



## Row Oriented Security Using Triggers

This is an abstract only. Details may be found here: <http://www.newdalesystems.com/articles/Article014.htm>

### Overview

This project created an abstract model for implementing row-oriented security in any client-server relational database. It is independent of the database server's own security model, although blending features from both could be advantageous. The model is generic, and may be applied to any relational database. It is based on a small number of security tables with a fixed interpretation but, subject to that, application developers may deploy it in a variety of different ways. Some may wish to impose row- and column-oriented security within their applications, while others may simply want to attach their legacy code to a fixed server view with minimal re-programming.

The basic premise of the model is that some user tables act as gateways to all the other tables. Row-oriented security involves using the security tables to define which rows of those gateways may be accessed by a given user. That user may access related rows in other tables if access has been granted to the corresponding gateway rows.

Microsoft's SQL Server was used as the relational database in this project (it does not offer row-level security). However, Oracle's Virtual Private Database and Label Security technologies provide row-level security add-ons that require custom programming by the end-user. One of the features of Oracle's approach is to silently re-write user queries, based on their security clearance. A much simpler approach<sup>1</sup> would be to place an "owner" column on each table, where stored procedures are used to view, add, delete or modify rows. Only that user who "owns" a particular row may access it using those procedures. The stored procedure for adding a new row automatically sets the owner to the user who created it. Although this is very secure, it prevents different users from viewing the same row.

The security in SQL Server is view- and column-oriented, but this project was concerned with row-oriented security that allows groups of users to access some but not necessarily all rows in each table. However, it also has its own column-oriented security. Therefore, the model is self-contained and independent of any security imposed by the underlying database management system.

For example, if three people from three Departments form a group that is responding to an RFP, then they should be able to view each other's information, except perhaps for personal information such as SIN numbers. That would involve both row- and column-oriented security.

This model allows scenarios like that, using special security tables that designate what roles users belong to, what rows each role may access from the tables in the database, and what columns each role may see in selected tables.

It assumes that users may access the database *only through approved applications*. This is required because the model puts a small part of the security burden on the database front-end applications rather than the database server.

This model has been successfully used by a large organization for several years.

---

<sup>1</sup> [http://vyaskn.tripod.com/row\\_level\\_security\\_in\\_sql\\_server\\_databases.htm](http://vyaskn.tripod.com/row_level_security_in_sql_server_databases.htm)

## Functionality

Users, roles and security restrictions are defined by an organization.

A security restriction may involve any set of rows of a table, any set of columns of a table, or just those rows of a table defined by a filter.

Users may be assigned any number of roles. They automatically inherit the “sum” of all current security restrictions for those roles when logging in.

The filtering mechanism for invoking security is very flexible:

- If any row becomes visible to a security filter (because some underlying data has suddenly changed) then that row is immediately accessible to any user whose security includes that filter.
- Likewise, if any row becomes invisible, then that row is no longer accessible by any user whose security is includes that filter.
- But if a user creates new data, then that data is automatically available to that user, regardless of what filters are used to define that user’s security restriction. Security may be defined in the model by selecting particular rows in a table, rather than using filters. Such selections override filters, for this and other reasons.

## Implementation

The entire system was written using SQL triggers and stored procedures. So it is basically independent of any desktop or web application, which need only include by some simple, front-end security code. That’s why only “approved” applications should be allowed to access the database. Of course, one may by-pass the security model directly since its triggers only fire for those applications with the front-end code. This allows developers to write and test applications that are security-free, before imposing user restrictions prior to deployment.

A security management program is available<sup>2</sup> to interactively define users, roles and restrictions. Any changes to security are immediately witnessed by all users, except if they’re editing something whose security has just changed. In that case, such changes are realized when they log in again.

This system is ideal for a Citrix server farm, where only approved applications are published. The stored procedures are Citrix-aware, so that multiple sessions using a single copy of an application will work together seamlessly.

One administrator should be responsible for maintaining security, to avoid unintended restrictions that can be spawned by casual use of the model’s tables.

## Example

A University faculty could use this model to allow the Dean’s office to access anything, while each Department is restricted to just that data belonging to them. Temporary workers in a given Department could be given access to selected rows in selected tables (or selected filters) containing that Department’s data. Some could be further restricted to read-only access of certain columns. All of them could be denied viewing access for selected columns (eg. SIN numbers).

An inter-disciplinary committee from several Departments could be given rights to see each other’s data (except for, say, salary and appointment information). Usually, the administrator would define a special role for them to do that. That role would be removed later when the committee dissolves.

---

<sup>2</sup> Windows version only