Information Management

IBM

Data Management

DB2 for Linux, UNIX and Windows Query Access Plan Stability

John Hornibrook
IBM Canada

Best practices for writing good SQL

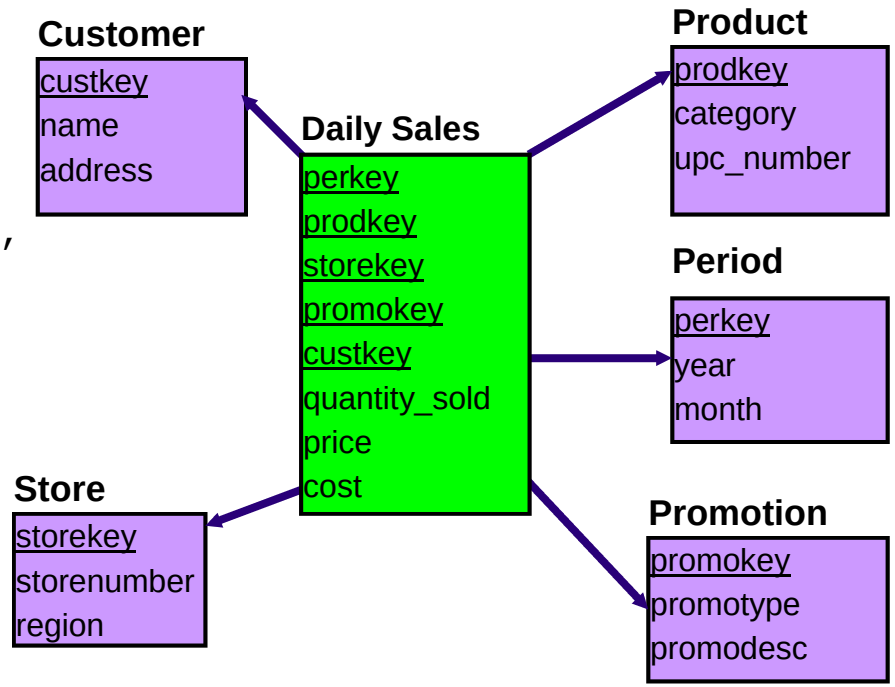
- Avoid complex expressions in search conditions
  - Avoid join predicates on expressions
  - Avoid expressions over columns in local predicates
  - Avoid data type mismatches on join columns
  - Avoid non-equality join predicates
- Avoid unnecessary outer joins
- Use OPTIMIZE FOR N ROWS clause with FETCH FIRST N ROWS ONLY clause
- If you are using the star schema join, ensure your queries fit the required criteria
- Avoid redundant predicates
- Refer to:
  - The DB2Night Show #52: Writing Optimized DB2 LUW SQL Queries
    - http://www.dbisoftware.com/blog/db2nightshow.php?id=268
  - Best Practices: Writing and Tuning Queries for Optimal Performance
    - http://www.ibm.com/developerworks/data/bestpractices/querytuning/

Proper system configuration

- Additional considerations:
  - Consider using **constraints** to improve query optimization
    - Allows more semantic query rewrites
  - Choose the best **optimization class** for your workload
    - Default is 5
    - 0,1,2,3,5,7 and 9 available
  - Certain DB2 registry variables can provide improved optimization
    - See http://publib.boulder.ibm.com/infocenter/db2luw/v9r7/topic/com.ibm.db2.luw.admin.regvars.doc/doc/r0005664.html

So how do we fix query B?

```
SELECT CATEGORY_DESC, SUM(PERCENT_DISCOUNT),
  SUM(EXTENDED_PRICE),
SUM(SHELF_COST_PCT_OF_SALE)
FROM PERIOD, DAILY_SALES, PRODUCT, STORE,
  PROMOTION
WHERE PERIOD.PERKEY=DAILY_SALES.PERKEY AND
PRODUCT.PRODKEY=DAILY_SALES.PRODKEY AND
STORE.STOREKEY=DAILY_SALES.STOREKEY AND
PROMOTION.PROMOKEY=DAILY_SALES.PROMOKEY AND
CALENDAR_DATE BETWEEN '04/01/2004' AND
  '04/14/2004' AND
STORE_NUMBER='01' AND
PROMODESC = 'Web' AND
PACKAGE_SIZE = '16 OZ' AND
SUB_CATEGORY = 747
GROUP BY CATEGORY_DESC ;
```



Query B

- Notice the query is over a star schema
- There is often:
  - Skew in the fact table foreign keys
  - Many more primary keys than foreign keys
- **Statistical views** will allow the optimizer to see these characteristics

Fixing query B1

- A simple approach for statistical views in a star schema*:
  - Create a statistical view for each dimension-fact join
  - Limit to dimensions with:
    - Skew in the fact table foreign key columns
    - Many more dimension ids than exist in fact table
- Correct for data correlation between columns

```
PACKAGE_SIZE = '16 OZ' AND
SUB_CATEGORY = 747
```

  - PACKAGE_SIZE has 3680 distinct values
  - SUB_CATETORY has 160 distinct values
  - There are 5000 distinct combinations
  - The optimizer thinks there are 3680*160 = 588800 !
  - **Collect column group statistics on the PRODUCT table**

*Statistical views are very general and can be used for many types of schemas and queries

Fixing query B1 with statistical views

Create a statistical view for:

- **(store - daily_sales)**
- **(product - daily_sales)**
- **(period - daily_sales)**

```
CREATE VIEW DB2DBA.SV_STORE AS
 (SELECT S.*
 FROM STORE S, DAILY_SALES F
 WHERE S.STOREKEY = F.STOREKEY)

CREATE VIEW DB2DBA.SV_PRODUCT AS
 (SELECT P.*
 FROM PRODUCT P, DAILY_SALES F
 WHERE S.PRODKEY = F.PRODKEY)

CREATE VIEW DB2DBA.SV_PERIOD AS
 (SELECT P.*
 FROM PERIOD P, DAILY_SALES F
 WHERE S.PERKEY = F.PERKEY)
```

Include all dimension columns
Don't need to include fact table columns

Fixing query B1 with statistical views

Enable statistical views for query optimization

```
ALTER VIEW DB2DBA.SV_STORE ENABLE QUERY OPTIMIZATION
ALTER VIEW DB2DBA.SV_PRODUCT ENABLE QUERY OPTIMIZATION
ALTER VIEW DB2DBA.SV_PERIOD ENABLE QUERY OPTIMIZATION
```

Gather statistics for the statistical views:

```
RUNSTATS ON TABLE DB2DBA.SV_STORE WITH DISTRIBUTION
RUNSTATS ON TABLE DB2DBA.SV_PRODUCT WITH DISTRIBUTION
RUNSTATS ON TABLE DB2DBA.SV_PERIOD WITH DISTRIBUTION
```

Fixing query B1 with a column group statistic

- Note the nested parentheses

```
RUNSTATS ON TABLE DB2INST1.PRODUCT ON ALL COLUMNS
AND COLUMNS ((PACKAGE_SIZE, SUB_CATEGORY))
WITH DISTRIBUTION
AND SAMPLED DETAILED INDEXES ALL
```

- **Now query B and B_1 run in 10s !!**
  - Previously B was 15s and B_1 was 500s

Access plan stability

- Ability to alter a package to specify:
  - Access plan reuse option (APREUSE)

```
ALTER PACKAGE DB2USER.EMPADMIN APREUSE
REBIND DB2USER.EMPADMIN
```

  - Optimization profile

```
ALTER PACKAGE DB2USER.EMPADMIN
  OPTIMIZATION PROFILE DB2USER.JOINHINT
```

    - Immediately affects subsequent dynamic SQL for that package
    - Affects static SQL on next REBIND

Optimization profiles

- **Mechanism to control statement optimization**
  - Can control both query rewrite optimization and access path optimization
- **Sets of explicit optimization guidelines (DML statements)**
  - “For app1.0, only consider routing to MQTs: Newt.AvgSales and Newt.SumSales”
  - “Use index ISUPPKEY to access SUPPLIERS in the subquery of query 9”
- **Can be put into effect without editing application code**
  - Compose optimization profile, add to DB, rebind targeted packages
- **Should only be used after all other tuning options exhausted**
- **Available since DB2 9**

Optimization profiles: anatomy

- **XML document**
  - Elements and attributes understood as explicit optimization guidelines
  - Composed and validated with Current Optimization Profile Schema (COPS)
    - sqllib/misc/DB2OptProfile.xsd
  - **Profile Header** (exactly one)
    - Meta data and processing directives
  - **Global optimization guidelines** (at most one)
    - Applies to all statements for which profile is in effect
    - E.g. eligible MQTs guideline defining MQTs to be considered for routing
  - **Statement-level optimization guidelines** (zero or more)
    - Applies to a specific statement for which profile is in effect
    - Specifies aspects of desired execution plan

Sample optimization profile

```
<?xml version="1.0" encoding="UTF-8"?>
<OPTPROFILE VERSION="9.7.0">
<!--
    Global optimization guidelines section.
    Optional but at most one.
-->
<OPTGUIDELINES>
    <MQT NAME="DBA.AvgSales"/>
    <MQT NAME="DBA.SumSales"/>
</OPTGUIDELINES>
<!--
    Statement profile section.
    Zero or more.
-->
<STMTPROFILE ID="Guidelines for TPCD Q9">
    <STMTKEY SCHEMA="TPCD">
        <![CDATA[SELECT S.S_NAME, S.S_ADDRESS, S.S_PHONE,
S.S_COMMENT FROM PARTS P, SUPPLIERS S, PARTSUPP PS
WHERE P_PARTKEY = PS.PS_PARTKEY AND S.S_SUPPKEY = PS.PS_SUPPKEY AND P.P_SIZE = 39
AND P.P_TYPE = 'BRASS' AND S.S_NATION = 'MOROCCO' AND S.S_NATION IN ('MOROCCO', 'SPAIN')
AND PS.PS_SUPPLYCOST = (SELECT MIN(PS1.PS_SUPPLYCOST) FROM PARTSUPP PS1, SUPPLIERS S1
WHERE P.P_PARTKEY = PS1.PS_PARTKEY AND S1.S_SUPPKEY = PS1.PS_SUPPKEY AND
S1.S_NATION = S.S_NATION)]]>
    </STMTKEY>
    <OPTGUIDELINES>
        <IXSCAN TABID="Q1" INDEX="I_SUPPKEY"/>
    </OPTGUIDELINES>
</STMTPROFILE>
</OPTPROFILE>
```

Putting an optimization profile into effect

- Create the OPT_PROFILE table in the SYSTOOLS schema:

```
CALL SYSPROC.SYSINSTALLOBJECTS('OPT_PROFILES', 'C',
   CAST (NULL AS VARCHAR(128)), CAST (NULL AS VARCHAR(128)))
```

- Prior to DB2 9.5:

```
CREATE TABLE SYSTOOLS.OPT_PROFILE (
    SCHEMA VARCHAR(128) NOT NULL,
    NAME VARCHAR(128) NOT NULL,
    PROFILE BLOB (2M) NOT NULL,
PRIMARY KEY ( SCHEMA, NAME ));
```

- Compose document, validate, insert into table with qualified name

  Inserts inventory_db.xml from current directory into the SYSTOOLS.OPT_PROFILE table with qualified name "DBA"."INVENTDB"

```
File profiledata:
    "DBA","INVENTDB","inventory_db.xml"

IMPORT FROM profiledata OF DEL MODIFIED BY LOBSINFILE
INSERT INTO SYSTOOLS.OPT_PROFILE;
```

Putting an optimization profile into effect (cont.)

- At the package level using the *optprofile* bind option
- In DB2 9.7 use ALTER PACKAGE:
  - **ALTER PACKAGE DB2USER.EMPADMIN OPTIMIZATION PROFILE DB2USER.JOINHINT**

Bind optimization profile "DBA"."INVENTDB" to the package "inventapp"

```
db2 prep inventapp.sqc bindfile optprofile DBA.INVENTDB
db2 bind inventapp.bnd
```

- At the dynamic statement level: using *current optimization profile* special register

```
EXEC SQL SET CURRENT OPTIMIZATION PROFILE = 'DBA.INVENTDB';
/* The following statements are both optimized with 'DBA.INVENTDB' */
EXEC SQL PREPARE stmt FROM SELECT ... ;   EXEC SQL EXECUTE stmt;
EXEC SQL EXECUTE IMMEDIATE SELECT ... ;

EXEC SQL SET CURRENT SCHEMA = 'JON';
EXEC SQL SET CURRENT OPTIMIZATION PROFILE = 'SALES';
/* This statement is optimized with 'JON.SALES' */
EXEC SQL EXECUTE IMMEDIATE SELECT ... ;
```

Optimization Guidelines

- **Access path guidelines**
  - Base access request
    - **Method to access a table e.g. TBSCAN, IXSCAN**
  - Join request
    - **Method and sequence for performing a join e.g. HSJOIN, NLJOIN, MSJOIN**
    - **IXAND star joins**
- **Query rewrite guidelines**
  - IN-list to join
  - Subquery to join
  - NOT EXISTS subquery to anti-join
  - NOT IN subquery to anti-join
- **General optimization guidelines**
  - REOPT (ONCE/ALWAYS/NONE)
  - DEGREE
  - QUERYOPT
  - RTS
  - MQT choices

Putting an optimization profile into effect

- Clearing the special register:

```
EXEC SQL SET CURRENT OPTIMIZATION PROFILE = NULL;
/* The following statement is optimized with the setting of the OPTPROFILE bind option */
EXEC SQL PREPARE stmt FROM SELECT ... ;   EXEC SQL EXECUTE stmt;

EXEC SQL SET CURRENT OPTIMIZATION PROFILE = "";
/* The following statement is optimized with no optimization profile */
EXEC SQL PREPARE stmt FROM SELECT ... ;   EXEC SQL EXECUTE stmt;
```

- At the dynamic statement level: using *db2_optprofile* *CLI* option

-- after each successful connect to the SANFRAN database, the CLI client would issue the command:

SET CURRENT O TIMIZATION PROFILE=J LES.

```
[SANFRAN]
DB2_OPTPROFILE JON.SALES
```

Putting an optimization profile into effect

- SQL procedures

```
CALL SET_ROUTINE_OPTS('OPTPROFILE DBA.INVENTDB ') %

CREATE PROCEDURE MY_PROC
BEGIN
        DECLARE CUR1 CURSOR FOR SELECT ...
END %
```

- SQL may be modified during CREATE PROCEDURE processing
- Use explain facility or query system catalogs to get modified SQL statements to include in optimization profile STMTKEY element for profile statement matching

```
SELECT STMTNO, SEQNO, SECTNO, TEXT
  FROM SYSCAT.STATEMENTS AS S,
       SYSCAT.ROUTINEDEP AS D,
       SYSCAT.ROUTINES   AS R
 WHERE PKGSCHEMA = BSCHEMA
   AND PKGNAME = BNAME;
   AND BTYPE = 'K'
   AND R.SPECIFICNAME = D.SPECIFICNAME
   AND R.ROUTINESCHAME = D.ROUTINESCHEMA
   AND ROUTINENAME = ?
   AND ROUTINESCHEMA = ?
   AND PARM_COUNT = ?
 ORDER BY STMTNO
```

- STMNO should be the line number in the source code of the CREATE PROCEDURE, relative to the beginning of the procedure statement (line number 1)

Table references in views

- **Example**

```
CREATE VIEW “DBGuy".V1 as (SELECT * FROM EMPLOYEE A WHERE SALARY >
   50,000) ;
CREATE VIEW DB2USER.V2 AS (SELECT * FROM “DBGuy".V1 WHERE DEPTNO
   IN (’52’, ’53’,’54’) ;
SELECT * FROM DB2USER.V2 A WHERE V2.HIRE_DATE > ’01/01/2004’ ;
```

```
<OPTGUIDELINES><IXSCAN TABLE=’A/“DBGuy".V1/A’/></OPTGUIDELINES>
```

- Extended syntax allows unambiguous table references in views
  - ‘A’ is ambiguous
- Extended name consists of exposed names in the path from the statement reference to the nested reference separated by slashes
- Same rules for exposed names apply to extended syntax

Table references in views

- Extended syntax is not necessary if references are unique with respect to all table references in the query

- **Example**

```
CREATE VIEW “DBGuy".V1 as (SELECT * FROM EMPLOYEE E WHERE SALARY >
   50,000) ;
CREATE VIEW DB2USER.V2 AS (SELECT * FROM “DBGuy".V1 WHERE DEPTNO
   IN (’52’, ’53’,’54’) ;
SELECT * FROM DB2USER.V2 A WHERE V2.HIRE_DATE > ’01/01/2004’ ;
```

<OPTGUIDELINES><IXSCAN TABLE=‘E’/></OPTGUIDELINES>

Ambiguous table references

- **Example**

```
CREATE VIEW V1 AS
  (SELECT * FROM EMPLOYEE WHERE SALARY >
      (SELECT AVG(SALARY) FROM EMPLOYEE);

SELECT * FROM V1 WHERE DEPTNO IN ('M62', 'M63') ;
<OPTGUIDELINES><IXSCAN TABLE='V1/EMPLOYEE'/></OPTGUIDELINES>
```

- Which EMPLOYEE reference?
- The IXSCAN request is ignored
- Uniquely identify EMPLOYEE by adding correlation names in the view
- Use TABID
  - Correlation names in the optimized SQL are always unique