Performance basics for row and column access control

Sandy Ryan

November 27, 2014
(First published November 27, 2014)

IBM DB2 for i version 7.2 has the new database security capability, row and column access control (RCAC). RCAC provides the capability to control data access to the record and column level. Specified through SQL statements, though it controls all access to the enabled tables, performance is a consideration when using RCAC. This article discusses the basic factors of RCAC performance and provides examples of the performance effects on OLTP workloads.

Introduction

The IBM i 7.2 release includes new DB2 for i security capabilities known as RCAC. RCAC provides the capability to control data access to the record and column level. It is designed to provide an extra layer of data security at the database level and to make data easier to control.

DB2 for i clients have a strong business motivation to engage and deploy RCAC for the following reasons:

- RCAC is a data-centric technology.
- RCAC is a no-charge feature.
- RCAC can be used with SQL or DDS-created tables.

To understand the details behind any of these assertions, refer to the RCAC IBM Redpaper™ listed at the end of this article. As with most database technologies, you'll have greater success if you consider all aspects (that is, have a complete strategy) before moving forward. This article covers performance topics related to RCAC that are not covered elsewhere.

The SQL mechanisms to specify RCAC are row permissions and column masks, which define the rules for allowing access to the rows and columns for a database file. You can define and enable row permissions and column masks using the `CREATE PERMISSION` and `CREATE MASK` SQL statements.

- `CREATE PERMISSION` defines a rule that is automatically applied by the database to determine the set of rows that are visible.

- `CREATE MASK` defines a rule that is automatically applied when the target column is the target of selection. A CASE expression indicates the portion of the column data that is returned to the user or the application.

Both, the `CREATE MASK` and `CREATE PERMISSION` statements, are very powerful and flexible.

SQL workload with RCAC

As expected, with this power and flexibility you need to exercise caution and good SQL tuning practices to maintain good application performance. The framework of RCAC itself has been implemented very efficiently to have minimal effect on SQL query performance. To ensure this, an OLTP SQL workload was used to measure the effects of RCAC on workload performance and the following four scenarios were considered:

1. Baseline (without RCAC).
2. Add row permissions: Row permissions were created over all tables in the workload.
3. Add row permissions and column masks:
   Row permissions were created over all tables in the workload and column masks were created over all frequently selected columns.
4. Use SQL views to emulate the row permissions.

The RCAC rule text for the masks and permissions was a simple invocation of the `VERIFY_GROUP_FOR_USER` built-in function. The `VERIFY_GROUP_FOR_USER` function was added in the IBM i 7.2 release to provide an easy and fast mechanism to determine whether a user is a member of a specific group profile. The permissions, masks, and views used in the workload measurements were similar to those shown in examples 1, 2, and 3.

Listing 1. Row permission used to evaluate performance impact

```
CREATE PERMISSION library.permissionname
ON library.tablename
FOR ROWS WHERE
        VERIFY_GROUP_FOR_USER(SESSION_USER, ‘groupname’) = 1
ENFORCED FOR ALL ACCESS
```

Listing 2. Column mask used to evaluate performance impact

```
CREATE MASK library.maskname
ON library.tablename
FOR COLUMN columnname
RETURN  CASE
        WHEN VERIFY_GROUP_FOR_USER ( SESSION_USER , 'groupname' ) = 1
        THEN columnname
        ELSE 'XXXXXXXX'
END
```

Listing 3. SQL view used to emulate a row permission

```
CREATE VIEW library.viewname as
        SELECT *
        FROM library.tablename
        WHERE VERIFY_GROUP_FOR_USER(SESSION_USER, 'groupname') = 1;
```

The workload was measured at several memory pool sizes to determine whether the RCAC framework increased the workload memory usage. As shown in Figure 1, the average response

time was negligibly affected with RCAC. There was no measureable difference in average transaction response time for the RCAC permission implementation compared to the view equivalent. The difference in the average transaction response time for the scenario with RCAC permissions and masks compared to the baseline with no security checks was in the 5% to 6% range.

Figure 1. Average transaction response time against memory pool size

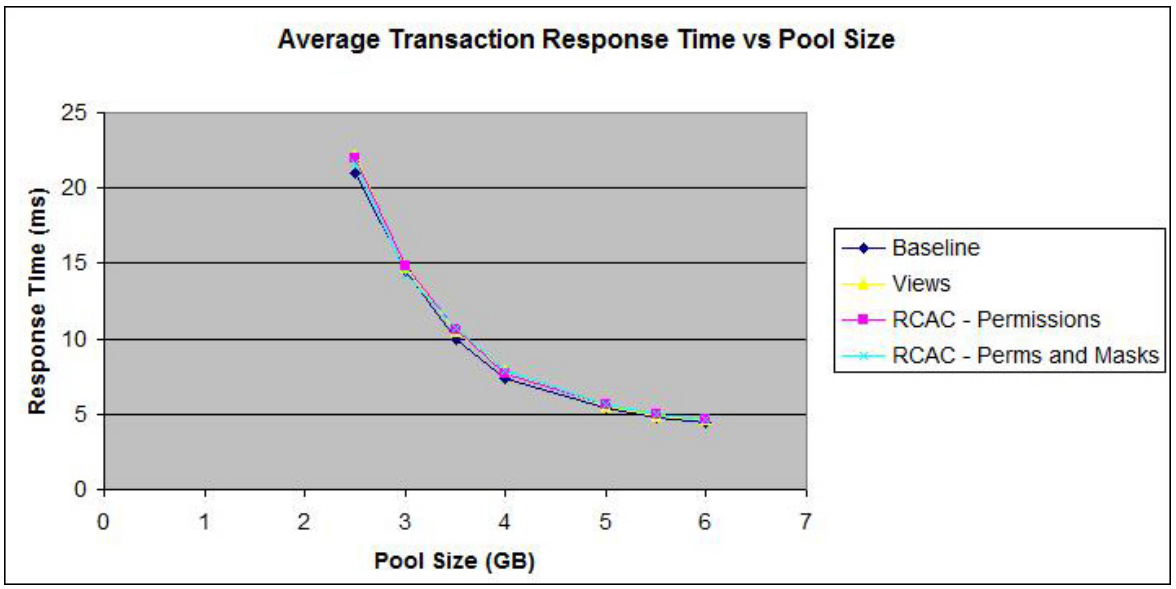
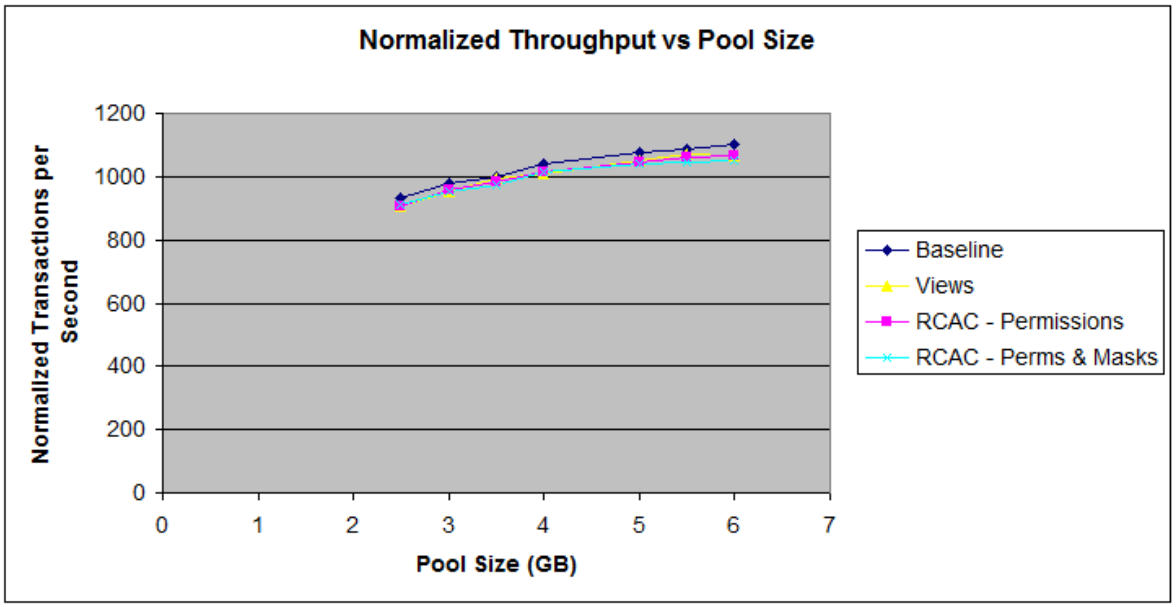


Figure 2 shows normalized throughput against memory pool size for the four scenarios. Normalized throughput is the transactions per second at a constant level of processor utilization. The higher the normalized transactions per second, the lower the processor usage of the workload per transaction. There was 1% or less difference in the normalized throughput for the RCAC permissions when compared with view implementations. There was less than 5% difference for the RCAC permissions and masks when compared with the baseline with no extra security checks.

Figure 2. Normalized throughput against memory pool size



As with any performance data, results might vary for different configurations and workloads. For the OLTP workload used in this study, the measurement data shows that the basic RCAC implementation itself has little impact on processor or memory usage for SQL queries. However, the permission and mask creation statements allow for the creation of complex checks in the masks and permissions. If the row permission rule text includes a complex and/or data-intensive subquery, the performance of an application could be adversely impacted. The same applies to the `CREATE MASK` definition. With every row or column accessed in a table, which has at least one active permission or mask, the conditions of that permission and/or mask must be checked. You can think of it as similar to potentially running another query for each row or column. So, you need to be very careful while defining permissions and masks. Performance testing and tuning before production use is highly recommended.

Native record-level access with RCAC

While RCAC is controlled through SQL statements the security settings must apply for any access to tables, either through SQL or native record-level access. RCAC is implemented only in the SQL Query Engine (SQE), not the Classic Query Engine (CQE). Special processing is done to enable native access, traditionally implemented through CQE, to automatically go through the SQE path. This might be advantageous for performance, as SQE, with advanced optimization and I/O techniques, is far more efficient than CQE. However, there is additional processing in the query open phase to enable the native query to be processed through SQE. In general, the more the data to be processed in the query, the more the efficiency gained by SQE which can mitigate or even surpass the additional processing during the open phase.

To understand the effect of RCAC on the performance of an OLTP workload with native access, measurements were done on a workload of an order processing application with the transactions using native access to the database with COBOL. For this workload, the additional processing

to enable SQE to do the database queries increased the processor use by an additional 4%. However, the more efficient I/O implementation in SQE improved the disk usage by 2% to 6%, offsetting some of the cost to transition a native open to SQE.

In native record-level access workloads in which there are permissions and masks on the tables, the native to SQE performance effects would be in addition to any performance impacts from the permission and mask processing. In this workload, with `VERIFY_GROUP_FOR_USER` checks in permissions over all the tables in the workload, the workload increased processor usage by 7%, where 4% related to the native to SQE processing and 3% was associated with the row permission implementation.

The effect of RCAC on the performance of this OLTP workload shows the overall workload impacts of RCAC with native record-level access to be relatively small. However, the performance impact might vary by workload and configuration. Thus, performance testing is essential when implementing RCAC.

Conclusion

RCAC provides very powerful new functionality to secure data. However, with these capabilities there are some performance considerations. Testing of OLTP benchmarks has shown that the RCAC framework itself has little impact on performance for SQL workloads and only a small effect for SQL queries over tables with permissions and masks employing the new `VERIFY_GROUP_FOR_USER` function. RCAC does impact the performance of native record-level access due to the open phase requiring more processor usage. While the impact of this additional processing has been measured as relatively small on an OLTP workload, the SQL implementation is faster. Applications that are modernized from the native record-level access to SQL can realize the benefit of this faster implementation. In general, care must be taken in the definition of permissions and masks, as complex permission and mask security checks can adversely affect performance as they might slow down data access. Performance testing and tuning before production use is highly recommended.

Resources

- IBM Knowledge Center - Row and column access control (RCAC)
- Redpaper - Row and Column Access Support in IBM DB2 for i
- DB2 for i forum