Information Management

Data Management

Writing and Tuning Queries for Optimal Performance

John Hornibrook, DB2 Query Optimization Development
IBM Canada

Avoid complex expressions in search conditions

- Avoid join predicates on expressions
  - Limits the join method to nested loop
    - No hash or merge sort join possible
  - Prevents accurate selectivity estimates

```
WHERE SALES.PRICE * SALES.DISCOUNT = TRANS.FINAL_PRICE
WHERE UPPER(CUST.LASTNAME) = TRANS.NAME
```

- Consider using a generated column

Avoid complex expressions in search conditions

- Avoid expressions over columns in local predicates
  - Prevents the use of index start and stop keys
  - Results in inaccurate selectivity estimates
  - Requires extra processing at query execution time
  - Use the inverse of the expression
    - Instead of this:

```
<expression>(C) = 'constant'
INTEGER(TRANS_DATE)/100 = 200802
```

    - Do this:

```
C = <inverse-expression>('constant')
TRANS_DATE BETWEEN ‘2008-02-01’ AND ‘2008-02-29’
```

Avoid complex expressions in search conditions

- Watch out for views!

```
CREATE VIEW CUST_V AS
(SELECT LASTNAME, (CUST_ID * 100) + INT(CUST_CODE) AS CUST_KEY
FROM CUST)


SELECT LASTNAME FROM CUST_V WHERE CUST_KEY = 123456
```

- The query looks innocent, but view merging results in:

```
SELECT LASTNAME FROM CUST WHERE (CUST_ID * 100) +
INT(CUST_CODE) = 123456
```

Avoid complex expressions in search conditions

- Consider using generated columns when the inverse function is difficult to express
- LASTNAME IN ('Woo', 'woo', 'WOO', 'WOo', and so on)

```
CREATE TABLE CUSTOMER
(LASTNAME VARCHAR(100),
U_LASTNAME VARCHAR(100) GENERATED ALWAYS AS (UCASE(LASTNAME)))

CREATE INDEX CUST_U_LASTNAME ON CUSTOMER(U_LASTNAME)

SELECT CUST_ID FROM CUSTOMER WHERE U_LASTNAME = UCASE(‘Woo’)
```

- Consider using case-insensitive search in V9.5 FP1 for this particular example, however, it applies to the entire data base.

Avoid multiple aggregations with the DISTINCT keyword

- If multiple distinct aggregations can't be avoided, consider:
  - **DB2_EXTENDED_OPTIMIZATION = ENHANCED_MULTIPLE_DISTINCT**
  - Input stream is read once and shared by each UNION arm
  - Applies only to DPF environments
  - DPF considerations:
    - May improve performance where the ratio of processors to the number of database partitions is low e.g. <= 1
    - Otherwise multiple arms may benefit from parallelization
  - Performance testing necessary before use in production

Use OPTIMIZE FOR N ROWS clause with FETCH FIRST N ROWS ONLY clause

- OPTIMIZE FOR N ROWS
  - Indicates to the optimizer that the application intends to only retrieve N rows, but the query will return the complete result set
  - Optimizer will favor ‘piped’ plans
    - Avoids buffering operations such as temporary tables, sorts, hash joins
- FETCH FIRST N ROWS ONLY
  - Indicates that the query should only return N rows
- Optimizer doesn’t automatically assume OPTIMIZE FOR N ROWS when FETCH FIRST N ROWS ONLY is specified for the outer subselect
- Try specifying both

Optimization classes

- Use greedy join enumeration
  - 0 - minimal optimization for OLTP
    - use index scan/nested loop join
    - basic set of query rewrite rules
  - 1 - low optimization
    - consider merge scan join and table scans
    - subset of query rewrite rules
  - 2 - full optimization, limit space/time
    - use same query transforms & join strategies as class 5
- Use dynamic programming join enumeration
  - 3 - moderate optimization
    - rough approximation of DB2 for z/OS
  - 5 - self-adjusting full optimization (default)
    - uses all techniques with heuristics
  - 7 - full optimization
    - similar to 5, without heuristics
  - 9 - maximal optimization
    - spare no effort/expense
    - considers all possible join orders, including Cartesian products!

Reducing optimization time

- If reducing optimization class doesn’t reduce optimization time sufficiently OR
- Lower optimization classes aren’t appropriate for workload
- Consider setting **DB2_REDUCED_OPTIMIZATION** registry variable
- Provides more control over optimizer's search space than optimization class

Manually updating statistics

- Statistics values are...
  - readable in the system catalogs
    - e.g., HIGH2KEY, LOW2KEY
  - updateable, e.g.

    ```
    UPDATE SYSSTAT.TABLES
    SET CARD = 1000000 WHERE TABNAME = `NATION'
    ```
- Implications:
  - Can simulate a non-existent database
  - Can "clone" a production database (in a test environment)
    - db2look tool
- However:
  - Don’t ‘fake’ the statistics to fool the optimizer!
  - May fix some queries, but others may degrade
  - Follow best-practices for query tuning first

As a last resort…optimization profiles

- Mechanism to control statement optimization
  - Can control both query rewrite optimization and access path optimization
- Sets of explicit optimization guidelines
  - “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**
  - RUNSTATS, indexes, optimization class, DB and DBM configs, etc.
  - caution: results in circumvention of usual cost-based optimization
- Available in DB2 9