

SQL Tuning, SQL Tips and Tricks

July 22nd, 2008



Florian Boldt, itgain



SQL Tuning, Tips and Tricks

Webinar: actually 153 attendees (that's why I mute your speaker phone, ok?)

- we will have 5 minutes at the end to talk about comments and questions
- please feel free to send questions and comments to florian.boldt@itgain.de
- you can download the powerpoint slides and scripts from <http://www.itgain.de> (see the news section), available on Wednesday.
- this webinar will also be recorded and available soon at <http://www.idug.org>



SQL Tuning, Tips and Tricks

typical questions:

What is the best SQL language construct for the job?

Which indexes are used and which not?

How can I generate alternate SQL access path?

How to put application logic into SQL statements?

Where to start SQL access path tuning?

What are my most important SQL statements?



SQL Tuning, Tips and Tricks

Goal:

provide means to collect and analyze the SQL Workload of your database

tune the workload with the most efficient tuning object:
the index

apply „new“ SQL constructs to simplify and speedup SQL statements



SQL Tuning, Tips and Tricks

Agenda

How to collect the SQL workload for your database

The DB2 Design Advisor: Index recommendations and virtual indexes

Index analysis: DB2PD (which indexes are used, which not)

SQL Tips and Tricks



SQL Tuning, Tips and Tricks

Agenda

How to collect the SQL workload for your database

The DB2 Design Advisor: Index recommendations and virtual indexes

Index analysis: DB2PD (which indexes are used, which not)

SQL Tips and Tricks



What are my most important SQL statements?

Those which hurt...

How to find out?

- phone calls from frustrated users
- bad performance metrics
- Reports sorted by elapsed time or number of executions



Collect your SQL workload

create a table to store SQL statements and performance counters (Version 0.5)

```
CREATE TABLE DB2MONITOR.DYNSQL_COLL
(
  NUM_EXECUTIONS      BIGINT,
  TOTAL_EXEC_TIME     BIGINT,
  STMT_TEXT            VARCHAR(32000)
)
IN TS_32K;
```



Collect your SQL workload

DB2 built in Snapshot Table Functions / Views

```
INSERT INTO DB2MONITOR.DYNSQL_COLL
SELECT  NUM_EXECUTIONS ,
        TOTAL_EXEC_TIME,
        STMT_TEXT
FROM    TABLE(sysproc.snapshot_dyn_sql('SAMPLE',-1))
AS temp
WHERE   STMT_TEXT LIKE '%select%'
        OR STMT_TEXT LIKE '%SELECT%'
...
```



Collect your SQL workload

Problem: same statements stored multiple times

-> hard to analyse (imagine “group by” on statement text)

Solution: if the sql stmt does not exist in the table, store it
if it does exist, update the performance counters

-> **MERGE** is perfect for the job

```
MERGE INTO <target> USING <source> ON <match-condition>  
{WHEN [NOT] MATCHED [AND <predicate>]  
  THEN [UPDATE SET ... | DELETE | INSERT VALUES .... | SIGNAL ...] }  
[ELSE IGNORE]
```

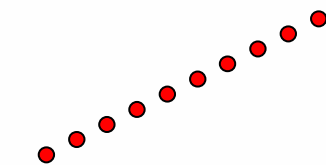


Collect your SQL workload using MERGE

still a little Problem: performance counters are cumulative,
 a MERGE (update branch) would overwrite existing counters.
 other SQL is kicking our SELECT out of the dyn sql cache

`SELECT * FROM employee`

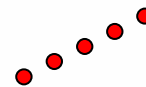
num_execs: 10



day 1

`SELECT * FROM employee`

num_execs: 5



day 2

MERGE inserts: 10

MERGE updates: 5

Summary: num_execs 5

wrong result, should be num_execs 15 (10 + 5)

Solution: we need to store the last snapshot and calculate the delta



Collect your SQL Workload

create a table to store SQL statements and performance counters (Version 1.0)

```
CREATE TABLE DB2MONITOR.DYNSQL_COLL  
(NUM_EXECUTIONS          BIGINT,  
TOTAL_EXEC_TIME         BIGINT,  
LAST_NUM_EXECUTIONS     BIGINT,  
LAST_TOTAL_EXEC_TIME    BIGINT,  
STMT_TEXT                VARCHAR(32000)  
)  
IN TS_32K;
```



script
available



Collect your SQL workload

Combine MERGE and Snapshot function / view



script
available

```
MERGE INTO DB2MONITOR.DYNSQL_COLL mdyn USING
(SELECT NUM_EXECUTIONS,
        CAST(STMT_TEXT AS VARCHAR(32000)) AS STMT_TEXT,
        total_exec_time
FROM TABLE(sysproc.snapshot_dyn_sql('SAMPLE',-1)) AS temp
WHERE STMT_TEXT LIKE '%select%'
        OR STMT_TEXT LIKE '%SELECT%'
) sdyn ON (mdyn.stmt_text = sdyn.stmt_text)
```



Collect your SQL workload

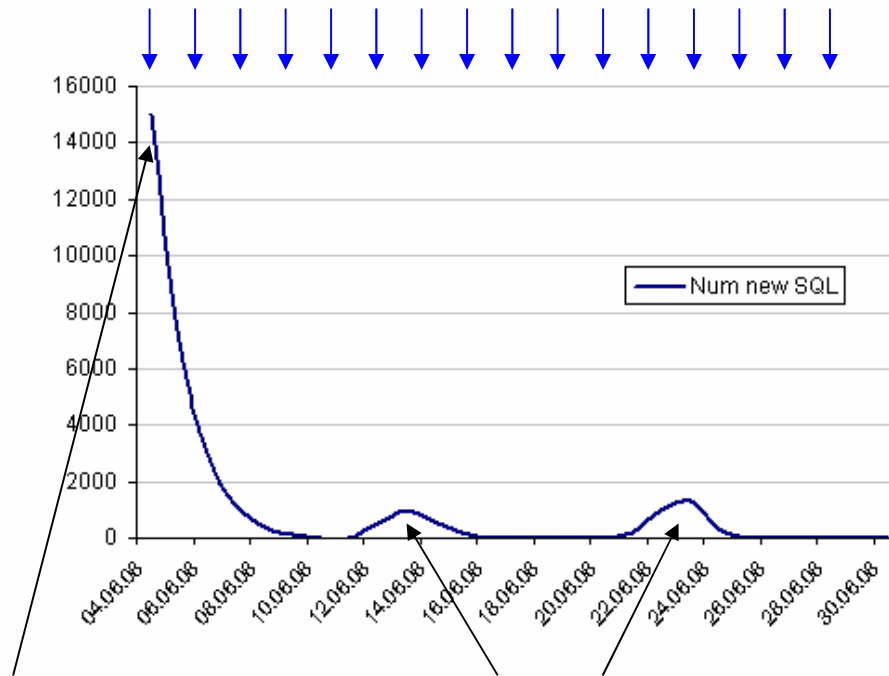
Combine MERGE and Snapshot function / view

```
WHEN MATCHED AND tdyn.last_num_executions < sdyn.num_executions THEN  
UPDATE SET  
num_executions      = num_executions + sdyn.num_executions  
                    - tdyn.last_num_executions,  
total_exec_time     = total_exec_time + sdyn.total_exec_time  
                    - tdyn.last_total_exec_time,  
last_num_executions = sdyn.num_executions,  
last_total_exec_time = sdyn.total_exec_time  
WHEN MATCHED AND tdyn.last_num_executions > sdyn.num_executions THEN  
UPDATE SET  
num_executions      = num_executions + sdyn.num_executions,  
total_exec_time     = total_exec_time + sdyn.total_exec_time,  
last_num_executions = sdyn.num_executions,  
last_total_exec_time = sdyn.total_exec_time  
WHEN NOT MATCHED THEN  
INSERT (stmt_text, num_executions, last_num_executions, total_exec_time,  
         last_total_exec_time)  
VALUES (sdyn.stmt_text, sdyn.num_executions, sdyn.num_executions,  
         sdyn.total_exec_time, sdyn.total_exec_time)
```



Collect your SQL Workload using MERGE + SNAPSHOT

Run MERGE statement in a loop (every 1 hour, or more often)



Begin of Collection

new SQL coming in

TUNING.DYNSQL

Stmt_text	Num_execs
Select * fro..	10.500
Select a,b ...	8.235
Select c fr...	4.856
....

here is your workload, what's next?



Remarks on this approach:

- much better than a single DYNSQL snapshot
- you may miss some statements but it is still showing the trend
- statement text is limited to 32 k (now 2 MB possible, Clob might help, but harder to handle)
- time consuming to compare statement text, **one could add a hashing algorithm for sql stmt text** to speedup search performance for given sql
 - >capture workloads for a given time (nightly batch, etl, user queries) and delete the sql table more frequently
- same statements with different literals are considered different

```
SELECT * FROM employee WHERE empno = 'A1000'
```

```
SELECT * FROM employee WHERE empno = 'B2000'
```

- there are tools which deal with those problems! (that was the shameless marketing moment)



use your workload and find appropriate indexes (and/or MQT/MDC)

TUNING.DYNSQL

Stmt_text	Num_execs
Select * fro..	10.500
Select a,b ...	8.235
Select c fr...	4.856
....

```

SELECT
'--#SET FREQUENCY ' || char(num_executions) ||
chr(13) || chr(10) || ltrim(rtrim(stmt_text ))
|| ';'
FROM "DB2MONITOR"."MYDYNSQL"
ORDER BY num_executions desc
fetch first 20 rows only;
    
```



transfer into file

```

--#SET FREQUENCY 10500
Select * from EMPLOYEE
WHERE...
--#SET FREQUENCY 8235
select a,b ...
--#SET FREQUENCY 4856
select c fr....
    
```

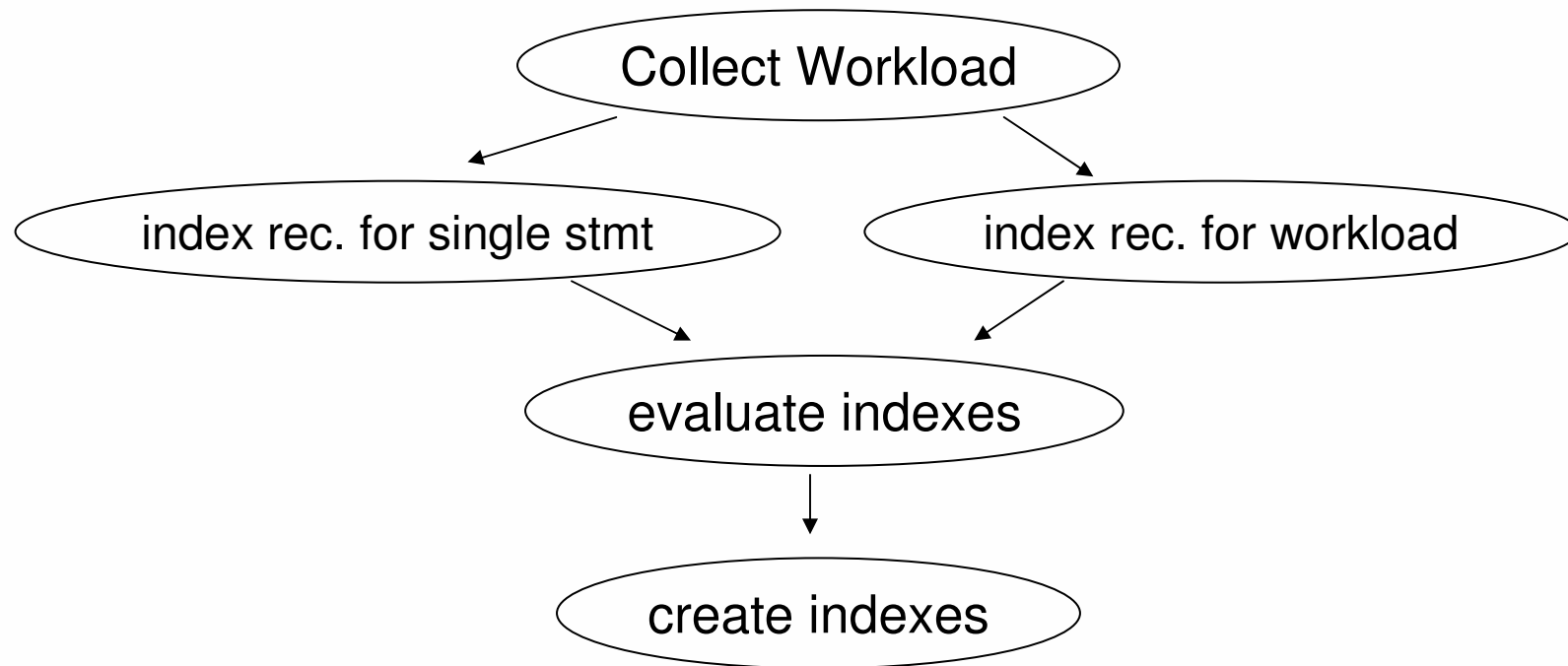
use file as input for DB2 Design Advisor

DB2ADVIS

receive index recommendations



Approach



SQL Tuning, Tips and Tricks

Agenda

How to collect the SQL workload for your database

The DB2 Design Advisor: Index recommendations and virtual indexes

Index analysis: DB2PD (which indexes are used, which not)

SQL Tips and Tricks



DB2 Design Advisor (DB2ADVIS)

Example:

```
db2adviz -db sample -stmt "select e.lastname, d.deptname  
from employee1 e, department1 d where  
e.workdept=d.deptno"
```

-- LIST OF RECOMMENDED INDEXES

```
CREATE INDEX "DB2ADMIN"."IDX807172138370000" ON "DB2ADMIN"."DEPARTMENT1"  
("DEPTNO" ASC, "DEPTNAME" ASC) ALLOW REVERSE SCANS ;  
CREATE INDEX "DB2ADMIN"."IDX807172138430000" ON "DB2ADMIN"."EMPLOYEE1"  
("WORKDEPT" ASC, "LASTNAME" ASC) ALLOW REVERSE SCANS ;
```

Optimization finished.

2 indexes in current solution

[1127,0000] timerons (without recommendations)

[960,0000] timerons (with current solution)

[14,82%] improvement with 1 call and no thinking ☺



script
available



alternative **Solution** to DB2ADVIS

set register from command line:

```
SET CURRENT EXPLAIN MODE RECOMMENT INDEX
```

run you SQL statement:

```
SELECT e.lastname, d.deptname  
FROM employeel e, department1 d  
WHERE e.workdept=d.deptno
```

index recommendations in
Table ADVIS.INDEX
(one of your Explain tables)
.../sqllib/misc/EXPLAIN.DDL



**script
available**

Visualize Access Path with DB2EXFMT

```
db2exfmt -1 -d sample -g TIC -o output.txt -w -1
```



„ok, i got those index recommendations, but are they good?“

-> virtual indexes can show the new access path without physical index creation

SOLUTION:

```
SET CURRENT EXPLAIN MODE EVALUATE INDEX
```

Visualize Access Path with DB2EXFMT

```
db2exfmt -1 -d sample -g TIC -o output.txt -w -1
```



SAMPLE SCRIPT for index recommendations and virtual indexes

```
title find and emulate indexes, itgain GmbH 2008
db2 connect to %1 user %2 using %3 > NULL
echo create „old“ access path
db2 set current explain mode EXPLAIN > NULL
db2 -tsf %4 > NULL
db2exfmt -1 -d %1 -e %2 -g TIC -o before.txt -u %2 %3 -w -1
echo create new access path with virtual index
db2 set current explain mode recommend indexes
db2 -tsf %4 > NULL
db2exfmt -1 -d %1 -e %2 -g TIC -o after.txt -u %2 %3 -w -1
db2 set current explain mode no > NULL
db2 connect reset > NULL
echo writing virtual Indexes to file virind_out.txt
db2 connect to %1 user %2 using %3 > NULL
db2 select name, creator, tname, colnames from %2.ADVISE_INDEX where EXPLAIN_TIME=(SELECT
    MAX(EXPLAIN_TIME) from %2.ADVISE_INDEX) > virind_out.txt
echo finished, see files before.txt and after.txt
notepad before.txt
notepad after.txt
```



this simple script delivered an index for sql on a database migrated from z/os to AIX. That index reduced runtime from 10 hours to 123 seconds.



SQL Tuning, Tips and Tricks

Agenda

How to collect the SQL workload for your database

The DB2 Design Advisor: Index recommendations and virtual indexes

Index analysis: DB2PD (which indexes are used, which not)

SQL Tips and Tricks



Index recommendations, index tuning, virtual indexes....

... and which indexes are used and which not?

the only tool that delivers the answer: DB2PD, the DB2 problem determination tool.

You have DB2PD (when you are at least on DB2 LUW Version 8 Fixpak 9) 😊



usage of DB2PD

- Serverside command line tool
- no API
- „unformatted“ output

call it from command line:

```
db2pd -db sample -tcbstats index
```

same thing for unused tables

look for section TCB Index Stats:

TableName	IID	Scans
SYSTABLES	1	6413
SYSCOLUMNS	1	631
EMPLOYEE	2	25
....

number of index scans

these are the index scans since the last database activation!

index ID per Table



DB2PD

- unfortunately no SQL table function or stored procedure for DB2PD
- you need to grep the output on the db2 server

**we saw for an SAP customer (on DB2 AIX) 48,500 indexes
after 3 weeks of monitoring with DB2PD just 4,500 indexes were used.
So that database has 44,000 unused indexes.**



SQL Tips and Tricks

your SQL is a beauty? Make it look like a beauty..

<http://www.sqlinform.de> (online and desktop version available)



SQL Tips and Tricks

- generating Testdata using recursive SQL (simple)



generating Testdata using recursive SQL (simple)

check whether a date columns has not any gaps for the year 2007

problem:

Table `Daily_SUM`, Columns `TransDate`

...

25.08.2007

26.08.2007

_____ gap

28.08.2008

...

solutions:

generate a temporary table with all dates for the year 2007



generate a temporary table with all dates for the year 2007

- subtract (except) DAILY_SUM from temp

```
WITH ALL_DAYS(DT) AS
( VALUES (DATE('2007-01-01'))
  UNION ALL
  SELECT DT + 1 DAY FROM ALL_DAYS
  WHERE DT < '2007-12-31'
)
SELECT DT FROM ALL_DAYS
EXCEPT
SELECT TRANSDATE FROM DAILY_SUM;
```



script
available

notice:
no physical table here



SQL Tips and Tricks

- generating Testdata using values clause (simple)



generating Testdata using the values clause (simple)

```
VALUES (1, 2), (3, 4), (5, 6);
INSERT INTO TEST VALUES (1, 2), (3, 4), (5, 6);
SELECT * FROM TABLE (VALUES (1, 2), (3, 4), (5, 6));
SELECT * FROM TABLE (VALUES (1, 2), (3, 4), (5, 6)),
                     TABLE (VALUES (1, 2), (3, 4), (5, 6))
ORDER BY 1, 2, 3, 4;
```



1	2	1	2
1	2	3	4
1	2	5	6
3	4	1	2
3	4	3	4
3	4	5	6
5	6	1	2
5	6	3	4
5	6	5	6



SQL Tips and Tricks

- **OLAP functions**
 - delete duplicates rows
 - avoid self-joins



- OLAP functions
 - **problem**: delete duplicates (or even n-plicate rows)

Testdata:

```
CREATE TABLE employee_clone LIKE db2admin.employee;  
repeat 3 times (or so):  
INSERT INTO employee_clone SELECT * FROM db2admin.employee;
```

what makes a duplicate row a duplicate? In this case **EMPNO** would be good. However to make the idea clear we use **FIRSTNME**, **LASTNAME** and **BIRTHDATE**



OLAP functions

- **solution**: delete duplicates (or even n-plicate rows)

get rid of duplicates:

```
DELETE FROM  
(SELECT row_number() OVER(PARTITION BY FIRSTNME,  
LASTNAME, BIRTHDATE ORDER BY BIRTHDATE DESC) AS rn  
FROM employee_clone)  
WHERE rn > 1;
```



script
available

notice:

ORDER BY Clause
can determine which
one to delete (here it
is obsolete)

FIRSTNME, LASTNAME,
BIRTHDATE is the
determination for
uniqueness here



SQL Tips and Tricks

- OLAP functions
 - avoid self-joins

Problem: calculate delta between rows

Predecessor row unknown, self join needed?

Sample:

```
INSERT INTO DB2MONITOR.DB_ROWS (  
SELECT snapshot_timestamp, rows_read, rows_selected  
FROM TABLE(sysproc.snapshot_database('SAMPLE', -1))  
as temp)
```

Snapshot_time	rows_read	rows_selected
2008-07-20 18:00:37.346862	200237	17001
2008-07-20 18:07:42.129935	200283	17046

delta reads, delta selected???



OLAP functions

- avoid self-joins



Solution:

```

SELECT snapshot_time,
rows_read,
MAX(rows_read) OVER(ORDER BY snapshot_time ROWS BETWEEN 1
PRECEDING AND 1 PRECEDING) pred_read,
rows_selected,
MAX(rows_selected) OVER(ORDER BY snapshot_time ROWS BETWEEN 1
PRECEDING AND 1 PRECEDING) pred_selected
FROM tuning.db_rows
    
```

Snapshot_time	rows_read	pred_read	rows_selected	pred_selected
2008-07-20 18:19:30.068233	200304	[Null]	17048	[Null]
2008-07-20 18:19:48.022780	200388	200304	17132	17048
2008-07-20 18:24:56.554384	200441	200388	17186	17132

now we can calculate the delta



OLAP functions

- avoid self-joins

Solution:



```

SELECT
snapshot_time,
rows_read,
rows_selected,
rows_read - MAX(rows_read) OVER(ORDER BY snapshot_time ROWS
BETWEEN 1 PRECEDING AND 1 PRECEDING) delta_read,
rows_selected - MAX(rows_selected) OVER(ORDER BY snapshot_time ROWS
BETWEEN 1 PRECEDING AND 1 PRECEDING) delta_selected
FROM tuning.db_rows
    
```

Snapshot_time	rows_read	rows_selected	delta_read	delta_selected
2008-07-20 18:19:30.068233	200304	17048	[Null]	[Null]
2008-07-20 18:19:48.022780	200388	17132	84	84
2008-07-20 18:24:56.554384	200441	17186	53	54

delta: 137 = sum: 137



SQL Tips and Tricks

- truncate table (l a) and b))

solution a)

```
ALTER TABLE t1 ACTIVATE NOT LOGGED INITIALLY  
WITH EMPTY TABLE;
```

solution b)

```
IMPORT FROM /dev/null OF DEL REPLACE INTO t1;  
  
(name of nul device on Windows is NUL instead  
of /dev/nul)
```



SQL Tips and Tricks

- truncate (part) table **solution II**
 - committed delete per SQL Stored Procedure (Serge Rielau, IBM Toronto „SQL on Fire“)

```
CREATE PROCEDURE purgeInventory(IN dt DATE)
BEGIN
  DECLARE SQLCODE INTEGER;
loop: LOOP
  DELETE FROM
    (SELECT 1 FROM Inventory
     WHERE InvDate <= dt
      FETCH FIRST 1000 ROWS ONLY) AS D;
  IF SQLCODE = 100 THEN
    LEAVE loop;
  END IF;
  COMMIT;
END LOOP loop;
END

CALL purgeInventory('2003-10-01')
```



SQL Tips and Tricks

Last day of month (given date or timestamp)

Solution:

```
CREATE FUNCTION LAST_DAY (D Date)
RETURNS Date
LANGUAGE SQL
SPECIFIC LAST_DAYDate
DETERMINISTIC
CONTAINS SQL
NO EXTERNAL ACTION
RETURN D + 1 month - day(D + 1 month) day ;
```



SQL Tips and Tricks

- Access Path Optimization

- in detail far too complex for this webinar
- but here are some tips on how to deal with it
 - the index is key for SQL performance
 - „make“ a „good“ join order
 - sometimes the optimizer is working with assumptions which do not reflect reality
 - rewrite rules often lead to different/better access paths
 - how many indexes per table (myth: 4 – 6, reality: as many as your environment needs and allows, could be even 10 or more)



Access Path Optimization

sometimes the optimizer is wrong on it's assumptions:

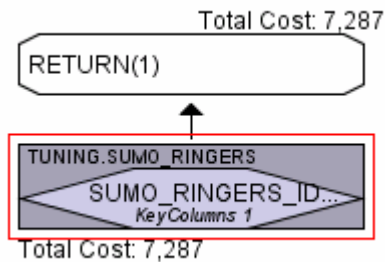
1. (sorry IBM people that I put you in a sumo wrestler table)
2. (sorry Eileen that you are the only women in that table (you were the last women in the employee table, that's why it is you))
3. how would the access path look like for:

FIRSTNME	LASTNAME	SALARY	SEX
MICHAEL	THOMPSON	94250	M
JOHN	GEYER	80175	M
IRVING	STERN	72250	M
THEODORE	SPENSER	86150	M
VINCENZO	LUCCHESI	66500	M
SEAN	O'CONNELL	49250	M
BRUCE	ADAMSON	55280	M
MASATOSHI	YOSHIMURA	44680	M
JAMES	WALKER	50450	M
DAVID	BROWN	57740	M
WILLIAM	JONES	68270	M
JAMES	JEFFERSON	42180	M
SALVATORE	MARINO	48760	M
DANIEL	SMITH	49180	M
JOHN	PARKER	35340	M
PHILIP	SMITH	37750	M
RAMLAL	MEHTA	39950	M
WING	LEE	45370	M
JASON	GOUNOT	43840	M
GREG	ORLANDO	39250	M
KIYOSHI	YAMAMOTO	64680	M
ROBERT	MONTEVERDE	37760	M
EILEEN	SCHWARTZ	46250	F
ROY	ALONZO	31840	M

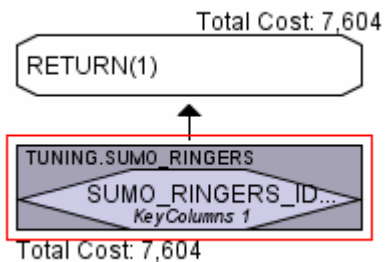


```
SELECT SALARY
FROM tuning.sumo_wrestlers
WHERE SEX='M'
```

1. index on SEX, no statistics



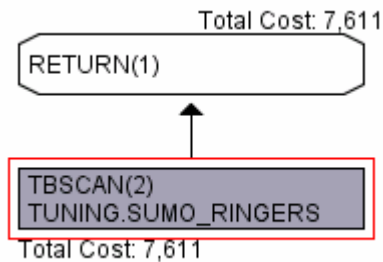
2. index on SEX with simple statistics



FIRSTNME	LASTNAME	SALARY	SEX
MICHAEL	THOMPSON	94250	M
JOHN	GEYER	80175	M
IRVING	STERN	72250	M
THEODORE	SPENSER	86150	M
VINCENZO	LUCCHESI	66500	M
SEAN	O'CONNELL	49250	M
BRUCE	ADAMSON	55280	M
MASATOSHI	YOSHIMURA	44680	M
JAMES	WALKER	50450	M
DAVID	BROWN	57740	M
WILLIAM	JONES	68270	M
JAMES	JEFFERSON	42180	M
SALVATORE	MARINO	48760	M
DANIEL	SMITH	49180	M
JOHN	PARKER	35340	M
PHILIP	SMITH	37750	M
RAMLAL	MEHTA	39950	M
WING	LEE	45370	M
JASON	GOUNOT	43840	M
GREG	ORLANDO	39250	M
KIYOSHI	YAMAMOTO	64680	M
ROBERT	MONTEVERDE	37760	M
EILEEN	SCHWARTZ	46250	F
ROY	ALONZO	31840	M

```
SELECT SALARY
FROM tuning.sumo_wrestlers
WHERE SEX='M'
```

3. index on SEX, distribution statistics on key columns



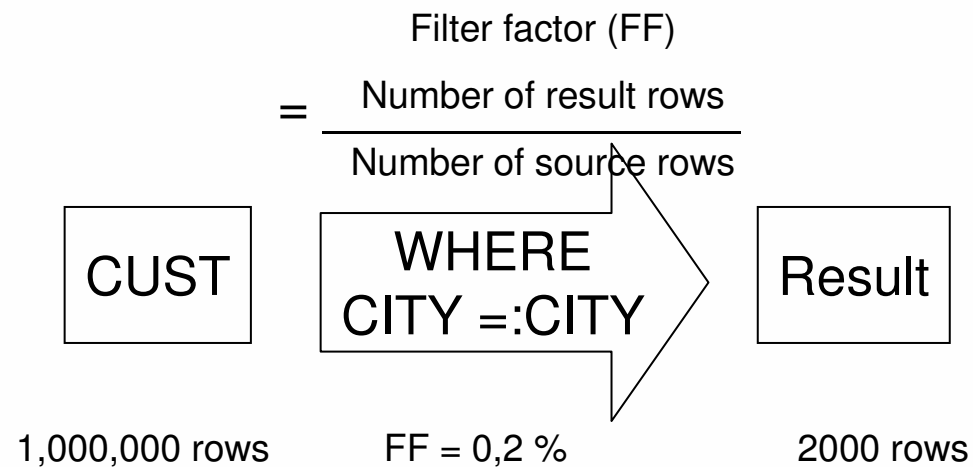
where SEX='M'
touches 23 of 24 rows
of the table which is
95,8%. A Tablescan
is faster here.
Distribution statistics
generated the correct
access path here.

FIRSTNME	LASTNAME	SALARY	SEX
MICHAEL	THOMPSON	94250	M
JOHN	GEYER	80175	M
IRVING	STERN	72250	M
THEODORE	SPENSER	86150	M
VINCENZO	LUCCHESI	66500	M
SEAN	O'CONNELL	49250	M
BRUCE	ADAMSON	55280	M
MASATOSHI	YOSHIMURA	44680	M
JAMES	WALKER	50450	M
DAVID	BROWN	57740	M
WILLIAM	JONES	68270	M
JAMES	JEFFERSON	42180	M
SALVATORE	MARINO	48760	M
DANIEL	SMITH	49180	M
JOHN	PARKER	35340	M
PHILIP	SMITH	37750	M
RAMLAL	MEHTA	39950	M
WING	LEE	45370	M
JASON	GOUNOT	43840	M
GREG	ORLANDO	39250	M
KIYOSHI	YAMAMOTO	64680	M
ROBERT	MONTEVERDE	37760	M
EILEEN	SCHWARTZ	46250	F
ROY	ALONZO	31840	M



Access Path Optimization

- the index is key for SQL performance
- know the filter factor



Filter Factor is a property
of a predicate

(relational database index design and the optimizers, wiley, 2005)



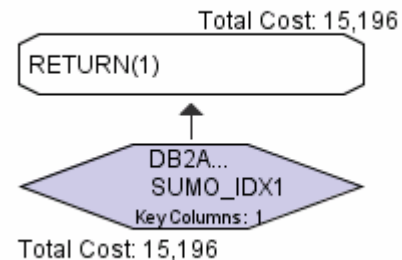
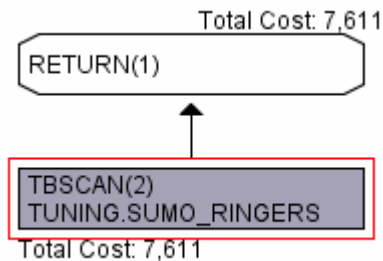
```
SELECT SALARY
FROM tuning.sumo_wrestlers
WHERE SEX=?
```

4. what about parameter markers (no distribution statistics applicable)

```
DB2SET DB2_SELECTIVITY=YES
DB2STOP
DB2START
```

```
SELECT SALARY
FROM db2monitor.sumo_wrestlers
WHERE sex=? SELECTIVITY 0.958
```

```
SELECT SALARY
FROM db2monitor.sumo_wrestlers
WHERE sex=? SELECTIVITY 0.001
```



Access Path Optimization

- a good index (**SIMPLE SELECT STATEMENT**)
 - find equal predicates (WHERE COL =...) and make these the first columns of the index
 - > cut index slices, reduce data reads
 - add ORDER BY columns
 - > avoid sorts
 - add remaining columns from select list (if possible)
 - > achieve index only access



Access Path Optimization

- a good index (**SIMPLE SELECT**)

```
SELECT
  lastname,
  firstnme,
  salary
FROM
  tuning.sumo_ringers
WHERE sex = 'M'
       AND salary > 10000
ORDER BY lastname
;
```

I equal predicate: sex

II range predicate: salary

III sort order: lastname

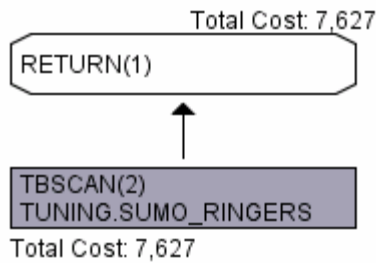
IV remaining columns:
firstnme

CREATE INDEX sumo_idx1 **ON** tuning.sumo_ringers (sex, salary, lastname, firstnme)

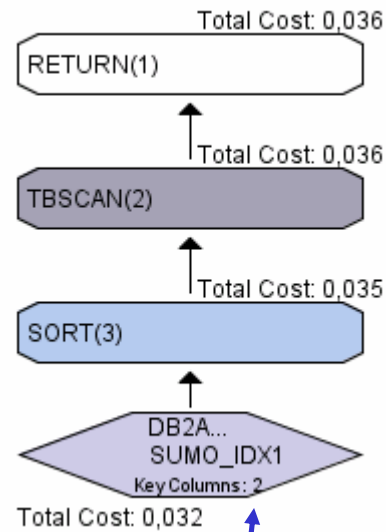
take [include columns] into consideration!!!



before:



after:



ORDER BY lastname,
can't avoid a sort here
but still good for index only access

key_columns: 2

first 2 columns of index were used

(sex, salary) !sex has a bad selectivity when evenly distributed!!



Access Path Optimization

- rewrite Rules (just one sample, not part of this presentation)

NOT IN to EXCEPT

```
SELECT lastname  
FROM employee WHERE workdept  
NOT IN (SELECT deptno FROM  
department)
```

```
SELECT lastname FROM employee  
EXCEPT  
SELECT lastname  
FROM employee WHERE workdept  
IN (SELECT deptno FROM  
department)
```

there are many more
rewrite rules, check

[Gulutzan, Pelzer. *SQL
Performance Tuning*, Addison
Wesley, 2003]



Summary - SQL Tuning, Tips and Tricks

What is the best SQL language construct for the job?

MERGE/OLAP functions/Rekursion/SQL PL/table functions

Which indexes are used and which not?

use DB2PD

How can I generate alternate SQL access path?

Selectivity/Runstats/Optimizer Level/rewrite rules

How to put application logic into SQL statements?

MERGE/SQL PL/..

Where to start SQL access path tuning?

most expensive SQL / start with index tuning and DB2ADVIS

What are my most important SQL statements?

use snapshot table functions / views and rank your workload



Speedgain for DB2 LUW Webinar:

Links

[Using a SELECTIVITY clause to influence the optimizer](#)

Books

[SQL Performance Tuning](#)

[Relational Database Index Design and the Optimizers](#)

[SQL Tuning](#)

This presentation

<http://www.itgain.de>



Questions



Further information

Florian Boldt, Product Manager Speedgain for DB2

florian.boldt@itgain.de

<http://www.itain.de> -> news -> IDUG webinar download

