



## **Informix SQL Performance Tuning Tips**

**A15 – May 7 2:30**

**Jeff Filippi**



**Integrated Data Consulting, LLC**

[jeff.filippi@itdataconsulting.com](mailto:jeff.filippi@itdataconsulting.com)



# Introduction

- 26 years of working with Informix products
- 22 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 consulting
- IBM Business Partner
- OLTP and Data warehouse systems
- Informix 4, 5, 7, 9, 10, 11.10, 11.50, 11.70, 12.10



# Overview

- Identify Problem SQL Statements
- Tracing SQL in Informix
- Options Available with Set Explain & Reading sqexplain output with tuning examples
- Methods to use for Improving SQL Performance
- Subquery Support for Update/Delete
- External Tables
- Informix 11 & 12 New SQL Features

# 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, OAT and 3<sup>rd</sup> party tools like Server Studio.
  - Look at how many times SQL statements have been executed using SQL Statement Cache (onstat –g ssc)
  - Informix tracing feature (SQLTRACE)
  - Review with developers known problem areas in the application
  - Verify update statistics are current
  - Review what tables/indexes have the most reads



# Use SQL Statement Cache

- `onstat -g ssc`
- Look at SQL's with a large number of executions.
- Saving even a second on a SQL statement that is executed 1 million times can make a difference in performance.

# Use SQL Statement Cache

IBM Informix Dynamic Server Version 11.70.FC7 -- On-Line -- Up 23 days 23:46:38 -- 2530056 Kbytes

## Statement Cache Summary:

```
#lrus currsz maxsz Poolsize #hits nolimit
8 22491472 40960000 11710464 10 0
```

## Statement Cache Entries:

```
lru hash ref_cnt hits flag heap_ptr database user
-----
0 140 0 15 -F bb164020 ntlcom informix
select descr, rowid, seq_nbr from fl_cntrl where uid in ( 'all',
'cschabel' ) and program_name in ( 'all', 'cr_inv_dl' ) and exc_type is not null order by seq_nbr

0 116 0 1011 -F aa23bc20 ntlcom informix
update state_tax set row_status = "V", updt_user_id =? where seq_nbr =? And ( rec_type = 6 or rec_type = 7)

0 207 0 6003 -F a004f820 ntlcom informix
select count ( *) from invoice_state where cde = "CORR" and seq_nbr =?

2 138 0 6244 -F afa9ec20 ntlcom informix
select int_comm, int_comm2, updt_user_id from invoice_cmnts where seq_nbr =? and extend ( updt_dte, year to
second) =
( select max ( extend ( updt_dte, year to second)) from invoice_cmnts where seq_nbr =?)
```

## Review Number of Reads on Table/Index

- Use the table SYSPTPROF to look at the buffer reads, page reads, sequential scans.
  - SELECT tablename[1,25], bufreads, pagreads, isreads, seqscans  
FROM sysmaster:sysptprof  
ORDER BY 2 desc

## Example - Reads

tabname	bufreads	pagreads	isreads
trnsit_1	-2122091061	429	1630786736
trnsit_1	-812314372	3162	-678524115
trnsit_1	-110705308	233	-390409810
im_p_route_1	1806427782	247	865918944
ed_rl_event	1749246222	23550	1709746386
loc_sup_data	1479941490	39	1108625557
ed_rl_event_3	1186682507	2668713	789739458
460_4902	1125173161	1003575	373042018
ed_rl_event_4	893520660	25725	886704767
im_mv_event	870108889	30477208	780365364



# New Index Added

tabname	buffers	pagereads	isreads
140_409	-845911921	0	1722690950
cntrct_num	1297728722	1	2868878
221_1360	812752007	0	406226191
.....			
<b>trnsit</b>	<b>17215024</b>	<b>22</b>	<b>12007375</b>
<b>trnsit_ix1</b>	<b>15638629</b>	<b>106</b>	<b>12898627</b>
<b>im_p_route_1</b>	<b>23045</b>	<b>347</b>	<b>12904</b>

# Tracing SQL in Informix

- There are more ways to find information to tune in Informix 11 utilizing the tracing of SQL
  - onconfig parameter: SQLTRACE
  - SQL Admin API
  - Ability to trace by database (11.50XC3)
  - Ability to save trace buffer (11.50XC4)
  - Open Admin Tool (OAT)

# Tracing SQL in Informix

- There are a couple ways to turn tracing on in Informix
  - 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)
  - Open Admin Tool (OAT)

# Tracing SQL in Informix

- Improved SQL tracing with the SQL Admin API in Informix 11.50FC3
  - You can use these new commands to manage SQL tracing by databases.
    - SET SQL TRACING DATABASE ADD {Database}
    - CLEAR
    - LIST
    - REMOVE {Database}
  - You can also suspend and resume all tracing at the server without de-allocating any resources.
    - SET SQL TRACING SUSPEND/RESUME

# Tracing SQL in Informix

- Here is how you 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 Informix

**onstat -g his**

IBM Informix Dynamic Server Version 11.70.FC7 -- On-Line -- Up 25 days 23:44:15 -- 2530056  
Kbytes

Statement history:

Trace Level	Low
Trace Mode	Global
Number of traces	3000
Current Stmt ID	939412632
Trace Buffer size	2024
Duration of buffer	8241 Seconds
Trace Flags	0x00001611
Control Block	9df53018

# Tracing SQL in Informix

Statement # 94653656: @ 9df68cb0

Database: 0x1700002

Statement text:

```
SELECT * FROM invc_state WHERE seq_nbr = ?
```

Iterator/Explain

=====

ID	Left	Right	Est Cost	Est Rows	Num Rows	Type
1	0	0	26	4	6	Index Scan

Statement information:

Sess_id	User_id	Stmt Type	Finish Time	Run Time
7640	1001	SELECT	18:44:20	0.0006

# Tracing SQL in Informix

## Statement Statistics:

Page Read	Buffer Read	Read % Cache	Buffer IDX Read	Page Write	Buffer Write	Write % Cache
0	17	100.00	0	0	0	0.00

Lock Requests	Lock Waits	LK Wait Time (S)	Log Space	Num Sorts	Disk Sorts	Memory Sorts
13	0	0.0000	0.000 B	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
7294	6.6040	0.0009	0.0015	0.000000	0.000000	10869.5652

Estimated Cost	Estimated Rows	Actual Rows	SQL Error	ISAM Error	Isolation Level	SQL Memory
26	4	6	0	0	LC	13416



# Tracing in Informix

- 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_totalltime > 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 identify which part of the query plan took the most time to run.

## Tracing in Informix

- You can run the following SQL statement to get the SQL for the ones that have a higher run time than ½ a second.

```
select sql_runtime, sql_statement  
from sysmaster:sysssqltrace  
where sql_runtime > .5  
order by 1 desc
```

# Tracing in Informix

- Here is what the “syssqltrace” table looks like

```
sql_id      9894804
sql_address 13746521704
sql_sid     26646
sql_uid     668
sql_stmttype 2
sql_stmtname SELECT
sql_finishtime 1396011341
sql_begintxtime 301590260
sql_runtime 4.7
sql_pgreads 0
sql_bfreads 5
sql_rdcache 100.000000000000
sql_bfidxreads 0
```

# Tracing in Informix

```
sql_pgwrites    0
sql_bfwrites   0
sql_wrcache    0.00
sql_lockreq    1
sql_lockwaits  0
sql_lockwtime  0.00
sql_logspace   0
sql_sorttotal  0
sql_sortdisk   0
sql_sortmem    0
sql_executions 1
sql_totalltime 4.7
sql_avgtime    4.7
sql_maxtime    4.7
sql_numioawaits 0
sql_avgioawaits 0.00
sql_totalioawaits 0.00
sql_rowspersec 20898.94078138
```

# Tracing in Informix

```
sql_estcost    23
sql_estrows    46
sql_actualrows 1
sql_sqlerror   0
sql_isamerror  0
sql_isollevel  1
sql_sqlmemory  24200
sql_numiterators 1
sql_database   ir_live2
sql_numtables  4
sql_tablelist  systables
sql_statement   select owner,tabname,tabtype,tabid from informix.systables
sql_stmtlen    117
sql_stmthash   206750675
sql_pdq        0
sql_num_hvars  0
sql_dbspartnum 10488544
sql_aqt        None
sql_aqtinfo    -26500
```

# Tracing in Informix

- Starting in 11.50XC4 you can now save the history of the SQL tracing information. **NOTE:** Caution when using this, it can use a lot of space very quickly. I had a customer fill a 20 gig dbspace in 24 hours.
- In the Scheduler there is a new task “Save SQL Trace”.
- Information is saved in the following tables in the sysadmin database:
  - mon\_syssqltrace (SQL Statement text and profile info)
  - mon\_syssqltrace\_info (SQL Statement tracing setup info)
  - mon\_sqltrace\_iter (SQL Statement iterators)



# Tracing in Informix

- Here is an alternative to saving ALL tracing information.
- Create a task which saves SQL trace information for SQL's that have a run time of greater than 10 seconds.
- This allows you to still trace SQL statements, but only show you the really long running SQL statements

# Tracing in Informix

- Create a Table to save the SQL Trace information
- Create a new dbspace to put the table in so that if it does fill up a dbspace it does not affect any other processes.

```
create raw table "informix".save_sqltrace  
(  
  date_time datetime year to second,  
  sql_id int8,  
  sql_runtime float,  
  sql_sid int8,  
  sql_uid int8,  
  sql_statement char(11000),  
  sql_database char(30)  
) in sqltrace extent size 100000 next size 50000 lock mode row;
```

```
create index "informix".idx_savesql1 on "informix".save_sqltrace (date_time) in sqltrace;  
create index "informix".idx_savesql2 on "informix".save_sqltrace (sql_runtime) in sqltrace;  
create index "informix".idx_savesql3 on "informix".save_sqltrace (sql_id) in sqltrace;
```



# Tracing in Informix

- Here is the information that needs to be inserted into the “ph\_task” table to activate the task.
- In my case I was running it every minute.
- You will want to see what the shortest time that your trace buffer is and make the frequency the task runs smaller than that.

```
0|save_trace|Saves SQL Trace when run time greater than set  
value.|TASK|9251|||sysadmin|insert into save_sqltrace select  
current, sql_id, sql_runtime, sql_finishtime, sql_sid, sql_uid,  
sql_statement, sql_database from sysmaster:sysssqltrace where  
(sql_runtime > 5 and (sql_finishtime > (select max(sql_finishtime)  
from save_sqltrace)) or (sql_runtime > 5 and ((select count(*)  
from save_sqltrace) = 0)));| 30 00:00:00|00:00:00|| 0 00:01:00|2014-03-  
27 14:54:17|9237|0|t|t|t|t|t|t|t|400|PERFORMANCE|t|0|
```



## Use Open Admin Tool (OAT)

- Use the Open Admin Tool to find slow SQL statements.
  - Under “System Reports” there is an option to show the Slowest SQL Statements.

# Using OAT

http://laptop-itdata-18080/openadmin/index.php - Windows Internet Explorer

http://laptop-itdata-18080/openadmin/index.php

File Edit View Favorites Tools Help

Search web...

http://laptop-itdata-18080/openadmin/index.php

## Slowest 5 SQL Statements

SQL Profile

```

    graph TD
      1[1.Group] --> 2[2.Nested Join]
      2 --> 3[3.Index Scan]
      2 --> 4[4.Index Scan]
      style 1 fill:#add8e6,stroke:#000,stroke-width:1px
      style 2 fill:#add8e6,stroke:#000,stroke-width:1px
      style 3 fill:#add8e6,stroke:#000,stroke-width:1px
      style 4 fill:#add8e6,stroke:#000,stroke-width:1px
    
```

Session ID	User ID	Statement Type	PDQ	Statement Completion Time	Response Time
2740	5437	SELECT	NA	2009-03-09 08:26:04	0.4323930 Sec
<b>Database</b>					
<b>Statement</b> "select count (*) from invc_state_cur a where a.top_stack > 0 and a.code = 'LOADED' and a.arc_nbr in ( select b.opt_arc from cust_verify_opt b where a.arc_nbr = b.opt_arc and b.opt_nbr =?)"					

Statement Statistics							
Page Reads	Buffer Reads	Reads Cache	Data Buffer Reads	Index Buffer Reads	Page Writes	Buffer Writes	Writes Cache
0	33441	100.00 %	33441	0	0	0	0.00 %
Lock Requests	# Lock Waits	Lock Wait Time (S)	Log Space	Disk Sorts	Memory Sorts	Number of Tables	Number of Iterators
25847	0	0	0.000 B	0	0	0	4
Total Executions	Total Executions Time (S)	Average Execution Time (S)	Maximum Execution Time (S)	Number of IO Wait	IO Wait Time (S)	Average IO Wait (S)	Average Rows/Second
1	0.43239	0.43239	0.43239	0	0.00000	0.00000	2.31271
Estimated Cost	Estimated Rows	Actual Rows	SQL Error	ISAM Error	Isolation Level	SQL Memory	
1749	1	1	0	0	18	17.9 KB	

SQL Profile

Done

Local intranet | Protected Mode: On 100%

SUSE Linux Enterpris... http://laptop-itdata... C:\Program Files JB...

100% 8:27 AM

# Options Available with SQEXPLAIN

- Optimizer Directives – AVOID\_EXECUTE
  - Introduced in Informix 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

# Options Available with Set Explain

- Set Explain Enhancements

- Another improvement with Informix 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 Informix 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.

# 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 Informix 11 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.

# Set Explain Output

- Add “set explain on” before the statement you want to examine
- Starting in Informix 11.10 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

- **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
  - Using “in” causes sequential scans
  - Use of Directives
  - Use of substrings in queries
  - Use of functions in queries
  - Using a better index (Creation of new index)

# Functions cause 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

# 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

# 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;
```

# Criteria used to select from views 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 = '05-04-CVC-6-KINS'  
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: 8425

Estimated # of Rows Returned: 1

Temporary Files Required For: Order By Group By

1) **sysadm.ps\_cm\_deplete: SEQUENTIAL SCAN**

2) sysadm.ps\_cm\_receipts: INDEX PATH

(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.dt\_timestamp = sysadm.ps\_cm\_deplete.cm\_dt\_timestamp AND  
sysadm.ps\_cm\_receipts.cm\_dt\_timestamp\_a = sysadm.ps\_cm\_deplete.cm\_dt\_timestamp\_a) AND  
sysadm.ps\_cm\_receipts.inv\_item\_id = sysadm.ps\_cm\_deplete.inv\_item\_id) AND sysadm.ps\_cm\_receipts.seq\_nbr =  
sysadm.ps\_cm\_deplete.cm\_seq\_nbr) 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

# 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



# Using “in” causes sequential scans

QUERY:

```
-----  
SELECT inv3_invno  
FROM inv3  
WHERE 448050 IN (inv3_physcode,inv3_altphys1,inv3_altphys2)
```

Estimated Cost: 157852

Estimated # of Rows Returned: 566880

1) informix.cus3: **SEQUENTIAL SCAN**

Filters: 448050 IN (informix.inv3.inv3\_physcode , informix.inv3.inv3\_altphys1 , informix.inv3.inv3\_altphys2 )

Query statistics:

-----  
Table map :

-----  
Internal name    Table name  
-----

	t1	inv3				
type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t1	21	566880	2096652	00:01.26	157852

# Resolution to Criteria used for Using “in” causes sequential scans

- First I create two new indexes
  - create index `ixinv3_altphys1` on `inv3(inv3_altphys1)`
  - create index `ixinv3_altphys2` on `inv3(inv3_altphys2)`
- Then I changed the query to use “or” instead of “in”

# Resolution to Criteria used for Using “in” causes sequential scans

QUERY:

-----

```
SELECT inv3_invno  
FROM inv3  
WHERE (inv3_physcode = 448050  
or inv3_altphys1 = 448050  
or inv3_altphys2 = 448050)
```

Estimated Cost: 13

Estimated # of Rows Returned: 9

1) informix.inv3: INDEX PATH

- (1) Index Name: informix.ixinv3\_physcode  
Index Keys: inv3\_physcode (Serial, fragments: ALL)  
Lower Index Filter: informix.inv3.inv3\_physcode = 448050
- (2) Index Name: informix.ixinv3\_altphys2  
Index Keys: inv3\_altphys2 (Serial, fragments: ALL)  
Lower Index Filter: informix.inv3.inv3\_altphys2 = 448050
- (3) Index Name: informix.ixinv3\_altphys1  
Index Keys: inv3\_altphys1 (Serial, fragments: ALL)  
Lower Index Filter: informix.inv3.inv3\_altphys1 = 448050

# Resolution to Criteria used for Using “in” causes sequential scans

Query statistics:

-----

Table map :

-----

Internal name    Table name

-----

t1            inv3

**type    table rows\_prod est\_rows rows\_scan time est\_cost**

-----

**scan    t1            21            9            21            00:00.24    13**

# 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 ;
```

# 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

# Use of Directives in Queries

- 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 Directives in Queries

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



# Use of Directives in Queries

## 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 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.acnl: INDEX PATH

Filters: (informix.acnl.production\_id = '12334' AND SUBSTR  
(informix.acnl.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.acnl.business\_unit = 'ABCDE'

# Resolution for 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.acnl: INDEX PATH

**(1) Index Keys: business\_unit production\_id account (Key-First) (Serial, fragments: ALL)**  
**Lower Index Filter: (informix.acnl.production\_id = '12334' AND informix.acnl.business\_unit = 'ABCDE')**  
**Key-First Filters: (((informix.acnl.account MATCHES '085\*' OR informix.acnl.account MATCHES '334\*') OR informix.acnl.account MATCHES '072\*'))**

# Use of Functions in Queries cause specific Indexes not to 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

# Use of Functions in Queries cause 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

# 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

# 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'



# 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')

# Subquery – Slows Query

## ORIGINAL QUERY:

```
SELECT *
FROM doc
WHERE doc_date BETWEEN '2016-03-15 09:00' AND '2016-03-31 09:00'
      AND doc_num IN (SELECT DISTINCT r_doc_num
                      FROM rev
                      WHERE re_op_id= 'johnsmith')
ORDER BY doc_num DESC;
```

# Subquery – Slows Query

## CHANGED QUERY:

```
SELECT *
FROM doc
WHERE doc_date BETWEEN '2016-02-15 08:42' AND '2016-02-29 08:42'
      AND doc_num IN (SELECT DISTINCT r_doc_num
                      FROM rev
                      WHERE re_op_id= 'johnsmith'
                        AND doc_num = r_doc_num)
ORDER BY doc_num DESC;
```



# Methods for Improving SQL Performance

- Utilize “UNIONS” when you have “OR” in where clause
- Utilize temp tables when have large “IN” 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 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 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.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.email_template_id = '2152';
```

# Utilize Temp Tables with “IN” for SELECT

QUERY: (OPTIMIZATION TIMESTAMP: 03-31-2015 09:28:31)

```
-----  
select *  
from bike_cycle  
where bike_serial in (5259604,12127443,12778188,  
23600326,23600442,23600600,23600784,23600791,  
24893077,24893080,24893082,24893084,24893088,  
24893092,24893096,24893097,24893100,24893103,.....)
```

Estimated Cost: 14722

Estimated # of Rows Returned: 36240

1) informix.bike\_cycle: INDEX PATH

(1) Index Name: informix.accy\_idx\_1a

Index Keys: bike\_serial reported\_dt (Serial, fragments: ALL)

Lower Index Filter: informix.bike\_cycle.bike\_serial = 5259604

(2) Index Name: informix.accy\_idx\_1a

Index Keys: bike\_serial reported\_dt (Serial, fragments: ALL)

Lower Index Filter: informix.bike\_cycle.bike\_serial = 12127443

Query statistics:

-----  
Table map :

-----  
Internal name    Table name  
-----

t1            bike\_cycle

type    table rows\_prod est\_rows rows\_scan time    est\_cost  
-----

scan    t1    3788    36176    3788    00:18.86    19527

# Utilize Temp Tables with “IN” Use “Temp Table”

QUERY: (OPTIMIZATION TIMESTAMP: 04-08-2015 14:35:48)

-----  
select \* from tst

Estimated Cost: 1

Estimated # of Rows Returned: 1

1) informix.tst: SEQUENTIAL SCAN

Query statistics:

-----  
Table map :

-----  
Internal name    Table name  
-----

t1            tst

type    table rows\_ins    time  
-----

insert t1    1936    00:00.00

# Utilize Temp Tables with “IN”

CREATE INDEX:

=====

Index:           tst\_x1 on informix.tst

STATISTICS CREATED AUTOMATICALLY:

Column Distribution for:           informix.tst.ac

Mode:           HIGH

Number of Bins:   102           Bin size:  19.0

Sort data:        0.0 MB

Completed building distribution in:  0 minutes 0 seconds

Index LOW statistics gathered during index build

QUERY: (OPTIMIZATION TIMESTAMP: 04-08-2015 14:35:48)

-----

```
select * from bike_cycle where bike_serial in (  
select ac from tst  
)
```

Estimated Cost: 12459

Estimated # of Rows Returned: 24726

1) informix.bike\_cycle: INDEX PATH

(1) Index Name: informix.accy\_idx\_1a

Index Keys: bike\_serial reported\_dt (Serial, fragments: ALL)

Lower Index Filter: informix.bike\_cycle.bike\_serial = ANY <subquery>



# Utilize Temp Tables with “IN”

Subquery:

-----

Estimated Cost: 64

Estimated # of Rows Returned: 1936

1) informix.tst: INDEX PATH

(1) Index Name: tst\_x1

Index Keys: ac (Key-Only) (Serial, fragments: ALL)

Query statistics:

-----

Table map :

-----

Internal name    Table name

-----

t1            bike\_cycle

type    table   rows\_prod   est\_rows   rows\_scan   time    est\_cost

-----

scan   t1    3788    24726    3788    00:00.04   12459

# Utilize Temp Tables with “IN”

Subquery statistics:

-----

Table map :

-----

Internal name    Table name

-----

t1            tst

type    table rows\_prod est\_rows rows\_scan time        est\_cost

-----

**scan    t1    1936    1936    1936    00:00.00    64**

type    rows\_sort est\_rows rows\_cons time

-----

**sort    1936    0    1936    00:00.00**

# 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.pull and 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 PDQ Priority

**SET PDQPRIORITY 100;**

```
SELECT acct, pc, cntrl_wd, mfree, uauto, salestype
FROM vid_tran
WHERE substr(acct,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 sqlxec  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 produ 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

# Utilize DS\_NONPDQ\_QUERY\_MEM

In one case at a client by increasing

DS\_NONPDQ\_QUERY\_MEM

From 128 to 50000

The query time for a specific query went from 2.5 seconds down to .25 seconds.

Also the wait on I/O from “top” output went from 2% to basically 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 )



# Subquery Support (Update/Delete)

- The FROM clause of a subquery in the WHERE clause of the UPDATE statement can specify as a data source the same table or view that the Table Options clause of the UPDATE statement specifies.
- UPDATE operations with subqueries that reference the same table object are supported only if all of the following conditions are true:
  - The subquery either returns a single row,
  - Has no correlated column references.
- No SPL routine in the subquery can reference the same table that UPDATE is modifying.

# Subquery Support (Update/Delete)

- Example: Updates the stock table by reducing the unit\_price value by 5% for a subset of prices. The WHERE clause specifies which prices to reduce by applying the IN operator to the rows returned by a subquery that selects only the rows of the stock table where the unit\_price value is greater than 50:

**UPDATE stock**

**SET unit\_price = unit\_price \* 0.95**

**WHERE unit\_price IN (SELECT unit\_price**

**FROM stock**

**WHERE unit\_price > 50);**

# External Tables

- Ways to use External Tables
  - Creating external tables
    - » Create table ext\_xyz (col1 integer, col2 char(20))  
USING (DATAFILES (“DISK:/data/xyz.unl”),  
FORMAT “DELIMITED”)
    - » Create table ext\_xyz SAMEAS xyz  
USING (DATAFILES (“DISK:/data/xyz.unl”),  
FORMAT “DELIMITED”)

# External Tables

- Ways to use External Tables
  - Load data from external table into database table
    - » `insert into xyz select * from ext_xyz;`
  - Unload data to external table
    - » `insert into ext_xyz select * from xyz;`

# External Tables

- Using PIPE feature

- When you do not want to unload to a flat file, but pipe the data to another server/table you can use the PIPE feature.

- Here is what the create table statement looks like:

- » Create external table ext\_xyz SAMEAS xyz  
USING ( DATAFILES("PIPE:/data/ext\_xyz1"))

- To improve performance using PIPES make sure that there are enough FIFO VP's defined. The database server uses one FIFO VP for each named pipe that specify in the DATAFILES clause.

- » To add a FIFO VP – onmode -p + {# to add} FIFO



# New Features – Optimizer Directives

## – 11.70xC1 – NEW OPTIMIZER DIRECTIVES

- Query optimizer support for star-schema and snowflake-schema queries. A primary key column in each dimension table must correspond to a foreign key in the fact table.
  - New optimizer directives have been added:
    - » STAR\_JOIN, FACT, AVOID\_STAR\_JOIN & AVOID\_FACT
    - » Can also enable with SET OPTIMIZATION



# New Features – Optimizer Directives

## – 11.70xC1 – NEW OPTIMIZER DIRECTIVES

- Syntax support for DDL statement with IF [NOT] EXISTS:
  - Can include “IF [NOT] EXISTS” condition to SQL statements that create a database object or a database.



# New Features – Optimizer Directives

## – 11.70xC2

- Table and column Aliases in DML statements.
  - SELECT – statements and subqueries can declare an alias in the projection clause for columns in the select list.
  - DELETE / UPDATE – can declare an alias for a local or remote target table.



# Informix 12 SQL Features

- **Faster ANSI join queries**
  - ANSI outer join queries that have equality joins can run faster because the Informix optimizer now uses either a hash join or a nested loop on a cost basis. In earlier releases, Informix used only nested loop joins in ANSI outer joins.
- **Temporary table projection optimization for views and derived tables**
  - Applications and analytic tools can define a query in which a derived table contains multiple views joined with base tables, potentially including hundreds of columns. The database server materializes this query in a system-generated temporary table. The parent query, however, might project only a few columns.
  - The database server creates internally generated temporary tables that include only the columns that are specified in the Projection list, the WHERE clause, the ORDER BY clause, and in other clauses of the immediate parent query. By excluding unnecessary columns from the temporary table, the database server uses storage resources efficiently and avoids I/O operations on the columns that do not contribute to the query result.



# Summary

- Identify Problem SQL Statements
- Tracing SQL in Informix
- Options Available with Set Explain & Reading sqexplain output with tuning examples
- Methods to use for Improving SQL performance
- Subquery Support for Update/Delete
- External Tables
- Informix 11 & 12 New SQL Features



# Questions?

Jeff Filippi

Integrated Data Consulting, LLC

[jeff.filippi@itdataconsulting.com](mailto:jeff.filippi@itdataconsulting.com)

[www.itdataconsulting.com](http://www.itdataconsulting.com)