SQL Best Practices Tips

The purpose of this document is to establish best practices standards to be used for the development of DB2 SQL on IBM System i. Most relational tuning experts agree that the majority of performance problems among applications that access a relational database are caused by poorly coded programs or improperly coded SQL. This document provides a reference for developers for SQL development coding standards.

SQL developer goals

You should always code with two goals in mind.

The first and most important is to get the data right. Whatever data you are processing or retrieving needs to be correct. Data integrity and correct reporting is number one in processing.

The second goal is to get the processing executed as quickly as possible. The one thing (standard) that you should always know is how much data was processed during the runtime. Did the program process 10,000 rows of data or 10 million? How many OPEN CURSORS, SELECTS, UPDATES, DELETES, and so on occurred? The first question when there is a performance issue is typically: How much data was processed?

DB2 SQL Best Practices rules

1. Never put SQL scalar functions on columns in the WHERE predicates. When placing SQL scalar functions on columns in the SELECT portion of the SQL statement does incur some minimal overhead but applying it to a column in the WHERE clause causes the predicate to become stage 2 non indexable.

For example:

```
WHERE YEAR(PAYDATE) = 2014

should be recoded as

WHERE PAYDATE BETWEEN '2014-01-01'and '2014-12-31'
```

2. Same applies for mathematics. Having the mathematics on the host variable is not a problem.

For example:

```
WHERE PAYDATE - 7 DAYS > :HV_DATE

should be recoded as

WHERE PAYDATE > :HV_DATE + 7 days.
```

3. Don't return unnecessary rows. *(Filter, Filter, Filter)*

4. Don't filter rows in the application that DB2 can filter i.e. (Java or RPG).

5. Code only the columns needed in the SQL SELECT. Having extra columns can have an effect on the optimizer's choice of index only processing, larger sort costs, and at times the join type chosen.

```
SELECT PART_NUMBER, PART_DESCRIPTION, PART_TYPE
FROM PART_MASTER
WHERE PART_STATUS = '1';
```

6. Avoid **"SELECT *"** unless really needed.

7. Use Multi-row FETCH.

When using multi-row fetch

- Avoid GET DIAGNOSTICS due to high CPU overhead
- Use the SQLCODE field of the SQLCA

- Fetch was successful (SQLCODE 000)
- Fetch failed (negative SQLCODE)
- End of file (SQLCODE 100)

8. Minimize SQL traffic or DB2 calls on
   Select from Insert / Update / Delete

Example:

```
/* Generate a unique id for the next customer */
SELECT CUSTID
FROM FINAL TABLE
 (INSERT INTO CUSTOMERS (CUSTID, CUSTNAME)
  VALUES
   (NEXT VALUE FOR CUSTSEQ, 'John Roberts'))
```

This example saves multiple calls to DB2.

9. Avoid unnecessary execution of SQL Consider accomplishing as much as possible with a single call, rather than multiple calls

10. Avoid Unnecessary Sorts
    Sorts can be expensive

11. Specify "FOR FETCH ONLY" on cursors that aren't going to be used for positioned update / delete. Default is to assume cursor might be used for positioned delete

example:

```
select * from INVMST for fetch only;
```

12. Specify “NO SCROLL” for cursors that don’t need scrolling (or take the default)
    Scrolling adds overhead, even if only fetching in a forward direction
    Some query optimizations can only be done on non-scroll cursors

13. Watch out for any ORDER BY, GROUP BY, DISTINCT, UNION, INTERSECT, and EXCEPT. These may cause sorts to occur in the processing. Make sure they are truly needed in the query.

14. When comparing column values to host variables - use the same
    Data Type and Length

    When DB2 must convert data, available indexes are sometimes not used

15. Minimize SQL requests from your application program. This is especially true in batch processing where the number of statements executed is typically high. Every time a SQL request is sent to DB2, there is overhead involved because it has to be sent from one address space in the operating system (for example RPG or Java) to the DB2 address space. In general, developers should minimize these requests by reducing the number of times they open and close cursors, execute select statements, and so on. DB2 has multi-row processing for developers specifically for this reason, where you can process more than one row at a time. Developers need to code more relationally and less procedurally.

~~To see SQLRPGLE example: go to link~~

If you are working on a query or program for performance, where do you start? Please following these steps:

- Check every predicate. Are they indexable and stage 1? Can you rewrite them more efficiently? Any stage 2 predicates?

- Can you rewrite any predicate different and contain the same logic? Sometime even rewriting 1 predicate may send the optimizer down a different access path.

- Check the number of times SQL requests are being sent to DB2. Is there any way to rewrite/redesign the program to minimize the number of requests being processed?

- Check the DB2 Explain output. Is there a tablespace scan? Any sorts? Any index processing with 0 matching column? Most times, developers want to see index processing chosen for their queries.

- Any subqueries involved? Try rewriting an IN subquery to an EXISTS subquery. At times, they optimize differently.

- Are there multiple subqueries? Make both subqueries the same type (correlated versus non-correlated) and then put most restrictive one first.