JDEtips
JD Edwards Knowledge. Transferred. Sustained.

April 2015 On Technical/Security

Row Security and System Performance

By Nathan Beaton, ALLOut Security

E1 Editor's Note: If you are avoiding row security in EnterpriseOne due to concerns about performance, this article will help you understand how you can have it both ways, or at least lessen the amount of performance degradation through effective use of indices.

Introduction

Row security is an often-misunderstood security type, and one of the main reasons that I hear for not implementing it is because of performance concerns. The two main discussions that I hear usually surround the number of row security records and the inclusive/exclusive setting. However, this article shows that those two common concerns are not the most likely to cause major performance concerns. While some implementations of row security can limit system performance, this does not mean that row security should be avoided. The reasons for performance degradation are complex, and proper understanding and configuration can provide the added benefit of security without hurting performance.

When row security is applied, the system modifies the SQL statements that are performed against the database. More specifically, it is the WHERE clause of the SQL statement that is affected. For example, let's say that we had security configured against the address book table, F0101, for the data item MCU such that only business unit 'STORE20' were visible.

The resulting SQL statement would be as follows.

```
SELECT …
FROM   F0101
WHERE  ABMCU = 'STORE20';
```

For comparison, a system with no row security would present a SQL statement like the one below.

```
SELECT …
FROM   F0101;
```

Factors Affecting Performance

Exclusive vs. Inclusive Mode

JDE provides two different configuration settings for row security: one can define what users **can** see; i.e., inclusive, or what they **cannot** see; i.e., exclusive. Generally, inclusive row security is seen as the better choice. Not only is maintenance easier, but performance for certain database configurations is more efficient with inclusive than with exclusive.

Sample Inclusive Query

```
SELECT …
FROM   F0101
WHERE  ABMCU = '          10';
```

Sample Exclusive Query

```
SELECT …

FROM   F0101

WHERE  ABMCU NOT BETWEEN '             ' AND    '           9'

AND    ABMCU NOT BETWEEN '          11' AND    'ZZZZZZZZZZZ';
```

Number of Row Security Records

While the number of row security records does indeed affect query performance, the impact is minimal for most moderate row security implementations. Moving to extreme examples, we do start to see a performance degradation of about 13% when several hundred lines exist in a single query.

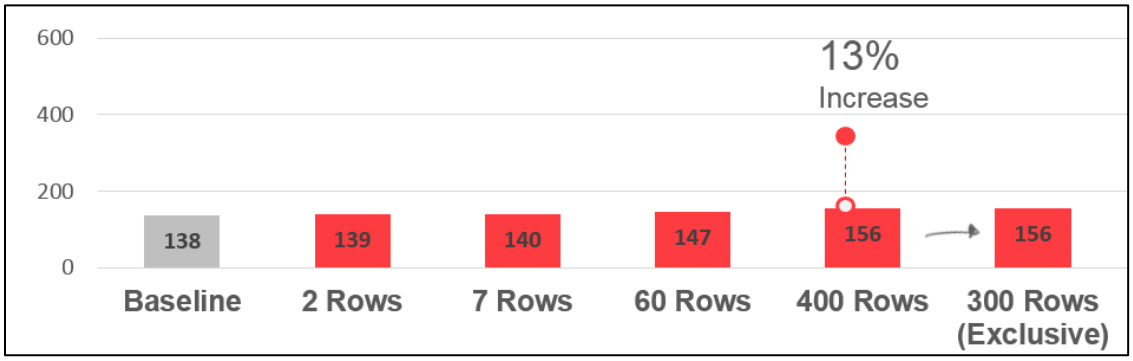

Figure 1: Query Performance for n Records in F00950

(Numbers represent estimated query resource cost from Oracle's Autotrace feature.)

Database Architecture

Why is this important?

To understand row security performance more fully, we need to dive into database architecture for a brief moment. Imagine you had a list of randomly sorted data, and you wanted to programmatically count all of the instances of a specific value. This would require that every single value be analyzed because it would be impossible to say that no more instances of that value existed until the final record were reached. However, sorting the values provides a way to guarantee that every instance of a value is found without having to review every record.

Indices

Database indices work in a similar, but more complex, fashion. Clearly, a sorted data set is going to provide much better performance; but tables often have many fields, and an efficient sort for one field might be completely inefficient for another. Therefore, a table can contain multiple indices each with different sort configurations.