #impact select t1.date,t1.stockid, t1.symbol, t1.stock_name, t1.name, t1.term, t1.standing, t1.id, t1.aliass, t1.coverage, h.closeat from #templ t1
inner join #mydates md on t1.date=md.date
left join stock_history h on h.stockid = t1.stockid and h.date=md.next_date

insert into #impact2 select t1.date,t1.stockid, t1.symbol, t1.stock_name, t1.name, t1.term, t1.standing, t1.id, t1.aliass, t1.coverage, h.closeat from #templ t1
inner join #mydates md on t1.date=md.date
left join stock_history h on h.stockid = t1.stockid and h.date=md.next_date

Yellow highlighted code above, have very high scan count and logical reads
And it takes 12 min to compute both the code where number of rows are just 2500

select date, next_date = (select max(date) from trading_dates d where (select count(*) from trading_dates where date < t.date and date >= d.date) = 5) into #mydates from trading_dates t

Table 'trading_dates'. Scan count 3267847, logical reads 19158127, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

select date, next_date = (select max(date) from trade_dates d where (select count(*) from trade_dates where date > t.date and date <= d.date) = 5) into #mydates3 from trade_dates t
After:

```sql
CREATE TABLE #impact(rating_date smalldatetime, stockid int, symbol varchar(100),
stock_name varchar(100), analyst_name varchar(100), term varchar(100),
standing int, alias varchar(100), coverage char(20),
weekafter_closeat decimal(20,4));
CREATE TABLE #impact2(rating_date smalldatetime, stockid int, symbol varchar(100),
stock_name varchar(100), analyst_name varchar(100), term varchar(100),
standing int, alias varchar(100), coverage char(20),
weekafter_closeat decimal(20,4));
CREATE TABLE #impact3(rating_date smalldatetime, stockid int, symbol varchar(100),
stock_name varchar(100), analyst_name varchar(100), term varchar(100),
standing int, alias varchar(100), coverage char(20),
weekafter_closeat decimal(20,4));

select date, next_date=(select max(d.date) from trading_dates d where d.date=t.date-7) into #trading_dates1 from trading_dates t
delete from #trading_dates1 where next_date is not null
select date, next_date=(select min(d.date) from trading_dates d where d.date=t.date+7) into #trading_dates2 from trading_dates t
delete from #trading_dates2 where next_date is not null

select r.date, r.stockid, s.symbol, s.name stock_name, a.name, t.term, t.standing, t.id, ata.alias, r.coverage
into #temp1 from analyst_ratings r
inner join analysts a on a.id = r.analystid
inner join alpha_firm_classification_data d on d.valueid = a.id
inner join alpha_firm_classification_items c on c.id = d.itemid
inner join stocks s on s.id = r.stockid
inner join analyst_terms t on t.id = r.termid
inner join analyst_terms_aliases ata on ata.id = r.termmaliasid
inner join #mydates md on r.date=md.date
left join stock_history h on h.stockid = r.stockid and h.date=md.next_date
where c.name='major fundamental / bulge bracket' and c.classificationid=2 and s.countrycode='USA' and t.standing <> 0 and coverage <> 're'
order by r.date desc
```
```sql
insert into #impact3 select t1.date, t1.stockid, t1.symbol, t1.stock_name, t1.name, t1.term, t1.standing, t1.id, t1.alias, t1.coverage, h.closeat 
from #temp1 t1 left join stock_history h on h.stockid = t1.stockid and h.date=(select min(date) from stock_history where stockid = h.stockid and date >= t1.date)
insert into #impact select t1.date, t1.stockid, t1.symbol, t1.stock_name, t1.name, t1.term, t1.standing, t1.id, t1.alias, t1.coverage, h.closeat from #temp1 t1 
inner join #mydates md on t1.date=md.date 
left join stock_history h on h.stockid = t1.stockid and h.date=md.next_date
insert into #impact2 select t1.date, t1.stockid, t1.symbol, t1.stock_name, t1.name, t1.term, t1.standing, t1.id, t1.alias, t1.coverage, h.closeat 
from #temp1 t1 
inner join #mydates3 md3 on t1.date=md3.date 
left join stock_history h on h.stockid = t1.stockid and h.date=md3.next_date
```

The scan count and logical reads for modified SQL in yellow highlighted above are

Table 'trading_dates'. Scan count 2557, logical reads 5459, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table '#trading_dates1______________________________________________00000001D22F'. Scan count 1, logical reads 7, physical reads 0, read-ahead reads 0, lob physical reads 0, lob read-ahead reads 0.
Table 'trading_dates'. Scan count 2557, logical reads 5459, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
Table '#trading_dates2______________________________________________00000001D230'. Scan count 1, logical reads 7, physical reads 0, read-ahead reads 0, lob physical reads 0, lob read-ahead reads 0.
And it hardly takes 2 seconds

Conclusion: Avoid unnecessary complicated subqueries.
APPENDIX

Tables and variables used in Case 1) A

DECLARE @AnalystID smallint,
@IndustryID smallint,
@PortfolioID smallint
SELECT @AnalystID = NULL,
@IndustryID = NULL,
@PortfolioID = NULL

DECLARE @positions TABLE(
    stockid int, rating char(1), price decimal(19,5), newprice decimal(19,5),
    Dividend decimal(19,5), NewDividend decimal(19,5), marketcap decimal(19,5),
    MktCapShares float, PriceShares float, EqualShares float, SShares float,
    MktCapWeight float, EqualWeight float, SubPeriodReturn float
    PRIMARY KEY(StockID))

DECLARE @newPositions TABLE(
    stockid int, rating char(1), price decimal(19,5), marketcap decimal(19,5), Date SMALLDATETIME
)

DECLARE @oldPositions TABLE(
    stockid int, rating char(1), price decimal(19,5), dividend decimal(19,5),
    marketcap decimal(19,5), MktCapShares float, PriceShares float, EqualShares float,
    SShares float
    PRIMARY KEY(StockID))

DECLARE @totals TABLE(
    MarketCap decimal(19,5), Price decimal(19,5), NumOfStocks smallint,
    MktCapValue float, PriceValue float, EqualValue float, SPValue float, MktCapDividend float,
    PriceDividend float, EqualDividend float, SPDividend float)

DECLARE @portfolio TABLE(stockid int PRIMARY KEY(stockid))

DECLARE @portfolio_history_dates TABLE(
    date SMALLDATETIME, prev_date SMALLDATETIME
    PRIMARY KEY(date))

DECLARE @portfolio_history TABLE(
    date smalldatetime, stockid int
    PRIMARY KEY(date, stockid))

DECLARE @dividends TABLE(
    stockid int, date smalldatetime, dividend decimal(19,5)
    PRIMARY KEY(stockid, date))
/* This table holds the exchanges which have to be excluded from the
calculations */
CREATE TABLE #excluded_exchanges (Exchange CHAR(3))
INSERT INTO #excluded_exchanges (Exchange) VALUES ('OTC')
INSERT INTO #excluded_exchanges (Exchange) VALUES ('PNK')
INSERT INTO #excluded_exchanges (Exchange) VALUES ('OBB')

CREATE TABLE #ratings(
stockid int, date smalldatetime, rating char(1), coverage char(2),
termid tinyint)

CREATE TABLE #index_component_history(
  Date smalldatetime, RatingDate smalldatetime, StockID int, TermID tinyint,
  Price decimal(19,5), MktCap decimal(19,5), MktCapShares float,
  PriceShares float, EqualShares float, SPShares float, PriceDate smalldatetime,
  Dividend decimal(19,5),
  NextPrice decimal(19,5), NextDividend decimal(19,5), MktCapWeight float,
  EqualWeight float,
  SubPeriodReturn float, StartSlice char(1) DEFAULT 'N'
  PRIMARY KEY (Date, StockID))

CREATE TABLE #performance_history(
  Date smalldatetime, MktCapValue float, PriceValue float, EqualValue float,
  SPValue float, NumOfStocks float, MktCapBuys float, PriceBuys float,
  EqualBuys float, SPBuys float, MktCapShsTraded float,
  PriceShsTraded float, EqualShsTraded float, SPShsTraded float, [Type] char(1) )
  PRIMARY KEY (Date))

CREATE TABLE #intra_dates (IntraDate SMALLDATETIME, MktCapValue FLOAT,
EqualValue FLOAT, NumOfStocks FLOAT)
--check--------------------------------------------------------------------------

CREATE TABLE #prices (IntraDate SMALLDATETIME, StockID INT, MktCapShares
FLOAT, EqualShares FLOAT, OldPrice DECIMAL(19,5), Price DECIMAL(19,5),
MktCapValue FLOAT, EuqalValue FLOAT)

DECLARE @dates TABLE(
  date smalldatetime, tradedate smalldatetime
  PRIMARY KEY(date))

DECLARE @date smalldatetime, @init char(1), @PrevDate smalldatetime,
@LastComponentDate smalldatetime, @PrevComponentDate smalldatetime
CREATE VIEW stock_exchange_history AS SELECT [ID],[StockID],[Exchange],[Date] FROM quotes.dbo.stock_exchange_history

CREATE VIEW [dbo].[trading_dates] AS SELECT Date, StocksTraded, MaxTraded, MonthEnd, WeekEnd, MonthStart, MonthMiddle,TimeAdded FROM Quotes..trading_dates (nolock)

Number of Rows: - 35,658

CREATE VIEW stock_history_safe AS SELECT StockID, Date, CloseAt = CASE WHEN CloseAt < 1 THEN 1 ELSE CASE WHEN CloseAt > 1000000000 THEN 1000000000 ELSE CloseAt END END, OpenAt = CASE WHEN OpenAt < 1 THEN 1 ELSE CASE WHEN OpenAt > 1000000000 THEN 1000000000 ELSE OpenAt END END, MarketCap FROM stock_history

Number of rows: -- 9, 15, 99,390

Case 1) B

CREATE TABLE #portfolios (PortfolioID SMALLINT, StockID INT, Price DECIMAL(18,5), NewPrice DECIMAL(18,5), Shares FLOAT, Status CHAR(1), NewStatus CHAR(1), OldShares FLOAT PRIMARY KEY (PortfolioID, StockID))

CREATE TABLE #new_portfolios (PortfolioID SMALLINT PRIMARY KEY (PortfolioID))

CREATE TABLE #portfolio_values (PortfolioID SMALLINT, Val FLOAT, Dividend DECIMAL(18,5), NumOfStocks SMALLINT, Buys FLOAT PRIMARY KEY (PortfolioID))

CREATE TABLE #portfolio_value_history (PortfolioID SMALLINT, Date SMALLDATETIME, Val FLOAT, NumOfStocks SMALLINT, Type CHAR(1), Buys FLOAT PRIMARY KEY (PortfolioID, Date))

CREATE TABLE #portfolio_history (PortfolioID SMALLINT, Date SMALLDATETIME, StockID INT, Price DECIMAL(18,5), NewPrice DECIMAL(18,5), Shares FLOAT PRIMARY KEY (PortfolioID, Date, StockID))

CREATE TABLE #changes (PortfolioID SMALLINT PRIMARY KEY (PortfolioID))

CREATE TABLE #portfolio_changes (PortfolioID SMALLINT, StockID INT, ActionID TINYINT PRIMARY KEY (PortfolioID, StockID))

CREATE TABLE #dividend_history (PortfolioID SMALLINT, Date SMALLDATETIME, StockID INT, Dividend DECIMAL(18,5), Shares FLOAT PRIMARY KEY (PortfolioID, Date, StockID))

CREATE TABLE #dividends (StockID INT, Dividend DECIMAL(18,5) PRIMARY KEY (StockID))

CREATE TABLE #stocks (StockID INT PRIMARY KEY (StockID))

CREATE VIEW index_component_history
AS
SELECT
IndexID, StockID, Date,
RatingDate, TermID, Price,
MktCap, MktCapShares, PriceShares,
EqualShares, PriceDate, Dividend,
SPShares, CustomShares, NextPrice,
NextDividend, MktCapWeight, PriceWeight,
EqualWeight, CustomWeight, SubPeriodReturn,
CustomContrib, MktCapContrib, EqualContrib,
CurrencyID
FROM CalcDB.dbo.index_component_history

Number of Rows:-- more than 90,10,00,000

CREATE VIEW [dbo].[stock_history]
AS
SELECT h.StockID, h.[Date], h.OpenAt, h.CloseAt,
h.Volume, h.NullOpen, [Dummy] = 1, MarketCap = h.MarketCap/* */
isnull(c.Factor,1)/, h.high, h.low, h.vwap
FROM Quotes.dbo.stock_history h

Number of Rows:-- 91599392

CREATE VIEW dbo.stocks
AS
SELECT
s.ID, s.Symbol, ISNULL(s.Name,'') as 'Name', s.DeadFlag,
 s.MarketCap,
 s.MarketCapUSD, s.RIC, s.CUSIP, s.ISIN, s.SEDOL, s.Exchange,
 s.LastQuoteDate, s.Disabled, s.CountryCode, s.CurrencyID,
s.InstrumentTypeID as 'InstrumentID',
e.RegionID, e.Description as 'CountryName', s.GeographicalUnit as 'CountryID'
FROM quotes..stocks AS s (nolock)
LEFT JOIN quotes..countries e (nolock) ON e.Code = s.GeographicalUnit

Number of Rows :-- 58850

CREATE VIEW iip_user_portfolio_history_usa AS
SELECT h.ID, PortfolioID, h.StockID, ActionID, Date, TargetWeight, Shares,
CreateDate
FROM iip_user_portfolio_history h
INNER JOIN stocks s ON s.ID = h.StockID
WHERE s.CountryCode = 'USA'

Number of rows:-- 46083

To remove the cursor I changed the DDL for most of the tables

CREATE TABLE #portfolios (PortfolioID SMALLINT, StockID INT, Price
DECIMAL(18,5), NewPrice DECIMAL(18,5), Shares FLOAT, Status CHAR(1),
NewStatus CHAR(1), OldShares FLOAT, Date SMALLDATETIME)

CREATE TABLE #positions (PortfolioID SMALLINT, StockID INT, Price
DECIMAL(18,5), NewPrice DECIMAL(18,5), Shares FLOAT, Date SMALLDATETIME,
Status CHAR(1), NewStatus CHAR(1), OldShares FLOAT PRIMARY KEY (PortfolioID, StockID)

CREATE TABLE #new_portfolios (PortfolioID SMALLINT PRIMARY KEY (PortfolioID))

CREATE TABLE #portfolio_values (PortfolioID SMALLINT, Val FLOAT, Dividend DECIMAL(18,5), NumOfStocks SMALLINT, Buys FLOAT, Date SMALLDATETIME) -- PRIMARY KEY (PortfolioID))

CREATE TABLE #portfolio_value_history (PortfolioID SMALLINT, Date SMALLDATETIME, Val FLOAT, NumOfStocks SMALLINT, Type CHAR(1), Buys FLOAT) -- PRIMARY KEY (PortfolioID, Date))

CREATE TABLE #portfolio_history (PortfolioID SMALLINT, Date SMALLDATETIME, StockID INT, Price DECIMAL(18,5), NewPrice DECIMAL(18,5), Shares FLOAT PRIMARY KEY (PortfolioID, Date, StockID))

CREATE TABLE #changes (PortfolioID SMALLINT, Date SMALLDATETIME PRIMARY KEY (PortfolioID, date))

CREATE TABLE #portfolio_changes (PortfolioID SMALLINT, StockID INT, ActionID TINYINT, Date SMALLDATETIME )

CREATE TABLE #dividend_history (PortfolioID SMALLINT, Date SMALLDATETIME, StockID INT, Dividend DECIMAL(18,5), Shares FLOAT)

CREATE TABLE #dividends (StockID INT, Dividend DECIMAL(18,5), Date SMALLDATETIME)

CREATE TABLE #stocks (StockID INT, Date SMALLDATETIME, Price DECIMAL(18,5))

Case 2, A) same as Case 1) B
Case 2, B) same as Case 1) B