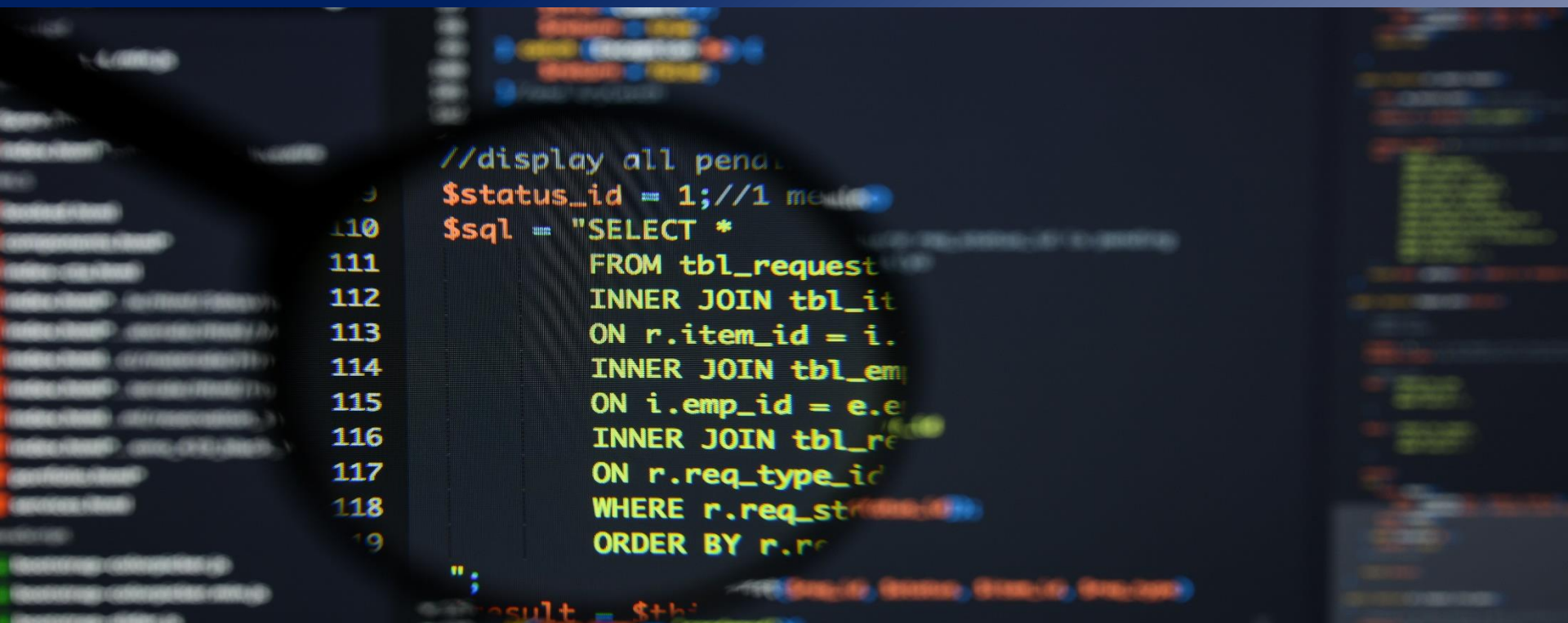


# Explaining Explain Plans



**XDB**  
SYSTEMS

*Mike Walker*  
*mike@xdbsystems.com*

8 December 2022

# SQL Query Execution Plans

- The Query Plan shows us exactly *how* a SQL statement is being run by the engine
  - Indexes being used
  - Order that the tables are read
  - Joins
- We can use these plans to help tune a query
  - Find out the slow parts of a query
  - Missing or poor index choices
  - Out of date statistics

# Query Plan

QUERY: (OPTIMIZATION TIMESTAMP: 12-04-2022 11:02:23)

-----

select \* from snapshot

Estimated Cost: 182920

Estimated # of Rows Returned: 2560531

1) informix.snapshot: SEQUENTIAL SCAN

Query &  
Query Plan

Query statistics:

-----

Table map :

-----

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

-----

t1	snapshot
----	----------

type	table	rows_prod	est_rows	rows_scan	time	est_cost
------	-------	-----------	----------	-----------	------	----------

-----

scan	t1	2560709	2560531	2560709	00:01.32	182921
------	----	---------	---------	---------	----------	--------

Query  
Statistics

# SET EXPLAIN

Use the “SET EXPLAIN” SQL statement to start/stop the output of explain plans:

- **SET EXPLAIN ON**: write explain plans to a file for the SQL statements that follow
- **SET EXPLAIN OFF**: turn off explain plans
- **SET EXPLAIN ON AVOID\_EXECUTE**: Produce explain plan *without* running the SQL
- **SET EXPLAIN FILE TO “<filename>”**: Write explain file to the specified file

# SET EXPLAIN

- SET EXPLAIN ON / SET EXPLAIN OFF:

```
SET EXPLAIN ON;
```

```
SELECT * FROM x WHERE y = 10;
```

```
SET EXPLAIN OFF;
```

- By default, the query plan is written to the file:  
`sqexplain.out`
- File is created in the current directory (UNIX)
- If use client app, the file will be in home directory of the user that SQL was executed as
- File will be appended to each time more SQL is executed

# SET EXPLAIN

slow1.sql:

```
set explain file to "slow1.exp";
```

```
output to /dev/null
```

```
select c.customer_num, o.order_num  
from customer c, orders o  
where c.customer_num = o.customer_num  
      and c.company = "Play Ball!"  
order by 2;
```

```
time dbaccess -e stores_demo slow1.sql > slow1.out 2>&1 &
```

```
-rw-rw-rw- 1 informix informix 1563 Dec  4 11:07 slow1.exp
```

# SET EXPLAIN

slow1.exp

QUERY: (OPTIMIZATION TIMESTAMP: 12-04-2022 11:07:00)

-----

```
select c.customer_num, o.order_num
from customer c, orders o
where c.customer_num = o.customer_num
      and c.company = "Play Ball!"
order by 2
```

Estimated Cost: 6

Estimated # of Rows Returned: 2

Temporary Files Required For: Order By

1) informix.c: SEQUENTIAL SCAN

Filters: informix.c.company = 'Play Ball!'

2) informix.o: INDEX PATH

(1) Index Name: informix. 102\_4

Index Keys: customer\_num (Serial, fragments: ALL)

Lower Index Filter: informix.c.customer\_num = informix.o.customer\_num

NESTED LOOP JOIN

# SET EXPLAIN

Query statistics:

slow1.exp (Continued)

Table map :

-----	
Internal name	Table name
-----	
t1	c
t2	o

Query Statistics will be shown at the  
end of the plan  
[ EXPLAIN\_STAT=1 in ONCONFIG ]

type	table	rows_prod	est_rows	rows_scan	time	est_cost
-----						
scan	t1	1	3	28	00:00.00	4

type	table	rows_prod	est_rows	rows_scan	time	est_cost
-----						
scan	t2	4	23	4	00:00.00	0

type	rows_prod	est_rows	time	est_cost
-----				
nljoin	4	3	00:00.00	6

type	rows_sort	est_rows	rows_cons	time	est_cost
-----					
sort	4	3	4	00:00.00	0



# SET EXPLAIN

For long running SQL or for Insert, Update or Delete operations, use “AVOID\_EXECUTE” to get the explain plan ***without*** running the SQL:

slow2.sql:

```
set explain file to "slow2.exp";  
set explain on avoid_execute;
```

```
update orders  
set ship_instruct = null  
where customer_num = 104;
```

# SET EXPLAIN

```
dbaccess -e stores_demo slow2.sql
```

Database selected.

```
set explain file to "slow2.exp";  
Explain set.
```

```
set explain on avoid_execute;  
Explain set.
```

```
update orders  
set ship_instruct = null  
where customer_num = 104;  
0 row(s) updated.
```

Warning! avoid\_execute has been set

Database closed.

If use AVOID\_EXECUTE will  
NOT see the Query Statistics  
in the Explain Plan

# Dynamic Query Plans

```
onmode -Y <sid> <0|1|2> [filename] Set or unset dynamic explain
0=off 1=plan + statistics on 2=only plan on
filename is a valid argument only when setting the
dynamic explain or dynamic explain statistics on
```

```
onmode -Y 10563 1
```

Set Dynamic Explain for Session 10563

```
onstat -g ses
```

```
IBM Informix Dynamic Server Version 12.10.FC5AEE -- On-Line -- Up 1 days 12:01:36 --
2947104 Kbytes
```

session					#RSAM	total	used	dynamic
id	user	tty	pid	hostname	threads	memory	memory	explain
10657	informix	-	0	-	0	16384	12480	off
10653	informix	-	0	-	0	16384	12480	off
<b>10563</b>	<b>informix</b>	<b>2</b>	<b>4243</b>	<b>apollo</b>	<b>1</b>	<b>73728</b>	<b>64480</b>	<b>on</b>
10028	informix	-	0	apollo	1	335872	321728	off
10011	informix	-	0	apollo	1	241664	100072	off
44	informix	-	0	-	1	626688	472280	off
43	informix	-	0	-	1	626688	471576	off
42	informix	-	0	-	1	618496	494080	off
41	informix	-	0	-	1	102400	86784	off

# Dynamic Query Plans

Explain plan written to a file with the SID in the name:

```
-rw-rw-rw- 1 informix informix      573 Apr  7 11:17 sqexplain.out.10563
```

- Using “onmode -Y” will not produce anything until the *next* statement runs – so no good for getting the explain plan for a single, long running statement
- Limited value if prepared SQL is being executed
- Problems stopping the explain plan
- Capture the SQL to a file instead, and get the explain plan for that...

# Anatomy of a Query Plan

Create View (if applicable)

Query SQL

Cost/Rows Returned/Temp Tables/Directives

Table 1 : Name & Access Method

Table 1 : Filters

Table 1 : Index Info

Table 2 : Name & Access Method

Table 2 : Filters

Table 2 : Index Info

Table 1 & 2 : Join Method

Repeated  
for other  
tables

Subqueries

Query Statistics (if enabled)

# Query Plans

QUERY: (OPTIMIZATION TIMESTAMP: 04-09-2017 07:50:47)

-----

```
select c.customer_num, o.order_num
from customer c, orders o
where c.customer_num = o.customer_num
      and c.company = "Play Ball!"
order by 2
```

Estimated Cost: 6

Estimated # of Rows Returned: 2

Temporary Files Required For: Order By

Query SQL

Cost/Rows Returned/Temp  
Files/Directives

Table 1 : Name & Access Method

1) informix.c: SEQUENTIAL SCAN

Filters: informix.c.company = 'Play Ball!'

Table 1 : Filters

2) informix.o: INDEX PATH

Table 2 : Name & Access Method

(1) Index Name: informix. 102\_4

Index Keys: customer\_num (Serial, fragments: ALL)

Lower Index Filter: informix.c.customer\_num =

Table 2 : Index  
Info

informix.o.customer\_num

NESTED LOOP JOIN

Table 1 & 2 : Join Method

Order  
tables are  
accessed

# Query Plans - Statistics

Query statistics:

-----

Table map :

-----	
Internal name	Table name
-----	
t1	c
t2	o



Table Mapping

type	table	rows_prod	est_rows	rows_scan	time	est_cost
-----						
scan	t1	1	3	28	00:00.00	4

Table 1 Query Stats

type	table	rows_prod	est_rows	rows_scan	time	est_cost
-----						
scan	t2	4	23	4	00:00.00	0

Table 2 Query Stats

type	rows_prod	est_rows	time	est_cost
-----				
nljoin	4	3	00:00.00	6

Table 1 & 2 : Join Stats

type	rows_sort	est_rows	rows_cons	time	est_cost
-----					
sort	4	3	4	00:00.00	0

Sort Stats

# Query Plans - Statistics

Query statistics:

Table map :

Internal name	Table name
t1	bigtab
t2	c

**“scan”** is shown for table scans and index scans

**Retrieved** minus **Produced** is the result of filters (usually)

Operation		Num Records Produced		Num Records Retrieved	Duration	
type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t1	11	11	175000	00:02.23	63585
type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t2	297	21	8789	00:00.02	43
		Rows Returned		Duration		
type	rows_prod	est_rows	time	est_cost		
nljoin	297	232	00:02.26	64058		
				Total Time		
type	rows_sort	est_rows	rows_cons	time	est_cost	
sort	297	232	297	00:02.26	58	

- **“time”** will show you the slowest part of the query
- Useful when query is sub-second (ignores overhead)
- Joins show cumulative time



# Query Plans - Statistics

Internal name		Table name				
t1		l				
t2		p				
t3		c				
type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t1	830895	40737	4140946	00:03.97	535572
type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t2	830895	21310	830895	00:06.67	1
type	rows_prod	est_rows	time	est_cost		
nljoin	830895	39355	00:10.97	559218		
type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t3	0	8	830895	00:06.07	0
type	rows_prod	est_rows	time	est_cost		
nljoin	0	41460	00:17.22	565627		
type	rows_sort	est_rows	rows_cons	time	est_cost	
sort	0	41460	0	00:17.22	19701	

Time for each  
“scan” is for that  
read

Joins and sort  
are cumulative

# Access Methods

How is a table is read:

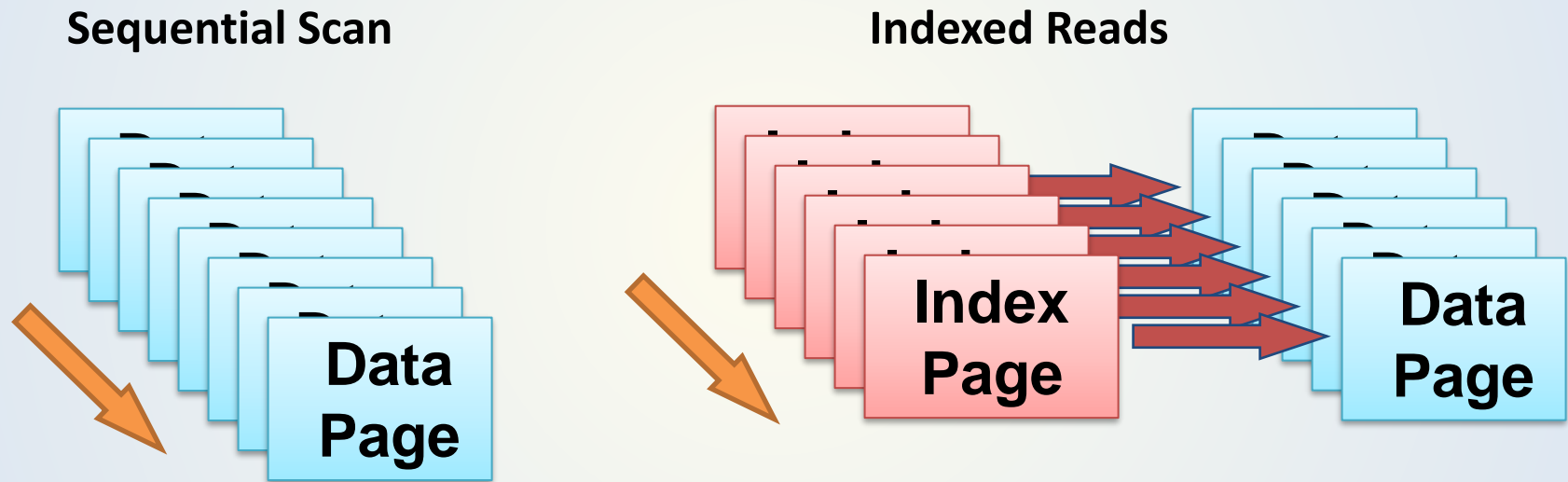
- Sequential Scan
  - Full table scan
- Index Path
  - Table is read via an index

# Sequential Scans

- If a Query Plan contains a Sequential Scan, all rows of the table are read (before any filter is applied)
- May not be bad
  - If the table is small
  - If most of the rows read from the table are needed, then it may be okay
  - Consider that many indexed reads of data can be costly because of the read of the index, *plus* the read of the data page

# Sequential Scans

A Scan of all Data Pages *may* be faster than lots of Indexed Reads



But it depends on how many rows are *actually needed*

A scan of a large table can trash the cache

# Sequential Scans

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

Filters: (informix.bigtab.a <= 20000 AND informix.bigtab.a >= 10 )

type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t1	19991	19991	175000	00:02.43	63585

Rows  
Produced

Rows Read

Filter is applied *after* each record is read  
19,991 records matched the filter condition  
**175,000 – 19,991 = 155,009 records were discarded**

Recommend an index on bigtab(a)

If rows\_scan ~ rows\_prod index *may* not help

# Sequential Scans

Why am I getting a scan on a table with an index?

- Order of columns in an index (leading column)
- Functions applied to the column (TRIM, DATE, UPPER, etc)
- Data type in query != table column datatype
- Statistics out of date

# Index Read

```
1) informix.s: INDEX PATH
```

```
(1) Index Name: informix.snapshot_idx5  
    Index Keys: instance_id snapshot_id (desc)      (Serial,  
fragments: ALL)  
    Lower Index Filter: (informix.s.instance_id = 38 AND  
informix.s.snapshot_id = <subquery> )
```

- Index was used to access table “s” (alias)
- Name of the index used to retrieve rows from the table (snapshot\_idx5)
- Columns in the index (instance\_id, snapshot\_id)
- The index was defined as descending on snapshot\_id
- Serial, fragments ALL indicates PDQ is not in use and fragment elimination not used
- Value of instance\_id was passed to the query as a literal
- snapshot\_id is obtained from a subquery

# Index Read

```
select b
from bigtab
where a between 10 and 20000
```

```
1) informix.bigtab: INDEX PATH
```

```
(1) Index Name: informix.bigtab_idx
    Index Keys: a      (Parallel, fragments: ALL)
    Lower Index Filter: informix.bigtab.a >= 10
    Upper Index Filter: informix.bigtab.a <= 20000
```

```
.
.
type      table  rows_prod  est_rows  rows_scan  time      est_cost
-----
scan      t1      19991      19991      19991      00:00.42  7396
```

- “between” clause is using an index on column “a”
- The index leaf nodes can be scanned within the lower and upper limits
- Query Statistics show that all rows read were used



# Index Filters

An index filter is applied *after* each record is read

3) informix.s: INDEX PATH

Filters: informix.s.eff\_dt > 07/29/2020

(1) Index Name: informix.customer\_x01

Index Keys: id (Serial, fragments: ALL)

Lower Index Filter: informix.s.id = informix.sc.id

type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t3	1183739	41762640	7604110	03:02.00	1

Rows  
Produced

Rows Read

Including eff\_dt at the end of the index would improve performance

# Index Filters

## 3) informix.c: INDEX PATH

Filters: `informix.c.notify_email = 'Y'`

(1) Index Name: `informix. 338_1436`

Index Keys: `category code` (Serial, fragments: ALL)

Lower Index Filter: (`informix.l.correction_type = informix.c.code`  
AND `informix.c.category = 'S'` )

type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t3	0	8	830895	00:06.07	0

Rows  
Produced

Rows Read

6 seconds

Many reads to return 0 records!  
Add `notify_email` to the index – or maybe partition?

# Index Filters

Functions applied to columns can result in a filter  
e.g. TRIM

When applying operations to a column consider if reverse operation can be done to the literal value

```
select *  
from billhist  
where billno = "D5678"  
and (billdate + 3) >= "01/31/2023";
```

Filter

Operation applied to every row retrieved

```
select *  
from billhist  
where billno = "D5678"  
and billdate >= ("01/31/2023"::date - 3 units day)
```

No Filter

*Fewer rows need to be read to satisfy the query*

*Operation evaluated ONCE*

# Partitioned Tables

Table is fragmented by INTERVAL:

```
create table fragtest (  
  id serial,  
  received_dt date,  
  store char(6),  
  department char(2))  
  FRAGMENT BY RANGE(received_dt)  
  INTERVAL(INTERVAL(1) DAY(9) TO day)  
  ROLLING (365 fragments) DISCARD  
  STORE IN (datadbs4)  
  PARTITION p1 VALUES < "1/1/2020" IN datadbs4,  
  PARTITION p2 VALUES IS NULL IN datadbs4;  
  
create index fragtest_idx on fragtest(store, department);
```

Query uses **received\_dt** – column used for partitioning:

```
select *  
from fragtest  
where store='009911' and department='00'  
      and received_dt between date('06-01-2021') and date('06-10-2021');
```

# Partitioned Tables

In this case, the index is partitioned with the table

Only the partitions matching the condition will be used  
(fragment elimination)

The partitions/fragments are listed explicitly in the plan

```
1) informix.fragtest: INDEX PATH
```

```
Filters: (informix.fragtest.received_dt <= 06/10/2021 AND  
informix.fragtest.received_dt >= 06/01/2021 )
```

```
(1) Index Name: informix.fragtest_idx
```

```
Index Keys: store department (Serial, fragments: 519,  
520, 521, 522, 523, 524, 525, 526, 527, 528)
```

```
Fragments Scanned: (519) sys_p519 in datadbs4, (520)  
sys_p520 in datadbs4, (521) sys_p521 in datadbs4, (522)  
sys_p522 in datadbs4, (523) sys_p523 in datadbs4, (524)  
sys_p524 in datadbs4, (525) sys_p525 in datadbs4, (526)  
sys_p526 in datadbs4, (527) sys_p527 in datadbs4, (528)  
sys_p528 in datadbs4
```

```
Lower Index Filter: (informix.fragtest.department = '00'  
AND informix.fragtest.store = '009911' )
```

# Key First

Used in a number of situations

- A column in the index is not specified in the query, but a value later in the index is specified (examine the order of columns in the index)

Index is on columns (a, b, c, d)

Query specifies a, b, d...*not* c

# Key First

## Skipped columns in the index

6) informix.p1: INDEX PATH

(1) Index Name: informix.prg\_pk

Index Keys: **log token** **e token** **prog token** **prog seq** (Key-First)  
(Serial, fragments: ALL)

Lower Index Filter: (informix.h.log\_token = informix.p1.log\_token AND  
informix.h.e\_token = informix.p1.e\_token )

Index Key Filters: (informix.p1.prog\_seq = 1 )

Read of p1 has log\_token, e\_token, AND prog\_seq  
**prog\_token is NOT specified**

Results in more rows read than needed

type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t9	<b>8975</b>	1082303360	<b>1026188</b>	00:04.81	1

# Key First

Need columns in an index AFTER an inequality operator

5) informix.oi: INDEX PATH

(1) Index Name: informix.oi\_transfers\_idx2

Index Keys: tran\_id item\_type item\_status login\_id (Key-First)  
(Serial, fragments: ALL)

Lower Index Filter: (informix.oi. tran\_id = informix.an. tran\_id  
AND informix.oi. item\_type = 'C' )

Index Key Filters: (informix.oi.item\_status != 'D' ) AND  
(informix.oi.item\_status != 'H' )

Consider whether inequalities can be changes to “IN” or “=”  
or  
Can the column orders in the index be changed



# Key First

Other reasons for Key-First include:

- Functions applied to indexed columns
- Wrong data type used

# Key Only

Used when the query can be satisfied from the index

- No reads of data pages
- ***Fast!***
- Index must include *all* columns used in filters, joins, select clause, order by...

```
select mytab1.a
from mytab1, mytab2
where mytab1.b = 10
      and mytab1.c = mytab2.c
order by mytab1.d;

create index mytab1_idx on mytab1(b,c,d,a);
```

# Key Only

items\_idx1(order\_num, quantity)

Key-Only **NOT** used

```
select o.*, i.total_price
from orders o, items i
where o.backlog = "n"
      and o.order_num = i.order_num
      and i.quantity > 1
order by i.manu_code
```

Estimated Cost: 5

Estimated # of Rows Returned: 1

Temporary Files Required For: Order By

1) informix.o: SEQUENTIAL SCAN

Filters: informix.o.backlog = 'n'

2) informix.i: INDEX PATH

(1) Index Name: informix.items\_idx1

Index Keys: order\_num quantity (Serial, fragments: ALL)

Lower Index Filter: (informix.o.order\_num = informix.i.order\_num

AND informix.i.quantity > 1 )

NESTED LOOP JOIN

# Key Only

items\_idx1(order\_num, quantity, manu\_code, total\_price)

Key-Only used

```
select o.*, i.total_price
from orders o, items i
where o.backlog = "n"
      and o.order_num = i.order_num
      and i.quantity > 1
order by i.manu_code
```

Estimated Cost: 4

Estimated # of Rows Returned: 1

Temporary Files Required For: Order By

1) informix.o: SEQUENTIAL SCAN

Filters: informix.o.backlog = 'n'

2) informix.i: INDEX PATH

(1) Index Name: informix.items\_idx1

Index Keys: order\_num quantity manu\_code total price

(Key-Only)

(Serial, fragments: ALL)

Lower Index Filter: (informix.o.order\_num = informix.i.order\_num

AND informix.i.quantity > 1 )

NESTED LOOP JOIN

Performance improvement of a Key-Only index read will be compounded when used many times in joins or subqueries

# Key Only

Big improvements possible by adding joining column to the end of an index

```
select mytab1.a
from mytab1, mytab2
where mytab1.a = 10
      and mytab1.b = mytab2.b;

create index mytab1_idx on mytab1(a);
```

An index on mytab1(a) will help retrieval of records

A read of the data page is required to get the value for “b”

```
create index mytab1_idx on mytab1(a, b);
```

An index on mytab1(a,b) will result in a key-only scan and eliminate any need to read the data page

# Autoindex

Optimizer determines that it is most efficient to build a temporary index

```
6) informix.grp: AUTOINDEX PATH  
  
  (1) Index Name: (Auto Index)  
      Index Keys: match_flag  
      Lower Index Filter: informix.h.match_flag = informix.g.match_flag  
NESTED LOOP JOIN
```

This is a potential indicator of a missing index

Will often see this when a view has been executed and results placed in a temporary table

# Join Methods – Nested Loop

Iterate through the table

→ Find matching rows in joined table

1) informix.o: SEQUENTIAL SCAN

2) informix.c: INDEX PATH

(1) Index Name: informix. 100\_1

Index Keys: customer\_num (Serial, fragments: ALL)

Lower Index Filter: informix.c.customer\_num = informix.o.customer\_num

NESTED LOOP JOIN

3) informix.i: INDEX PATH

(1) Index Name: informix. 105\_11

Index Keys: order\_num (Serial, fragments: ALL)

Lower Index Filter: informix.o.order\_num = informix.i.order\_num

NESTED LOOP JOIN

Read a row from table "o"

Read matching row from table "c"

Read matching row from table "i"



# Join Methods – Hash Join

Scan table1 – create hash table

Scan table2 – use hash to join with table1

```
1) informix.c: SEQUENTIAL SCAN
```

```
2) informix.o: SEQUENTIAL SCAN
```

**DYNAMIC HASH JOIN**

Dynamic Hash Filters: `informix.c.customer_num = informix.o.customer_num`

```
3) informix.i: SEQUENTIAL SCAN
```

**DYNAMIC HASH JOIN**

Dynamic Hash Filters: `informix.o.order_num = informix.i.order_num`

Read all rows from table “c” – build hash

Read all rows from table “o” – match on hash key – build new hash

Read all rows from table “i” – match on hash key



# Join Methods – Hash Join

- Hash joins *can* be faster than nested loop joins
  - When returning many rows from joined table
  - Using PDQ
- Will take longer before seeing first rows
- Will use temp space
- OPTCOMPIND will favor one join method over another

```
# OPTCOMPIND      - Controls how the optimizer determines the best
#                  query path. Acceptable values are:
#                  0 Nested loop joins are preferred
#                  1 If isolation level is repeatable read,
#                     works the same as 0, otherwise works same as 2
#                  2 Optimizer decisions are based on cost only
```

# Statistics

Statistics have a huge impact on the query plan chosen

```
select first 1 token from temp_load where profile_token = 103624815
order by token asc
```

1) informix.temp\_load: INDEX PATH

Filters: informix.temp\_load.profile\_token = 103624815

(1) Index Name: informix. 86875379\_205101811  
Index Keys: token (Serial, fragments: ALL)

Chosen index is on **token**  
**profile\_token** is a filter  
No need to sort results

Query statistics:  
-----

The final cost of the plan is reduced because of the FIRST n specification in the query.

Table map :

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

t1                      temp\_load

type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t1	1	550	208364	00:00.80	43

# Statistics

## After updating statistics on **token** & **profile\_token**:

```
select first 1 token from temp_load where profile_token = 103624815
order by token asc
```

### Temporary Files Required For: Order By

1) informix.temp\_load: INDEX PATH

(1) Index Name: informix.temp\_load\_x06

Index Keys: **profile\_token** (Serial, fragments: ALL)

Lower Index Filter: informix.temp\_load.profile\_token = 103624815

Query statistics:

-----

Table map :

-----

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

-----

t1	temp_load
----	-----------

type	table	rows_prod	est_rows	rows_scan	time	est_cost
------	-------	-----------	----------	-----------	------	----------

scan	t1	1	2	1	00:00.00	3
------	----	---	---	---	----------	---

type	rows_sort	est_rows	rows_cons	time	est_cost
------	-----------	----------	-----------	------	----------

sort	1	2	1	00:00.00	0
------	---	---	---	----------	---

Chosen index is on **profile\_token**  
Now need to sort the results

# Views

QUERY: (OPTIMIZATION TIMESTAMP: 11-01-2020 19:04:46)

-----  
**create view** "informix".program\_v (log\_token,id, ...  
<snip>

Estimated Cost: 1744013952

Estimated # of Rows Returned: 45423600

1) prod:informix.prg: SEQUENTIAL SCAN

Filters: prod:informix.program\_seq = 1

<snip>

QUERY: (OPTIMIZATION TIMESTAMP: 11-01-2020 19:04:46)

-----  
**select** d.first\_name, d.last\_name, ...  
<snip>

Estimated Cost: 22521730

Estimated # of Rows Returned: 285

1) informix.h: INDEX PATH

<snip>

6) (Temp Table For View): SEQUENTIAL SCAN

DYNAMIC HASH JOIN

Dynamic Hash Filters: (informix.h.log\_token = (Temp Table For View)  
.log\_token AND informix.h.id = (Temp Table For View).id )

Create View  
SQL & Query  
Plan

SQL &  
Query Plan

Join to  
"Temp  
Table for  
View"

6) (Temp Table For View): AUTOINDEX PATH

(1) Index Name: (Auto Index)

Index Keys: log\_token id

Lower Index Filter: (informix.h. log\_token = (Temp Table For View).log\_token AND  
informix.h.id = (Temp Table For View).id )

NESTED LOOP JOIN

# Views

- View is fully realized (executed)
- Results are placed in temp space
- Temp table of results joined to query
- Can be *very* slow
- May use significant temp space

# Views

- View folding may help
- IFX\_FOLDVIEW in onconfig file

```
# IFX_FOLDVIEW    - Enables (1) or disables (0) folding views that
#                  have multiple tables or a UNION ALL clause.
#                  Disabled by default.
```

```
IFX_FOLDVIEW 1
```

# Views

“create view” in explain plan may show conditions/filters brought in from the query

## Query:

```
select
from mytab, myview
where mytab.id = myview.id
    and mytab.name = myview.name
    and mytab.name = 'Mike'
```

## Explain Plan:

```
create view myview(...)
select ...
from ...
where ...
    and name = 'Mike'
```

**The filters will improve the performance when realizing the view**  
**Use less temp space**  
**Faster!**

# Views

Attempt to reduce the view dataset

May need to simplify joins, or repeat filters, or subqueries

```
select
from mytab, myview
where mytab.id = myview.id
  and mytab.name = myview.name
  and mytab.name = 'Mike'
  and myview.name = 'Mike'
```

```
where mytab.id in (select val from othertab)
  and mytab.id = myview.id
```



```
where mytab.id in (select val from othertab)
  and mytab.id = myview.id
  and myview.id in (select val from othertab)
```

Can you remove UNIQUE in view definition?

Sometimes have no choice but to rewrite the view SQL into the query



# Fun with Dates

```
select count(*)
from logh lh
where DATE (informix.lh.eff_dt ) > "07/06/2022"
      and DATE (informix.lh.eff_dt ) < "09/05/2022";
```

**eff\_dt is a datetime  
Use DATE function**

1) informix.lh: INDEX PATH

Filters: (DATE (informix.lh.eff\_dt ) > 07/06/2022 AND DATE (informix.lh.eff\_dt ) < 09/05/2022 )

(1) Index Name: informix.logh\_x3

Index Keys: eff\_dt status (Key-Only) (Serial, fragments: ALL)

Lower Index Filter: informix.lh.eff\_dt >= EXTEND (07/06/2022 ,year to minute) +  
interval( 1) day to day

Upper Index Filter: informix.lh.eff\_dt < EXTEND (09/05/2022 ,year to minute)

Query statistics:

Table map :

Internal name      Table name

t1                      lh

type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t1	91158	5	91158	00:00.26	3

type	rows_prod	est_rows	rows_cons	time
group	1	1	91158	00:00.28

**Optimizer changed literals to datetimes  
Index is used**

# Index used for Ordering

```
select ...  
from log_header  
where lh_sequence > 534088721 and lh_sequence <= 534089221  
order by lh_sequence;
```

```
create index "informix".log_header_ix on "informix".log_hdr  
  (lh_sequence desc)
```

Order by clause matches index

Estimated Cost: 1  
Estimated # of Rows Returned: 1

1) informix.log\_header: INDEX PATH

(1) Index Name: informix.log\_header\_ix  
Index Keys: lh\_sequence (desc) (Reverse) (Serial, fragments: ALL)  
Lower Index Filter: informix.log\_header.lh\_sequence > 534088721  
Upper Index Filter: informix.log\_header.lh\_sequence <= 534089221

Query statistics:

-----

Table map :

Internal name	Table name
t1	log_header

type	table	rows_prod	est_rows	rows_scan	time	est_cost
scan	t1	242	1	242	00:00.08	1

No need for a temporary table for order by

# Wrap Up

- Explain plans give a detailed insight into how the optimizer chooses to execute SQL
- Most useful tool to determine individual SQL performance
- More situations than have been covered here

# Advanced Informix Consulting and Support

- **Informix Remote DBA 24/7** Peace of mind for your systems
- **Expert consultants** for any Informix problem
- Support for **Informix Upgrades** from any version
- **Migrations** to new hardware, let us help virtualize your systems
- Get help **configuring** and **managing** UNIX systems
- Informix **cloud** migrations
- **IBM Informix sales**
- Let us **tune your system**, we can maximize the potential of your database
- *What can we do for you today?*



The logo for XDB SYSTEMS. The letters 'XDB' are in a large, bold, sans-serif font, with the 'D' and 'B' colored purple and the 'X' in black. A thin horizontal line is positioned below 'XDB'. Below the line, the word 'SYSTEMS' is written in a large, bold, black, sans-serif font.



# Questions?

Send follow-up questions to  
[mike@xdbsystems.com](mailto:mike@xdbsystems.com)



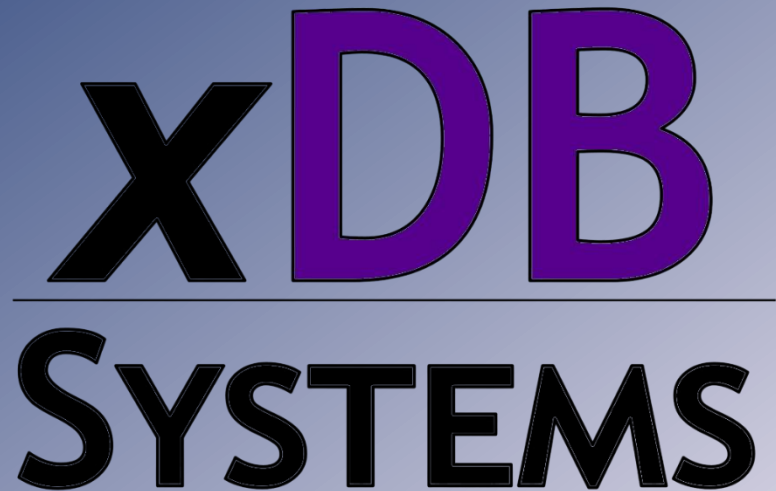
# Thank You

## Mike Walker

mike@xdbsystems.com

For more information:

<https://www.xdbsystems.com>

The logo for XDB SYSTEMS. The letters 'XDB' are in a large, bold, sans-serif font. The 'X' is black, while the 'D' and 'B' are purple with a black outline. A thin horizontal line is positioned below 'XDB'. Below the line, the word 'SYSTEMS' is written in a smaller, bold, black, sans-serif font.

**XDB**

---

**SYSTEMS**