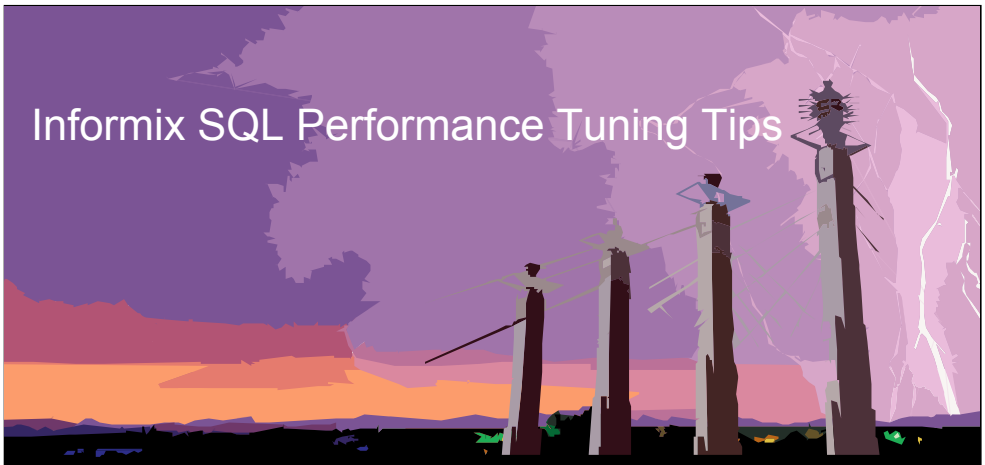
Informix SQL Performance Tuning Tips



Jeff Filippi
Integrated Data Consulting, LLC

Session: A12
Tuesday, April 29, 2008 • 4:40 p.m. – 5:40 p.m.

2008 IIUG Informix Conference

www.iiug.org

Session: A12

Informix SQL Performance Tuning Tips

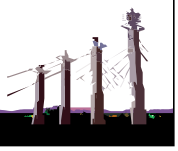
Jeff Filippi
Integrated Data Consulting, LLC
jeff.filippi@itdataconsulting.com

Introduction

- 18 years of working with Informix products
- 13 years as an Informix DBA
- Worked for Informix for 5 years 1996 – 2001
- Certified Informix DBA
- Started my own company in 2001 specializing in Informix Database Administration
- OLTP and Data warehouse systems
- Informix 4.x, 5.x., 7.x, 9.x, 10.x, 11.10

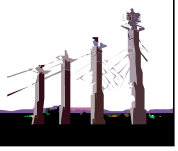
Overview

- Know your Application in Tuning SQL
- Case Studies
- Tuning for OLTP vs DSS Environments
- Reading sqexplain output and tuning examples
- Understanding Options used to Tune SQL
- Use of Informix Tools to Analyze and Tune SQL



Know Your Applications

- One of the biggest things that I see is that DBA's do not understand the business of the systems they are supporting to effectively support the systems.
- The most important item is that you understand the business of the system that you are trying to support and tune.
- Get involved early in the design, work with the developers in designing the systems.



Know your Application – cont'd

ISSUE

- Client where DBA's did not work with development.
- Developers designed tables.
- DBA's just maintained production.

RESULTS

- Poor Performance.
- DBA's did not understand how the application worked.
- There was finger pointing on who's problem it was, no accountability.

I was at a client where the DBA's were not involved with the design and development of the tables. The developers created the tables and stored procedures. The DBA's just executed the SQL to create them into production without really reviewing or understanding them. I saw developers creating tables with a primary key of (varchar (50)) and the table was to have 30 million records. After investigating, the field was going to have at most 8 characters used.

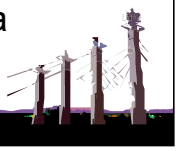
Know your Application – cont'd

RECOMENDATIONS

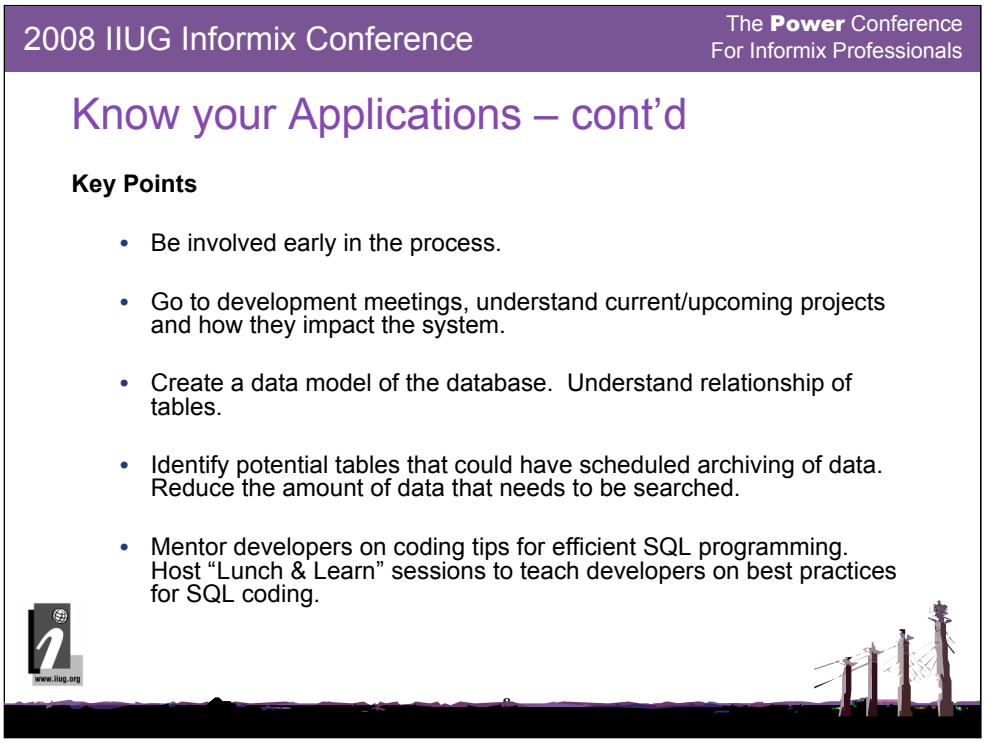
- Involved the DBA's to work with the developers from the beginning of the projects.
- Tables created jointly between developers and DBA's during development.

RESULTS

- DBA's now had a handle on enforcing what was approved for production.
- The number of issues that occurred after a project launch were reduced dramatically.



After a few positive results where they saw that the DBA's recommendations were making a positive impact, management was more open to new ideas that could further help.



Be involved with development to help them understand the proper way to write SQL statements. Conduct training classes with development group to educate them on proper SQL statement creation.

Case Studies – Case 1

Case Study 1

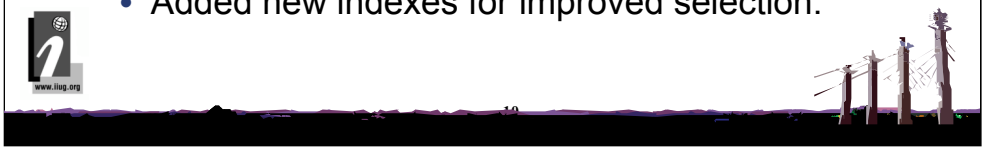
- Review the whole business logic, do not just review the specific SQL statements.
- Example:
  - In reviewing a client's performance issue, I saw that the specific SQL statement was written correctly.
  - The issues were the following:
    - Two tables had 20 times more reads than any other table
    - 87 extents on one table, 98 extents on the other

The tables in question were being selected in a common stored procedure that was being called by multiple stored procedures and was the last stored procedure called, it was the bottleneck.

Case Studies – Case 1

SOLUTION

- Reorganize the tables into single extent.
- Purged data that was no longer needed, which reduced table size by 70%.
- Created monthly job to purge records that were not needed.
- Rebuilt indexes that were last created 5 years ago.
- Added new indexes for improved selection.



The issue was not the specific SQL statement, but other issues. The SQL statement was optimized correctly. Do not only concentrate on the SQL statement itself.

Case Studies – Case 1

RESULTS

- Performance Improved Dramatically!!!!!
- Number of buffer reads on the two tables were reduced. They went from being the top two tables in number of reads by 20 times the next table to not even in the top 5.

After the changes were implemented, they saw an immediate improvement in performance. This was no longer the bottleneck in the system.

Case Studies – Case 2

Case Study 2

- Development was changing a process.
- How can DBA's help in development?
  - Probe them on how the new process will work.
  - Investigate how to improve the process further.

Development was changing a process of how they wrote data to the database for a specific process. Now instead of performing it in a batch type mode, it would be written to the database real time.

After asking numerous questions about how the tables were now to be updated, I then did some investigation.

Case Studies – Case 2

- During analysis of the process that was changing, take a step back and look at the whole process, not just the piece that is changing.
- Review how the change may help or hurt performance and what other changes may need to be made.
- Review other areas in the application where data is being selected and see if there are improvements that can be made.

Case Studies – Case 2

- The change was to write the data to the cart and cart_item table in real time instead of batch mode.
- After looking at the tables, I wondered why would the cart table have more records than the child table cart_items?
- I questioned the developers, should a cart exist with no items, the answer was no there should not be.

For every cart record, there were multiple cart_items records. Well after looking at the tables, cart (85 million rows), cart_item (16 million rows), I questioned the developers.

Asked the developers if there should be a cart record without a cart_item record. There response was that the previous method did not clean up carts with no cart_item records, but the new process would do this. I then identified that only 4 million cart records had a cart_item record.

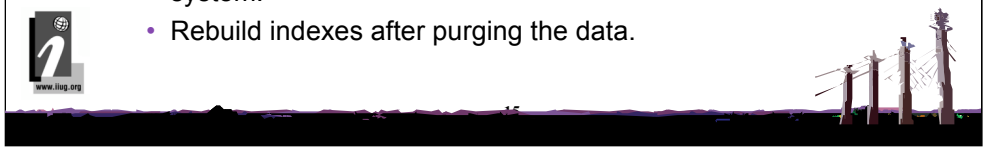
Case Studies – Case 2

After reviewing the new process, I found some improvements to be made.

- Reduced one table (CART) row count by 90% from 85 million to 4 million by purging cart records with no items.

  Implementation:

  - Since the system was 24X7, I could not take an outage long enough to rebuild the CART table which would have been optimal.
  - I ended up writing a stored procedure to delete the records and commit after every 100 deletes to keep the transactions small and to not cause any locking issues for the users on the system.
  - Rebuild indexes after purging the data.



Rebuilt cart table with only those records that had a corresponding cart_item record. Reduced the number of rows from 85 million to 4 million (90% reduction). This resulted in the existing queries performance improving due to the reduced number or records to be read and rebuilt indexes.

Case Studies – Case 3

Client was having performance issues when load on the system increased.

- Investigated the index with the highest buffer reads.
- After reviewing the SQL statements being processed using that index, I identified that it was not the best index on the table being used for the query.

This was one of the things that we did to help improve performance of the system, not the only thing.

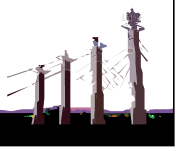
Case Studies – Case 3

Resolution

- Ran update statistics high on a column in the index which previously had "medium" statistics on it.
- Dropped an index that was a duplicate of another index.

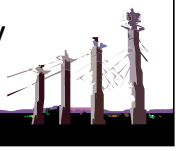
RESULTS

- Total number of buffer reads on the system decreased by 50%.
- During their peak times, there are no performance issues, the system performed flawlessly.



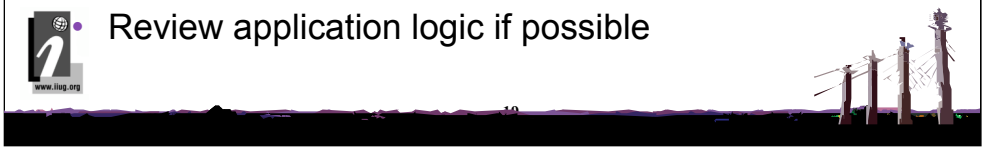
Tuning for OLTP vs DSS

- OLTP
  - Small number of rows returned
  - Quick response times of Queries
- DSS
  - Large number of rows returned
  - Quick response not as important
- Combination of OLTP/DSS
  - Balancing DSS Queries with OLTP activity



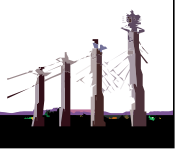
OLTP

- Focus on index reads, quick return of small amounts of data
- Review how often SQL statements are executed, use SQL Statement Cache
- Understand how the data is searched to then create the correct indexes
- Review application logic if possible



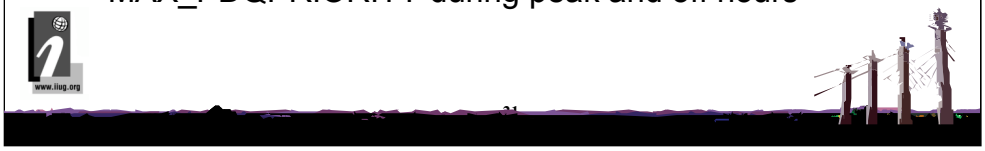
DSS

- Focus on how data is used, utilize fragmentation
- Review SQL for faster query timings
- Utilize PDQPRIORITY
- Utilize temp tables for improved performance
- Have enough temporary dbspaces for sorts



Combination of OLTP/DSS

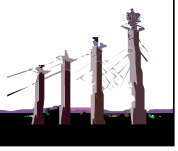
- Need to balance resources/loads on the system
- Run DSS type queries in off hours
- Minimize amount of DS_TOTAL_MEMORY / MAX_PDQPRIORITY used during peak time, increase during off hours to run DSS type queries
- Dynamically adjust DS_TOTAL_MEMORY / MAX_PDQPRIORITY during peak and off hours



In today's environment, most systems end up being a combination of both OLTP and DSS environments. A system may be mainly OLTP, but over time, management may want to pull more reports out of the system in a batch mode. The balance is making sure that the DSS queries do not negatively impact the OLTP system, but can work well enough to give the desired results.

Identify Problem SQL Statements

- First you have to identify what SQL statements are the culprits in causing performance issues
  - Use “onstat –g ntt” to identify the last time read/writes occurred
  - Gather slow SQL statements from onstats, 3rd party tools, etc.
  - Review with developers known problem areas in the application
  - Verify update statistics are current
  - Review what indexes have the most reads
  - Using IDS 11.10 feature Tracing SQL

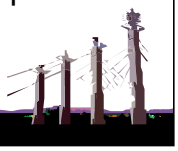


There are many different ways to identify problem SQL statements, here are just a few examples of how to accomplish this.

Tracing SQL in IDS 11.10

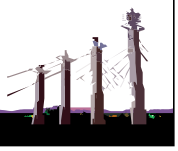
- There are a couple ways to turn tracing on in IDS 11.10
  - Onconfig parameter: SQLTRACE
    - level = [off,low,med,high]
    - ntraces = [# of traces]
    - size = [size of each trace buffer in kb]
    - mode = [global,user]
    - Example:
      - SQLTRACE level=low,ntraces=1000,size=2,mode= global

        (This allows me to trace the last 1000 sql statements of the instance)



Tracing SQL in IDS 11.10 (cont'd)

- You can also enable tracing thru the "sysadmin" database by running the following command:
  - EXECUTE FUNCTION task("set sql tracing on",1000,2,"low","global")
- To validate that tracing is turned on by:
  - onstat –g his
  - This option prints information about the SQLTRACE configuration parameter.



Tracing SQL in IDS 11.10 (cont'd)

- **onstat -g his Output**

Statement Statistics:

Page Read	Buffer Read	Read % Cache	Buffer IDX Read	Page Write	Buffer Write	Write % Cache
1781	34423	94.78	6175	863	16356	94.07

Lock Requests	Lock Waits	LK Wait Time (S)	Log Space	Num Sorts	Disk Sorts	Memory Sorts
22489	0	0.0000	100.5 KB	0	0	0

Total Executions	Total Time (S)	Avg Time (S)	Max Time (S)	Avg IO Wait	I/O Wait Time (S)	Avg Rows Per Sec
1	10.6451	10.6451	10.6451	0.0092	18.4552	521.9123

Estimated Cost	Estimated Rows	Actual Rows	SQL Error	ISAM Error	Isolation Level	SQL Memory
204	9412	3123	0	0	CR	65441

Tracing SQL in IDS 11.10 (cont'd)

- You can also get information on the tracing thru the **"syssqltrace"** table in the sysmaster database.
  - Ex. {# of queries that ran > 2 seconds)

```
SELECT count(*)
FROM syssqltrace
WHERE sql_totaltime > 2;
```

- Another useful table is the **"syssqltrace_iter"** which gives information in the form of an iteration tree for each SQL. It allows you to know identify which part of the query plan took the most time to run.

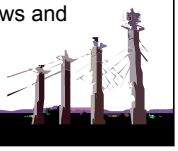
Options Available with Set Explain

- Optimizer Directives – AVOID_EXECUTE
  - Introduced in IDS 9.30
  - Generate query plan without executing SQL, useful for getting query plans for inserts, updates and delete where data is manipulated, but you do not want to change data
  - Example:
    - set explain on AVOID_EXECUTE;
    - SQL Statement

I will discuss in more detail later on “dynamic set explain”.

Options Available with Set Explain

- Set Explain Enhancements
  - Another improvement with IDS 11.10 is that you can turn on/off explain statistics thru the onconfig parameter “EXPLAIN_STAT”.
    - 0 – Disables the display of query statistics
    - 1 – Enables the display of query statistics
  - FYI, this is an undocumented feature in IDS 10.
  - You can also set it with the following statement:
    - SET EXPLAIN STATISTICS
  - When this is enabled, the inclusion of the “Query Statistics” section in the explain output file. It shows the query plan’s estimated number of rows and the actual number of rows returned.



I will discuss in more detail later on “dynamic set explain”.

Options Available with Set Explain - Query Statistics

```
QUERY:
------
select *
from partsupp
where ps_partkey >= 1 and ps_partkey <= 100 and ps_suppkey >= 0 and ps_suppkey <= 100000 and ps_availqty >= 1000 and ps_availqty <= 1000000

Estimated Cost: 49
Estimated # of Rows Returned: 360

 1) informix.partsupp: INDEX PATH
   (1) Index Keys: ps_partkey ps_suppkey ps_availqty   (Key-First)  (Serial, fragments: ALL)
       Lower Index Filter: informix.partsupp.ps_partkey >= 1 AND (informix.partsupp.ps_availqty >= 1000 ) AND (informix.partsupp.ps_suppkey >= 0 )
       Upper Index Filter: informix.partsupp.ps_partkey <= 100 AND (informix.partsupp.ps_availqty <= 1000000 ) AND (informix.partsupp.ps_suppkey <=100000
         )
       Index Key Filters:  (informix.partsupp.ps_availqty >= 1000 ) AND   (informix.partsupp.ps_availqty <= 1000000 ) AND
                    (informix.partsupp.ps_suppkey <= 100000 ) AND  (informix.partsupp.ps_suppkey >= 0 )

Query statistics:
-----------------
 Table map :
 ----------------------------
 Internal name    Table name
 ----------------------------------------
 t1               partsupp

 type    table  rows_prod  est_rows  rows_scan    time      est_cost
 ------------------------------------------------------------------------------------------------
 scan    t1          26        360          26        00:00:00     49
```

Dynamic Set Explain

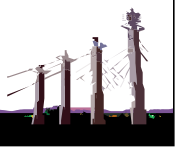
- Dynamically set explain on for a session
  (Introduced in 9.40)
  - Onmode –Y {session id} {0|1} (0 – Off/1 – On)
  - Output is to a file "sqexplain.out.{session id}
  - With IDS 11.10 there are a couple changes:
    - An additional value "2" (explain without statistics for session, displays query plan only)
    - Also you can specify the file name and directory that you want the explain output to be sent:
      - Onmode –Y {session id} {0|1|2} {filename}
- This is a great feature to allow you to see the SQL statements executed and the explain plan for each SQL statement.
  - **NOTE**: make sure that you only have this turned on for a short period of time, it creates a large file.

If you are tracing a session that was started by another user and/or in a specific directory, the "sqexplain.out.{session id}" will be in that directory, not where you executed the "onmode –Y'.

Watch the size of the "sqexplain.out" file, it can get very large very quickly.

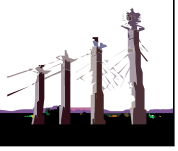
Set Explain Output

- Add “set explain on” before the statement you want to examine
- Starting in IDS11.10 you can specify the directory that you want the file to go:
  - Set explain file to “filename”
- Review the “set explain” output:
  - UNIX: The file “sqexplain.out” will be generated in the directory that you run the query from
  - Windows: Look for a file “username.out” in the directory on the UNIX server %INFORMIXDIR%\sqexpln



Set explain output

- **Query** – Displays the executed query and indicates whether "set optimization" was set to high or low
- **Directives Followed** – Lists any directives used for the query
- **Estimated Cost** – An estimated of the amount of work for the query. The number does not translate into time. Only compare to same query not others.
- **Estimated number of rows returned** – An estimate of the number of rows returned, number based on information from system catalog tables

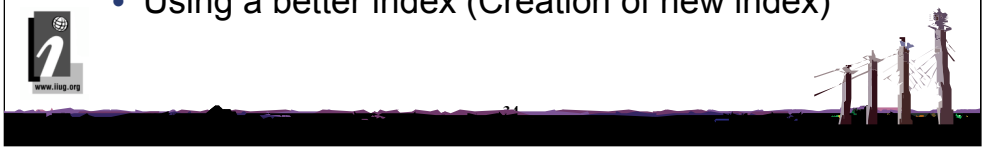


Set explain output – cont'd

- **Numbered List** – The order in which tables are accessed, followed by the access method (index or sequential scan)
- **Index Keys** – The columns used as filters or indexes
- **Query Statistics** – Enabled by onconfig parameter "EXPLAIN_STAT"

Examples of Explain Plans

- The following slides will show tuning of SQL based on the following scenarios:
  - Functions causing index to not be used
  - Criteria from views causing sequential scans
  - Use of Directives
  - Use of substrings in queries
  - Use of functions in queries
  - Using a better index (Creation of new index)



Here are a few examples of tuning SQL statements that will help you understand different scenarios to look for when tuning SQL statements.

Function causes index to not be used

```
QUERY:
------
SELECT DISTINCT BUSINESS_UNIT, VOUCHER_ID, INVOICE_ID, GROSS_AMT,
   INVOICE_DT, VENDOR_NAME_SHORT, VENDOR_ID, NAME1, VOUCHER_STYLE,
   ENTRY_STATUS_SRH
FROM PS_VOUCHER_SRCH_VW
WHERE BUSINESS_UNIT='GH'
AND UPPER(INVOICE_ID) LIKE UPPER('KURT') || '%' ESCAPE '\'
ORDER BY INVOICE_ID, BUSINESS_UNIT, VOUCHER_ID DESC FOR READ ONLY


Estimated Cost: 55943
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

 1) sysadm.ps_vendor: SEQUENTIAL SCAN

 2) sysadm.ps_voucher: INDEX PATH

     Filters: (sysadm.ps_voucher.entry_status IN ('P' , 'R' , 'T' )AND UPPER(sysadm.ps_voucher.invoice_id ) LIKE
     'KURT%' ESCAPE '\' )

   (1) Index Keys: vendor_id vendor_setid business_unit   (Serial, fragments: ALL)
       Lower Index Filter: ((sysadm.ps_voucher.vendor_id = sysadm.ps_vendor.vendor_id AND
       sysadm.ps_voucher.vendor_setid = sysadm.ps_vendor.setid ) AND sysadm.ps_voucher.business_unit = 'GH'
       )
NESTED LOOP JOIN
```

In this example, the query is performing a sequencial scan on the PS_VENDOR table even though there is an index on the BUSINESS_UNIT and INVOICE_ID field.

Using a function like “UPPER” on the field of the column of the table causes the index to not be used.

Resolution: Function causes index to not be used

```
QUERY:
------
SELECT DISTINCT BUSINESS_UNIT, VOUCHER_ID, INVOICE_ID, GROSS_AMT,
   INVOICE_DT, VENDOR_NAME_SHORT, VENDOR_ID, NAME1, VOUCHER_STYLE,
   ENTRY_STATUS_SRH
FROM PS_VOUCHER_SRCH_VW
WHERE BUSINESS_UNIT='GH'
AND INVOICE_ID LIKE 'KURT' || '%' ESCAPE '\'
ORDER BY INVOICE_ID, BUSINESS_UNIT, VOUCHER_ID DESC FOR READ ONLY

Estimated Cost: 35009
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

 1) sysadm.ps_voucher: INDEX PATH

      Filters: sysadm.ps_voucher.entry_status IN ('P' , 'R' , 'T' )

   (1) Index Keys: business_unit invoice_id   (Serial, fragments: ALL)
      Lower Index Filter: (sysadm.ps_voucher.business_unit = 'GH' AND sysadm.ps_voucher.invoice_id LIKE
      'KURT%' ESCAPE '\' )

 2) sysadm.ps_vendor: INDEX PATH

   (1) Index Keys: vendor_id setid   (Serial, fragments: ALL)
      Lower Index Filter: (sysadm.ps_voucher.vendor_id = sysadm.ps_vendor.vendor_id AND
      sysadm.ps_voucher.vendor_setid = sysadm.ps_vendor.setid )
NESTED LOOP JOIN
```

In this case it was guaranteed by the application that all character values in the INVOICE_ID field were upper case.

So being able to remove the “UPPER” function on the INVOICE_ID column we were now able to use the index.

Resolution: Function causes index to not be used

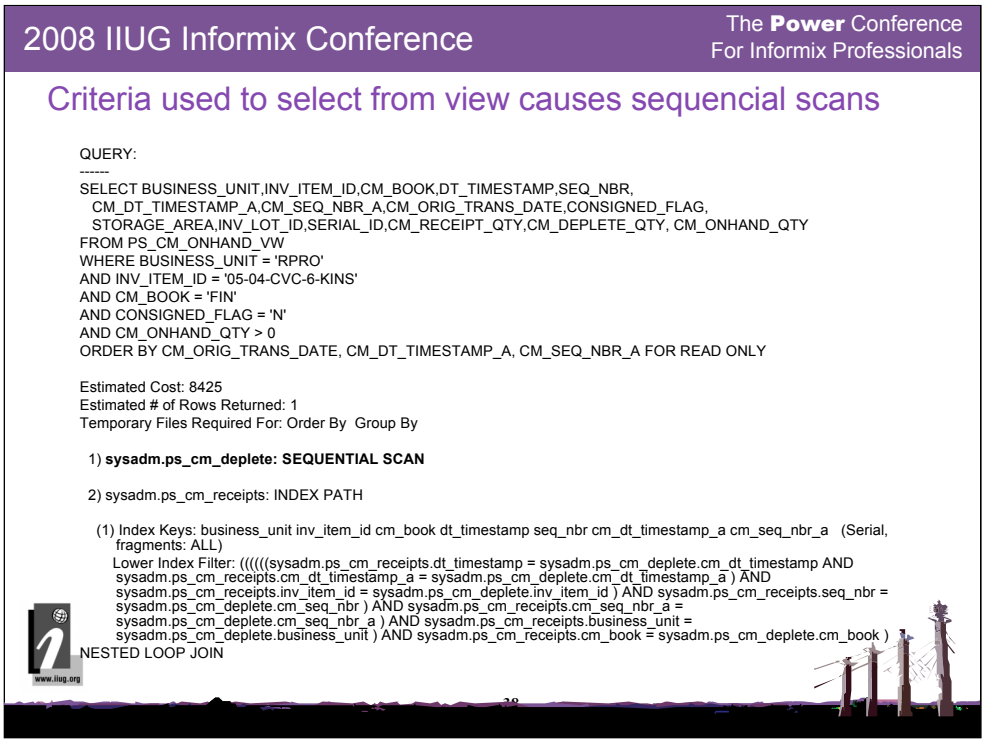
- Another way is to add a function which converts a character to all upper case and change the index to include the use of the function.

```
CREATE FUNCTION upper_idx(char_up char(20))
    RETURNING char(20) WITH (not variant);
    DEFINE char_out char(20);
    LET char_out = upper(char_up);
    RETURN char_out;
END FUNCTION;
```

```
CREATE INDEX upper_idx on ps_vendor(business_unit, (upper_idx(invoice_id))
    USING btree;
```

In this case it was guaranteed by the application that all character values in the INVOICE_ID field were upper case.

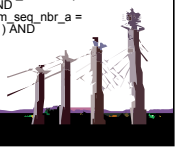
So being able to remove the “UPPER” function on the INVOICE_ID column we were now able to use the index.



In this example, the query is performing a sequencial scan on the view, even though there is an index with the fields (business_unit, inv_item_id, and cm_book).

Resolution to Criteria used for view causes sequential scans

```
QUERY:
------
SELECT BUSINESS_UNIT,INV_ITEM_ID,CM_BOOK,DT_TIMESTAMP,SEQ_NBR,
      CM_DT_TIMESTAMP_A,CM_SEQ_NBR_A,CM_ORIG_TRANS_DATE,CONSIGNED_FLAG,
      STORAGE_AREA,INV_LOT_ID,SERIAL_ID,CM_RECEIPT_QTY,CM_DEPLETE_QTY,
      CM_ONHAND_QTY
FROM PS_CM_ONHAND_VW
WHERE BUSINESS_UNIT = 'RPRO'
AND INV_ITEM_ID = '04X35-X-042'
AND CM_BOOK = 'FIN'
AND CONSIGNED_FLAG = 'N'
--AND CM_ONHAND_QTY > 0
ORDER BY CM_ORIG_TRANS_DATE, CM_DT_TIMESTAMP_A, CM_SEQ_NBR_A FOR READ ONLY

Estimated Cost: 10
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By  Group By

 1) sysadm.ps_cm_deplete: INDEX PATH

   (1) Index Keys: business_unit inv_item_id cm_book dt_timestamp seq_nbr cm_dt_timestamp cm_seq_nbr cm_dt_timestamp_a cm_seq_nbr_a   (Serial,
       fragments: ALL)
     Lower Index Filter: ((sysadm.ps_cm_deplete.inv_item_id = '04X35-X-042' AND sysadm.ps_cm_deplete.business_unit = 'RPRO' ) AND
       sysadm.ps_cm_deplete.cm_book = 'FIN' )

 2) sysadm.ps_cm_receipts: INDEX PATH

     Filters: sysadm.ps_cm_receipts.consigned_flag = 'N'

   (1) Index Keys: business_unit inv_item_id cm_book dt_timestamp seq_nbr cm_dt_timestamp_a cm_seq_nbr_a   (Serial, fragments: ALL)
     Lower Index Filter: (((((sysadm.ps_cm_receipts.inv_item_id = sysadm.ps_cm_deplete.inv_item_id AND sysadm.ps_cm_receipts.dt_timestamp =
       sysadm.ps_cm_deplete.cm_dt_timestamp ) AND sysadm.ps_cm_receipts.seq_nbr = sysadm.ps_cm_deplete.cm_seq_nbr ) AND
       sysadm.ps_cm_receipts.cm_dt_timestamp_a = sysadm.ps_cm_deplete.cm_dt_timestamp_a ) AND sysadm.ps_cm_receipts.cm_seq_nbr_a =
       sysadm.ps_cm_deplete.cm_seq_nbr_a ) AND sysadm.ps_cm_receipts.business_unit = sysadm.ps_cm_deplete.business_unit ) AND
       sysadm.ps_cm_receipts.cm_book = sysadm.ps_cm_deplete.cm_book )
NESTED LOOP JOIN
```

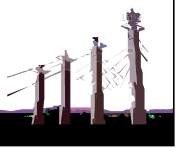


After trying the select without the criteria of "cm_onhand_qty > 0", the query was able to use the index. The use of the criteria that was calculated from the view caused the sequential scans.

In this case we were able to make a change to the application to filter out any "CM_ONHAND_QTY" less than zero.

View used in query

```
CREATE VIEW "sysadm".ps_cm_onhand_vw
  (business_unit,inv_item_id,cm_book,dt_timestamp,seq_nbr,cm_dt_timestamp_a, …….

cm_onhand_qty) AS

SELECT x1.business_unit ,x1.inv_item_id ,x1.cm_book ,x1.cm_dt_timestamp, …….

 (x0.qty_base - sum(x1.qty_base) )

FROM "sysadm".ps_cm_receipts x0 ,"sysadm".ps_cm_deplete x1
WHERE (((((((x0.business_unit = x1.business_unit )
AND (x0.inv_item_id = x1.inv_item_id ) )
AND (x0.cm_book = x1.cm_book) )
AND (x0.dt_timestamp = x1.cm_dt_timestamp ) )
AND (x0.seq_nbr = x1.cm_seq_nbr ) )
AND (x0.cm_dt_timestamp_a = x1.cm_dt_timestamp_a) )
AND (x0.cm_seq_nbr_a = x1.cm_seq_nbr_a ) )
GROUP BY x1.business_unit ,x1.inv_item_id ,x1.cm_book ,x1.cm_dt_timestamp ,
        x1.cm_seq_nbr,x0.cm_dt_timestamp_a ,x0.cm_seq_nbr_a ,
        x0.cm_orig_trans_date,x0.consigned_flag ,x0.storage_area ,
        x0.inv_lot_id ,x0.serial_id,x0.qty_base ;
```

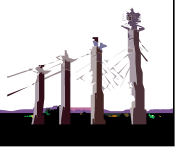


After investigating the view, the one criteria from the where clause (cm_onhand_qty > 0), the field is actually a calculated value.

Use of directives for Queries

```
QUERY:
------
SELECT D.BUSINESS_UNIT, D.VENDOR_SETID, E.VENDOR_ID, E.NAME1, E.NAME2, VNDR_LOC
FROM PS_PAYMENT_TBL A, PS_PYMNT_VCHR_XREF B, PS_VOUCHER_LINE C,
        PS_VOUCHER D, PS_VENDOR E, PS_VENDOR_LOC F
WHERE A.BANK_SETID = B.BANK_SETID
   AND A.BANK_CD = B.BANK_CD
   AND A.BANK_ACCT_KEY = B.BANK_ACCT_KEY
   AND A.PYMNT_ID = B.PYMNT_ID
   AND B.BUSINESS_UNIT = C.BUSINESS_UNIT
   AND B.VOUCHER_ID = C.VOUCHER_ID
   AND C.BUSINESS_UNIT = D.BUSINESS_UNIT
   AND C.VOUCHER_ID = D.VOUCHER_ID
   AND E.VENDOR_ID = D.VENDOR_ID
   AND A.PYMNT_STATUS = 'P'
   AND A.PYMNT_DT BETWEEN '01-01-2003' AND '12-31-2003'
   AND D.BUSINESS_UNIT IN ('CAT','SNCPY')
   AND E.SETID = F.SETID
   AND E.VENDOR_ID = F.VENDOR_ID
   AND C.WTHD_CD <> F.WTHD_CD
```

Estimated Cost: 57005
Estimated # of Rows Returned: 1



In some cases, no matter how you try to get the query to work the way you want, either with indexes, different update stats combinations, the optimizer decides to take a different path.

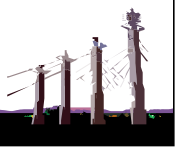
These are the times that the use of directives come in handy.

Use of Directive for Queries – cont'd

```
1) informix.f: INDEX PATH
     Filters: informix.f.effdt = <subquery>
   (1) Index Keys: setid vendor_id vndr_loc effdt (desc) eff_status   (Serial, fragments: ALL)
 2) informix.e: INDEX PATH
   (1) Index Keys: vendor_id setid   (Serial, fragments: ALL)
     Lower Index Filter: (informix.e.vendor_id = informix.f.vendor_id AND informix.e.setid = informix.f.setid )
     NESTED LOOP JOIN
 3) informix.d: INDEX PATH
     Filters: informix.d.business_unit IN ('CAT' , 'SNCPY' )
   (1) Index Keys: vendor_id vendor_setid entry_status   (Serial, fragments: ALL)
     Lower Index Filter: informix.d.vendor_id = informix.f.vendor_id NESTED LOOP JOIN
 4) informix.c: INDEX PATH
     Filters: informix.c.wthd_cd != informix.f.wthd_cd
   (1) Index Keys: business_unit voucher_id (desc) voucher_line_num   (Serial, fragments: ALL)
     Lower Index Filter: (informix.c.voucher_id = informix.d.voucher_id AND informix.c.business_unit =
     informix.d.business_unit ) NESTED LOOP JOIN
 5) informix.b: INDEX PATH
   (1) Index Keys: business_unit voucher_id (desc) pymnt_id bank_cd bank_acct_key   (Serial, fragments: ALL)
     Lower Index Filter: (informix.b.voucher_id = informix.c.voucher_id AND informix.b.business_unit =
     informix.d.business_unit )
NESTED LOOP JOIN
 6) informix.a: INDEX PATH
     Filters: ((informix.a.pymnt_dt >= 01/01/2003 AND informix.a.pymnt_status = 'P' ) AND informix.a.pymnt_dt <=
     12/31/2003 )

   (1) Index Keys: pymnt_id (desc) bank_acct_key bank_cd bank_setid   (Serial, fragments: ALL)
     Lower Index Filter: (((informix.a.pymnt_id = informix.b.pymnt_id AND informix.a.bank_acct_key =
     informix.b.bank_acct_key ) AND informix.a.bank_cd = informix.b.bank_cd ) AND informix.a.bank_setid =
     informix.b.bank_setid ) NESTED LOOP JOIN
```

Use of Directive for Queries – cont'd

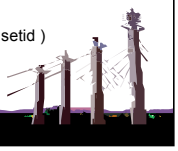
```
QUERY:
------
SELECT --+ORDERED
D.BUSINESS_UNIT, D.VENDOR_SETID, E.VENDOR_ID, E.NAME1, E.NAME2,   B.VNDR_LOC
FROM PS_PAYMENT_TBL A, PS_PYMNT_VCHR_XREF B, PS_VOUCHER_LINE C,
    PS_VOUCHER D, PS_VENDOR E, PS_VENDOR_LOC F
WHERE A.BANK_SETID = B.BANK_SETID
    AND A.BANK_CD = B.BANK_CD
    AND A.BANK_ACCT_KEY = B.BANK_ACCT_KEY
    AND A.PYMNT_ID = B.PYMNT_ID
    AND B.BUSINESS_UNIT = C.BUSINESS_UNIT
    AND B.VOUCHER_ID = C.VOUCHER_ID
    AND C.BUSINESS_UNIT = D.BUSINESS_UNIT
    AND C.VOUCHER_ID = D.VOUCHER_ID
    AND E.VENDOR_ID = D.VENDOR_ID
    AND A.PYMNT_STATUS = 'P'
    AND A.PYMNT_DT BETWEEN '01-01-2003' AND '12-31-2003'
    AND D.BUSINESS_UNIT IN ('CAT','SNCPY')
    AND E.SETID = F.SETID
    AND E.VENDOR_ID = F.VENDOR_ID
    AND C.WTHD_CD <> F.WTHD_CD

DIRECTIVES FOLLOWED:
ORDERED
DIRECTIVES NOT FOLLOWED:

Estimated Cost: 70888                (Cost of Original Query: 57005)
Estimated # of Rows Returned: 1
```



In this case we wanted the query to execute in the order that the tables were listed in the "FROM" clause. In order to do this we used the "ORDERED" directive.

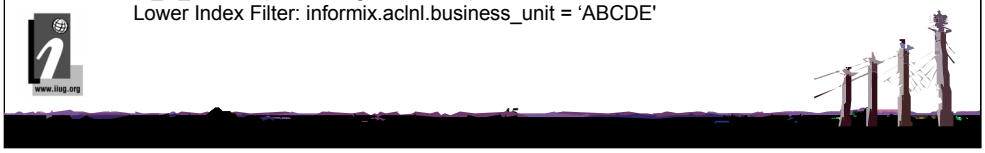
Use of Directive for Queries – cont'd

```
1) informix.a: INDEX PATH
     Filters: informix.a.pymnt_status = 'P'
   (1) Index Keys: pymnt_dt name1 remit_setid currency_pymnt   (Serial, fragments: ALL)
     Lower Index Filter: informix.a.pymnt_dt >= 01/01/2003
     Upper Index Filter: informix.a.pymnt_dt <= 12/31/2003
 2) informix.b: INDEX PATH
     Filters: informix.b.business_unit IN ('CAT' , 'SNCPY' )
   (1) Index Keys: bank_setid bank_cd bank_acct_key pymnt_id   (Serial, fragments: ALL)
     Lower Index Filter: (((informix.a.pymnt_id = informix.b.pymnt_id AND informix.a.bank_acct_key =
     informix.b.bank_acct_key ) AND informix.a.bank_cd = informix.b.bank_cd ) AND informix.a.bank_setid =
     informix.b.bank_setid ) NESTED LOOP JOIN
 3) informix.c: INDEX PATH
   (1) Index Keys: business_unit voucher_id (desc) voucher_line_num   (Serial, fragments: ALL)
     Lower Index Filter: (informix.b.voucher_id = informix.c.voucher_id AND informix.b.business_unit =
     informix.c.business_unit ) NESTED LOOP JOIN
 4) informix.d: INDEX PATH
   (1) Index Keys: voucher_id (desc) business_unit invoice_id   (Serial, fragments: ALL)
     Lower Index Filter: (informix.b.voucher_id = informix.d.voucher_id AND informix.b.business_unit =
     informix.d.business_unit ) NESTED LOOP JOIN
 5) informix.e: INDEX PATH
   (1) Index Keys: vendor_id setid   (Serial, fragments: ALL)
     Lower Index Filter: informix.e.vendor_id = informix.d.vendor_id NESTED LOOP JOIN
 6) informix.f: INDEX PATH
     Filters: (informix.c.wthd_cd != informix.f.wthd_cd AND informix.f.effdt = <subquery> )
   (1) Index Keys: setid vendor_id vndr_loc effdt (desc) eff_status   (Serial, fragments: ALL)
     Lower Index Filter: (informix.e.vendor_id = informix.f.vendor_id AND informix.e.setid = informix.f.setid )
     NESTED LOOP JOIN
```



Use of substrings (Best index not being used)

```
QUERY:
------
SELECT ACLNL.MONETARY_AMOUNT
FROM PS_CM_ACCTG_LINE ACLNL
WHERE ACLNL.BUSINESS_UNIT = 'ABCDE'
AND ACLNL.PRODUCTION_ID = '12334'
AND SUBSTR(ACLNL.ACCOUNT,1,3) IN ( '085' , '334', '072' )

Estimated Cost: 49722
Estimated # of Rows Returned: 1

 1) informix.aclnl: INDEX PATH

     Filters: (informix.aclnl.production_id = '12334' AND SUBSTR
    (informix.aclnl.account , 1 , 3 ) IN ('085' , '334' , '072' ))

   (1) Index Keys: business_unit cm_book gl_distrib_status budget_hdr_status
     cm_iu_status   (Serial, fragments: ALL)
      Lower Index Filter: informix.aclnl.business_unit = 'ABCDE'
```



Here is a case where the query was not using the best index available. There was an index by (business_unit, production_id and account), but it was not being used.

Resolution for Use of substrings

```
QUERY:
------
SELECT ACLNL.MONETARY_AMOUNT
FROM PS_CM_ACCTG_LINE ACLNL
WHERE ACLNL.BUSINESS_UNIT = 'ABCDE'
AND ACLNL.PRODUCTION_ID = '12334'
AND (ACLNL.ACCOUNT matches  '085*'
OR  ACLNL.ACCOUNT matches  '334*'
OR  ACLNL.ACCOUNT matches  '072*' )

Estimated Cost: 3
Estimated # of Rows Returned: 1

 1) informix.aclnl: INDEX PATH

   (1) Index Keys: business_unit production_id account   (Key-First)  (Serial, fragments: ALL)
       Lower Index Filter: (informix.aclnl.production_id = '12334' AND informix.aclnl.business_unit =
       'ABCDE' )
       Key-First Filters:  (((informix.aclnl.account MATCHES '085*' OR informix.aclnl.account
       MATCHES '334*' ) OR informix.aclnl.account MATCHES '072*' ) )
```

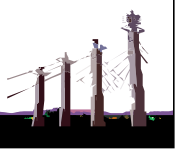
In changing the substring to a matches clause, the correct index was able to be used.

Use of Functions in Queries cause specific Indexes not be used

QUERY:

```
SELECT od.order_id AS order_id,od.club_model_id AS club_model_id,od.purchase_type_cd
    AS purchase_type_cd,od.order_status_cd AS order_status_cd,
    EXTEND(od.create_ts, YEAR TO DAY) AS create_ts,price AS price,
    shipping_amt AS shipping_amt,od.session_id AS session_id
FROM order_detail od, order_header oh
WHERE oh.source_id != -1
AND oh.source_id IS NOT NULL
AND oh.source_id != 23150010

AND EXTEND(od.create_ts, YEAR TO DAY) = '2004-05-17'

AND od.order_id = oh.order_id
AND club_model_id = 10
AND (purchase_type_cd = 'CLUB' OR purchase_type_cd = 'SEYMOS'
OR purchase_type_cd = 'DCSSORC' OR purchase_type_cd = 'SHVSSORC'
OR purchase_type_cd = 'DDVSSORC' OR purchase_type_cd = 'HSACNUF')
```

Estimated Cost: 546168
Estimated # of Rows Returned: 67774

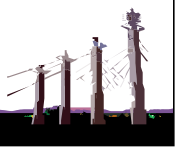


Here is a case where the query was not utilizing the most efficient index.

There was an index on the fields in the order_detail by (create_ts, purchase_type_cd, order_status_cd, club_model_id), but it was not being used.

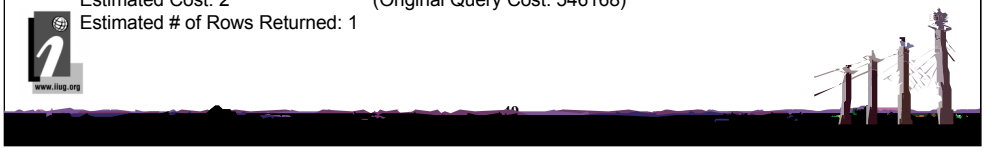
Use of Functions in Queries causes specific Indexes not be used

```
1) informix.od: INDEX PATH
   Filters: (EXTEND (informix.od.create_ts ,year to day) = datetime(2004-05-17) year to day

            AND (((((informix.od.purchase_type_cd = 'CLUB'
            OR informix.od.purchase_type_cd = 'SEYMOS' )
            OR informix.od.purchase_type_cd = 'DCSSORC' )
            OR informix.od.purchase_type_cd = 'SHVSSORC' )
            OR informix.od.purchase_type_cd = 'DDVSSORC' )
            OR informix.od.purchase_type_cd = 'HSACNUF' ) )

   (1) Index Keys: club_model_id   (Serial, fragments: ALL)
       Lower Index Filter: informix.od.club_model_id = 10

2) informix.oh: INDEX PATH
   Filters: (informix.oh.source_id != -1 AND (informix.oh.source_id IS NOT NULL
       AND informix.oh.source_id != 23150010 ) )

   (1) Index Keys: order_id   (Serial, fragments: ALL)
       Lower Index Filter: informix.oh.order_id = informix.od.order_id
NESTED LOOP JOIN
```



Resolution to Use of Functions in Queries

```
QUERY:
------
SELECT od.order_id AS order_id,od.club_model_id AS club_model_id,
      od.purchase_type_cd AS purchase_type_cd,od.order_status_cd AS order_status_cd,
      EXTEND(od.create_ts, YEAR TO DAY) AS create_ts,price AS price, shipping_amt
      ASshipping_amt,od.session_id AS session_id
FROM order_detail od, order_header oh
WHERE oh.source_id != -1
AND oh.source_id IS NOT NULL
AND oh.source_id != 23150010

AND (od.create_ts >= '2004-05-17 00:00:00.000' AND od.create_ts <= '2004-05-17 23:59:59.999')

AND od.order_id = oh.order_id
AND club_model_id = 10
AND (purchase_type_cd = 'CLUB' OR purchase_type_cd = 'SEYMOS'
OR purchase_type_cd = 'DCSSORC' OR purchase_type_cd = 'SHVSSORC'
OR purchase_type_cd = 'DDVSSORC' OR purchase_type_cd = 'HSACNUF')

Estimated Cost: 2                    (Original Query Cost: 546168)
Estimated # of Rows Returned: 1
```



The resolution was to not use the EXTEND function to check the date, but to query the field with the criteria that would get the same data for that day.

With this change the correct index was able to be used.

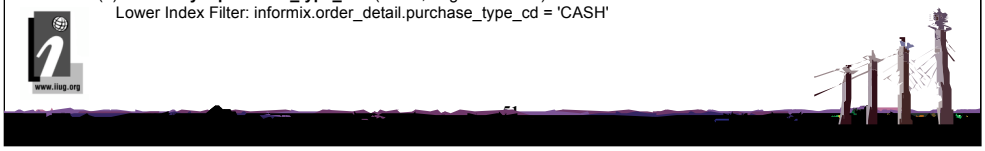
Resolution to Use of Functions in Queries

```
1) informix.od: INDEX PATH

  (1) Index Keys: create_ts purchase_type_cd order_status_cd club_model_id
    (Key-First)  (Serial, fragments: ALL)
     Lower Index Filter: informix.od.create_ts >= datetime(2004-05-17 00:00:00.000) year to
    fraction(3)
     Upper Index Filter: informix.od.create_ts <= datetime(2004-05-17 23:59:59.999) year to
    fraction(3)
     Key-First Filters:  ((((((informix.od.purchase_type_cd = 'CLUB'
    OR informix.od.purchase_type_cd = 'SEYMOS' )
    OR informix.od.purchase_type_cd = 'DCSSORC' )
    OR informix.od.purchase_type_cd = 'SHVSSORC' )
    OR informix.od.purchase_type_cd = 'DDVSSORC' )
    OR informix.od.purchase_type_cd = 'HSACNUF' ) )
    AND (informix.od.club_model_id = 10 )

2) informix.oh: INDEX PATH

   Filters: (informix.oh.source_id != -1 AND (informix.oh.source_id IS NOT NULL
 AND informix.oh.source_id != 23150010 ) )

   (1) Index Keys: order_id   (Serial, fragments: ALL)
      Lower Index Filter: informix.oh.order_id = informix.od.order_id
      NESTED LOOP JOIN
```

Here you can see the correct index was used.

Using a better index

```
QUERY:
------
select club_model_id, order_status_cd, count(distinct order_id) as
   order_count
from order_detail
where create_ts >= '2004-09-30 00:00:00.000'   and create_ts < '2004-10-02 00:00:00.000'
  and purchase_type_cd = 'CASH'
  and order_status_cd not in ('REJ', 'ACCP')
group by 1,2
order by 1,2


Estimated Cost: 407
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By  Group By

1) informix.order_detail: INDEX PATH

   Filters: (informix.order_detail.create_ts >= datetime(2004-09-30 00:00:00.000) year to fraction(3) AND
      (informix.order_detail.create_ts < datetime(2004-10-02 00:00:00.000) year to fraction(3) AND
      informix.order_detail.order_status_cd NOT IN ('REJ' , 'ACCP' )) )

   (1) Index Keys: purchase_type_cd   (Serial, fragments: ALL)
      Lower Index Filter: informix.order_detail.purchase_type_cd = 'CASH'
```



Here was a case where there was no indexes on the table that served the query well. It was using an index, but it was not much help since the PURCHASE_TYPE_CD contained only a few values.

Resolution to Using a better index

```
QUERY:  (AFTER ADDING A NEW INDEX)
------
select club_model_id, order_status_cd, count(distinct order_id) as
   order_count
from order_detail
where create_ts >= '2004-09-30 00:00:00.000'
  and create_ts < '2004-10-02 00:00:00.000'
  and purchase_type_cd = 'CASH'
  and order_status_cd not in ('REJ', 'ACCP')
group by 1,2
order by 1,2

Estimated Cost: 3
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By  Group By

1) informix.order_detail: INDEX PATH

   (1) Index Keys: create_ts purchase_type_cd order_status_cd club_model_id   (Key-First)  (Serial,
       fragments: ALL)
       Lower Index Filter: informix.order_detail.create_ts >= datetime(2004-09-30 00:00:00.000) year to fraction(3)
       Upper Index Filter: informix.order_detail.create_ts < datetime(2004-10-02 00:00:00.000) year to fraction(3)
       Key-First Filters:  (informix.order_detail.order_status_cd NOT IN ('REJ' , 'ACCP' )) AND
                           (informix.order_detail.purchase_type_cd = 'CASH'
```

The result was adding a new index starting the index with the column CREATE_TS, since this would filter the results better than any of the other fields in the query.

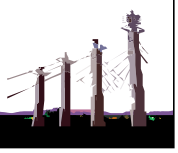
Understanding Options used to Tune SQL

- Utilize “UNIONS” when you have “OR” in where clause
- Utilize temp tables in optimizing queries by splitting the query into multiple queries
- Utilize PDQPRIORITY
- Utilize DS_NONPDQ_QUERY_MEM (V 9.40/10.00)
- Fragment tables (Understand the use of the data) to eliminate fragments from selection of the data
- Utilize external directives (V 10.00)
- Index Self-Join (V11.10)

There are many difference ways to help in tuning SQL statements, listed here are a few of them.

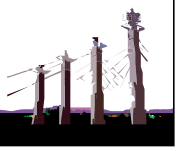
Utilize Unions

```
SELECT a.email_template_id, b.description, a.club_model_id, a.email_log_id, efd.field_id
FROM email_log a, email_template b, email_field_data efd
WHERE a.email_template_id = b.email_template_id
AND a.email_template_id = efd.email_template_id
AND efd.email_log_id = a.email_log_id
AND efd.field_id in (561, 558)
AND a.club_model_id in ('1', '2')
AND a.email_template_id in ('275','128')
UNION
SELECT a.email_template_id, b.description, a.club_model_id, 0 as email_log_id, 0 as field_id
FROM email_log a, email_template b
WHERE a.email_template_id = b.email_template_id
AND a.club_model_id in ('1', '2')
AND a.email_template_id in ('125','2171')
UNION
SELECT a.email_template_id, b.description, a.club_model_id, 1 as email_log_id, 1 as field_id
FROM email_log a, email_template b
WHERE a.email_template_id = b.email_template_id
AND a.club_model_id in ('3', '4')
AND a.email_template_id = '2152';
```



Utilize Temp Tables

```
SET PDQPRIORITY 100;
SELECT acct_n, gender
FROM v_master
WHERE acct_n MATCHES '90*' AND mbr_phase_cde IN ('E','F','M','R')
INTO TEMP tmp_v_master WITH NO LOG;
        NOTE: With V11.10, you can globally set the onconfig parameter "TEMPTAB_NOLOG"
                0 – Off (Enable logical logging on temp tables)
                1 – On (Disable logical logging on temp tables)
CREATE INDEX idx_vidmaster ON tmp_v_master(acct_n);
UPDATE STATISTICS LOW FOR TABLE tmp_v_master;
        NOTE: With V11.10 you no longer need to run update statistics on a temp table
SELECT pull, equip, type_equip
FROM cat_pull
WHERE equip in ('S','W','T','M')
INTO TEMP temp_cat_pull WITH NO LOG;

CREATE INDEX idx_cat_pull ON temp_cat_pull(pull, equip);
UPDATE STATISTICS LOW FOR TABLE temp_cat_pull;

SELECT --+ORDERED
    account, m.gender, c.type_equip,
FROM v_trans v, tmp_v_master m, temp_cat_pull c
WHERE cntrl_num >= 118265 AND cntrl_num < 118786
AND m.acct_n = v.account
AND (v.selection = c.pulland v.equip = c.equip)
AND (uimm > 0 OR upos > 0 OR udis > 0 OR ubon > 0 OR udoc > 0 OR ugaf > 0
     OR uxdoc > 0 OR ues > 0 OR ufso > 0 OR urain > 0 OR ufree > 0)
INTO TEMP tst WITH NO LOG;
```

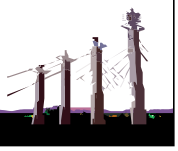


Utilize PDQPRIORITY

```
SET PDQPRIORITY 100;

SELECT accnt, pc, cntrl_wd, mfree, uauto, salestype
FROM vid_tran
WHERE substr(accnt,11,1) = '7'
AND (magz <> '' OR magz IS NOT NULL)
AND (pc LIKE 'BV1%' OR pc LIKE 'DA2%'
        OR pc LIKE 'DVM%' )
AND (cntrl_wd BETWEEN 118530 AND 119335)
```

Estimated Cost: 2485223
Estimated # of Rows Returned: 6067559

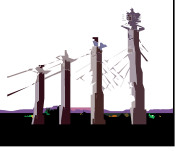
Maximum Threads: 3



Utilize DS_NONPDQ_QUERY_MEM

DS_NONPDQ_QUERY_MEM = 50,000

```
session                                  #RSAM    total      used       dynamic
id       user     tty      pid      hostname threads  memory     memory     explain
324499   indprod  -        27952    prodtu   1        2367488    2328592    off

tid      name     rstcb            flags    curstk   status
25339601 sqlexec  c0000002b2c9ac18 ---PR--  1842583336 sleeping(Forever)

Memory pools    count 2
name         class addr             totalsize  freesize   #allocfrag #freefrag
324499       V     c0000002b60be040 2306048    34848      4315       157
324499_SORT  V     c0000002b59a3040 61440      4048       7          1

name          free      used          name          free      used
sort          0         34144         sqscb         0         41344
sql           0         80            srtmembuf     0         20384

sqscb info
scb              sqscb            optofc   pdqpriority sqlstats optcompind  directives
c0000002b5be61d0 c0000002cb9d7030 0        0           0       0           1

Sess  SQL            Current            Iso Lock       SQL  ISAM F.E.
Id    Stmt type      Database           Lvl Mode       ERR  ERR  Vers Explain
324499 SELECT        elstest            CR  Wait 600   0    0    9.03 Off

Current SQL statement :
  select  unique b.* from tmp_fids a, name_init b where a.fid = b.fid
```



Utilize DS_NONPDQ_QUERY_MEM

DS_NONPDQ_QUERY_MEM 500,000

session #RSAM total used dynamic
id user tty pid hostname threads memory memory explain
16354 vijays - 16777 prodtu 1 2490368 2458128 off

Memory pools count 2
name class addr totalsize freesize #allocfrag #freefrag
16354 V c0000001a12a4040 2244608 27536 4211 82
16354_SORT_ V c0000001b3df5040 245760 4704 7 2

name free used name free used
sort 0 34128 sqscb 0 56080
sql 0 80 **srtmembuf 0 204064** **(vs 20384 with 50,000 DS_NONPDQ_QUERY_MEM)**

sqscb info
scb sqscb optofc pdqpriority sqlstats optcompind directives
c0000001ad54a8c0 c0000001a12af030 0 0 0 0 1

Sess SQL Current Iso Lock SQL ISAM F.E.
Id Stmt type Database Lvl Mode ERR ERR Vers Explain
16354 SELECT elstest CR Wait 600 0 0 9.03 Off

Current SQL statement :
select unique b.* from tmp_fids a, name_init b where a.fid = b.fid and a.fid > 0

Fragment Tables

```
SELECT UNIQUE fid, serial_num, total_nos
FROM addridx b
WHERE name  = “JOHN"
  AND b.state = 27
  AND value IN (12345, 98765)
  AND total_nos = 1;

Estimated Cost: 1
Estimated # of Rows Returned: 1

 1) informix.b: INDEX PATH
  (1) Index Keys: state value name total_nos serial_num fid
          (Key-Only)  (Serial, fragments: 26)
     Lower Index Filter: (((informix.b.name = ‘JOHN'  AND informix.b.value = 12345 )
   AND informix.b.state = 27) AND informix.b.total_nos = 1 )

  (2) Index Keys: state value name total_nos serial_num fid
          (Key-Only)  (Serial, fragments: 26)
     Lower Index Filter: (((informix.b.name = ‘JOHN' AND informix.b.value = 98765 )
   AND informix.b.state = 27) AND informix.b.total_nos = 1 )
```

Using External Directives

- External Directives allow you to use directives on SQL statements that cannot be changed.
  - For example, you have an application that you cannot changed the SQL statements in it, but are having an issue with the performance of a specific SQL statement. With the use of external directives, you can override the SQL statement by forcing it to use directives.

Using External Directives – cont'd

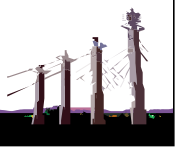
NOTE CAUTION:

- The purpose of external directives is to improve the performance of queries that match the *query* string.
- The use of such directives can potentially slow other queries, if the query optimizer must compare the *query* strings of a large number of active external directives with the text of every SELECT statement.
- For this reason, it is recommended that the DBA not allow the **sysdirectives** table to accumulate more than a few ACTIVE rows.

Use of External Directives – cont'd

- Syntax:

```
SAVE EXTERNAL DIRECTIVES /*+ AVOID_INDEX
 (table1 index1)*/, /*+ FULL(table1) */
      ACTIVE FOR
      SELECT /*+ INDEX( table1 index1 ) */ col1, col2
      FROM table1, table2
      WHERE table1.col1 = table2.col1
```



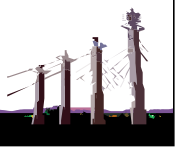
Use of External Directives – cont'd

- How do we enable the use of External Directives?
  - ONCONFIG:
    - EX_DIRECTIVES (0 – OFF, 1 – ON, 2 – ON)
  - Individual sessions external directives can be enabled with the following, all other combinations will have external directives OFF:
    - IFX_EXTDIRECTIVES
      - NOT SET/EX_DIRECTIVES = 2
      - 1 / EX_DIRECTIVES = 1 or 2
      - 0 "NO" External directives no matter what EX_DIRECTIVES is set to

Use of External Directives – cont'd

- The following table shows whether external directives are disabled (OFF) or enabled (ON) for various combinations of valid **IFX_EXTDIRECTIVES** settings on the client system and valid EXT_DIRECTIVES settings on Dynamic Server:

EXT_DIRECTIVES/ IFX_EXTDIRECTIVES	0	1	2
Not Set	OFF	OFF	ON
1	OFF	ON	ON
0	OFF	OFF	OFF



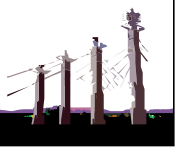
Use of External Directives – cont'd

- Individual sessions can enable or disable external directives by setting **IFX_EXTDIRECTIVES**, as the table on the previous slide shows. Any settings other than 1 or 2 are interpreted as zero, disabling this feature.
- When external directives are enabled, the status of individual external directives is specified by the ACTIVE, INACTIVE, or TEST ONLY keywords. (But only queries on which directives are effective can benefit from external directives.)

Index Self Join (V 11.10)

- New with IDS 11.10, query plans can now use index self-join path. It is a type of index scan that is like a union of many small index scans. It scans smaller ranges instead of one large range for a composite index, based on the filter on non-leading keys.
- Example:

```
SELECT t.*
FROM tst_tbl t
WHERE t.col1 >= 1 AND t.col1 <= 10
AND t.col2 >= 100 AND t.col2 <= 200
AND t.col3 >= 1000 AND t.col3 <= 2000;
Estimated Cost: 120
Estimated # of Rows Returned:  1
  1) informix.t: INDEX PATH
    (1) Index Keys: col1 col2 col3  (Key-Only)  (Serial, fragments: ALL)
        Index Self Join Keys (col1 col2)
          Lower bound: t.col1 >= 1 AND (t.col2 >= 100)
           Upper bound: t.col1 <= 10 AND (t.col2 <= 200)
        Lower Index Filter: (t.col1 = t.col1 AND t.col2 = t.col2) AND t.col3 >= 1000
        Upper Index Filter: t.col3 <= 2000
        Index Key Filters:  (t.col2 <= 200) AND (t.col2 >= 100)
```

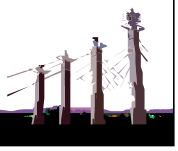


Use of Informix Tools to Analyze/Tune SQL

- Server Studio
- Cobrasonic
- ISPY
- onstats
- Custom Scripts selecting from sysmaster
- IDS 11.10 – Allow to keep history of SQL timings

Summary

- Know your Application in Tuning SQL
- Case Studies
- Tuning for OLTP vs DSS Environments
- Reading sqexplain output and tuning examples
- Understanding Options used to Tune SQL
- Use of Informix Tools to Analyze and Tune SQL



QUESTIONS

Informix SQL Performance Tuning Tips
Session: A12

Jeff Filippi

Integrated Data Consulting, LLC
jeff.filippi@itdataconsulting.com
www.itdataconsulting.com
630-842-3608