Paper TS02

Creating a personal view on the data via using a Virtual Private Database

Sébastien Roland, Business & Decision Life Sciences, Brussels, Belgium

ABSTRACT

Most of simple applications have been thought as monolithic blocks. They are viewed as a multi-layered system where the data layer is the last Russian doll. Smart users could however directly access that last layer, bypassing others where the security model is sketched.
In application whose architecture is complex (eg. where different systems interact with the database), it is vital to protect data directly at the source, as it is directly exposed to multiple tiers. The threat not only comes from the 'outside world' but also from the user that has been granted an access, who knows a little bit of SAS or SQL and wants to access more data.
That is the goal designed when building Virtual Private Databases: add security directly where data reside. Oracle dynamically adds conditions to user requests depending on a context. This gives security at the row level, preventing inside threats.

KEY WORDS

Oracle, Virtual Private Database, SAS, SDD, views, security

ABOUT THIS PAPER

The ***first section*** of this paper describes generic database terms that we will further use. Readers who are familiar with databases may skip that section.

In the ***second section*** we will describe more in depth what are Virtual Private Databases, its benefits and how to implement it.

The theory in head, we will go through a practical example of how to secure a solution in the ***third section***. That example will build up security incrementally, describing for each step the security issue and how to solve it.

The ***fourth section*** adds more complexity to our use-case scenario, scripts will be given to completely implement a Virtual Private Database.

TABLE OF CONTENTS

Abstract *1*

Key Words *1*

About this paper *1*

1. Background *3*

1.1. Database **3**

1.2. Database Management System (DBMS) **3**

1.3. Relational Model **3**

1.4. (Object) Relational Database Management System ((O)RDBMS) **3**

1.5. Schema **4**

1.6. Views **4**

1.7. Privileges and Roles **5**

2. Virtual Private Database (VPD) *6*

2.1. What is Virtual Private Database **6**

2.2. Benefits of Using Oracle Virtual Private Database Policies **6**

2.2.1. Basing Security Policies on Database Objects Rather Than Applications 6

2.2.2. Controlling How Oracle Database Evaluates Policy Functions 7

2.3. Attaching a Policy to a Database Table, View, or Synonym **7**

3. Securing database *8*

3.1. The problem **8**

3.2. Views **11**

3.3. Virtual Private Database **11**

4. VPD with a file & PERMISSIONS server *12*

5. CONCLUSION *14*

References *14*

Software used *14*

Contact Information *14*

1. BACKGROUND

1.1. Database

Every organization has information that it must store and manage to meet its requirements. This information must be available to those who need it. An information system is a formal system for storing and processing information. Most companies today use a database to automate their information systems. A database is an organized collection of information treated as a unit. The purpose of a database is to collect, store, and retrieve related information for use by database applications.

1.2. Database Management System (DBMS)

A database management system (DBMS) is software that controls the storage, organization, and retrieval of data. Typically, a DBMS has the following elements:

- ***Kernel code***: this code manages memory and storage for the DBMS.
- ***Repository of metadata***: this repository is usually called a data dictionary.
- ***Query language***: this language enables applications to access the data.

A database application is a software program that interacts with a database to access and manipulate data.

1.3. Relational Model

In his seminal 1970 paper "A Relational Model of Data for Large Shared Data Banks," E. F. Codd defined a relational model based on mathematical set theory. Today, the most widely accepted database model is the relational model.

A relational database is a database that conforms to the relational model. The relational model has the following major aspects:

- ***Structures***: well-defined objects store or access the data of a database.
- ***Operations***: clearly defined actions enable applications to manipulate the data and structures of a database.
- ***Integrity rules***: govern operations on the data and structures of a database.

A ***relational database*** stores data in a set of simple relations. A ***relation*** is a set of tuples. A ***tuple*** is an unordered set of attribute values.

A ***table*** is a two-dimensional representation of a relation in the form of rows (tuples) and columns (attributes). Each ***row*** in a table has the same set of columns. A relational database is a database that stores data in relations (tables). For example, a relational database could store information about company employees in an employee table, a department table, and a salary table.

1.4. (Object) Relational Database Management System ((O)RDBMS)

The relational model is the basis for a relational database management system (RDBMS). Essentially, an RDBMS moves data into a database, stores the data, and retrieves it so that it can be manipulated by applications. An RDBMS distinguishes between the following types of operations:

- ***Logical operations***: in this case, an application specifies what content is required. For example, an application requests an employee name or adds an employee record to a table.

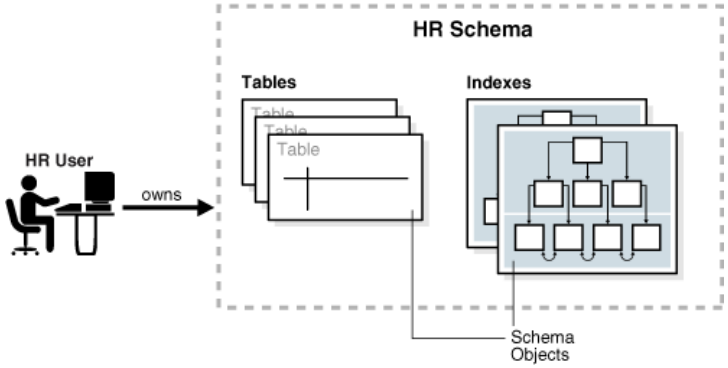
- ***Physical operations***: in this case, the RDBMS determines how things should be done and carries out the operation. For example, after an application queries a table, the database may use an index to find the requested rows, read the data into memory, and perform many other steps before returning a result to the user. The RDBMS stores and retrieves data so that physical operations are transparent to database applications.

Oracle is an RDBMS. An RDBMS that implements object-oriented features such as user-defined types, inheritance, and polymorphism is called an object-relational database management system (ORDBMS). Oracle Database has extended the relational model to an object-relational model, making it possible to store complex business models in a relational database.

1.5. Schema

A database schema is a logical container for data structures, called schema objects. Examples of schema objects are tables and indexes. Schema objects are created and manipulated with SQL.

A database user has a password and various database privileges. Each user owns a single schema, which has the same name as the user. The schema contains the data for the user owning the schema. For example, the hr user owns the hr schema, which contains schema objects such as the employees table. In a production database, the schema owner usually represents a database application rather than a person.



The most important schema objects are:

- ***Tables***: stores data in rows
- ***Indexes***: contains an entry for each indexed row of the table or table cluster and provide direct, fast access to rows. Oracle Database supports several types of index. An index-organized table is a table in which the data is stored in an index structure.
- ***Partitions***: pieces of large tables and indexes. Each partition has its own name and may optionally have its own storage characteristics.
- ***Views***: customized presentations of data in one or more tables or other views. Views do not actually contain data.
- ***Sequences***: user-created objects that can be shared by multiple users to generate integers. Typically, sequences are used to generate primary key values.
- ***Dimensions***: defines a parent-child relationship between pairs of column sets, where all the columns of a column set must come from the same table. Dimensions are commonly used to categorize data such as customers, products, and time.
- ***Synonyms***: aliases for another schema object. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.
- ***PL/SQL subprograms and packages***: PL/SQL is the Oracle procedural extension of SQL. A PL/SQL subprogram is a named PL/SQL block that can be invoked with a set of parameters. A PL/SQL package groups logically related PL/SQL types, variables, and subprograms.

Other types of objects are also stored in the database and can be created and manipulated with SQL statements but are not contained in a schema. These objects include ***database users***, ***roles***, ***contexts***, and ***directory objects***.

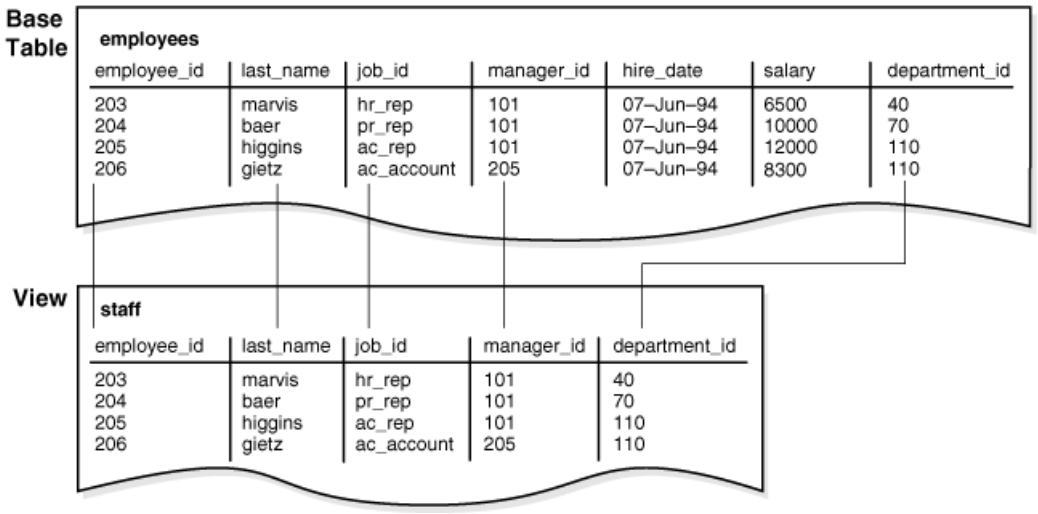
1.6. Views

A view is a logical representation of one or more tables. In essence, a view is a stored query. A view derives its data from the tables on which it is based, called base tables. Base tables can be tables or other views. All operations performed on a view actually affect the base tables. You can use views in most places where tables are used.

Materialized views use a different data structure from standard views but these are out of scope of this paper.

Views enable you to tailor the presentation of data to different types of users. Views are often used to:

- ***Provide an additional level of table security*** by restricting access to a predetermined set of rows or columns of a table.



For example, the figure above shows how the staff view does not show SALARY or HIRE_DATE columns of the base table employees.

- ***Hide data complexity***: for example, a single view can be defined with a join, which is a collection of related columns or rows in multiple tables. However, the view hides the fact that this information actually originates from several tables. A query might also perform extensive calculations with table information. Thus, users can query a view without knowing how to perform a join or calculations.

- ***Present the data in a different perspective from that of the base table:*** for example, the columns of a view can be renamed without affecting the tables on which the view is based.

- ***Isolate applications from changes in definitions of base tables***: for example, if the defining query of a view references three columns of a four column table, and a fifth column is added to the table, then the definition of the view is not affected, and all applications using the view are not affected.

For an example of the use of views, consider the HR.EMPLOYEES table from the figure above, which has several columns and numerous rows. To allow users to see only five of these columns or only specific rows, you could create a view as follows:

```
CREATE VIEW staff AS
  SELECT employee_id, last_name, job_id, manager_id, department_id
  FROM   employees;
```

Unlike a table, a view is not allocated storage space, nor does a view contain data. Rather, a view is defined by a query that extracts or derives data from the base tables referenced by the view. Because a view is based on other objects, it requires no storage other than storage for the query that defines the view in the data dictionary.

1.7. Privileges and Roles

A ***user privilege*** is the right to run a particular type of SQL statement, or the right to access an object that belongs to another user, run a PL/SQL package, and so on.

```
GRANT CREATE SESSION, accts_pay TO jward;
GRANT EXECUTE ON DIRECTORY exec_dir TO jward;
GRANT CREATE SESSION TO psmith IDENTIFIED BY password;
GRANT SELECT, INSERT, DELETE ON emp TO jfee, tsmith;
GRANT ALL ON salary TO jfee;
```

Roles are created by users (usually administrators) to group together privileges or other roles. They are a way to facilitate the granting of multiple privileges or roles to users.

2. VIRTUAL PRIVATE DATABASE (VPD)

Oracle introduced Virtual Private Database (VPD) technology with Oracle Database 8.1.5.

2.1. What is Virtual Private Database

Oracle Virtual Private Database (VPD) enables you to create security policies to control database access at the row and column level. Essentially, Oracle Virtual Private Database adds a dynamic WHERE clause to a SQL statement that is issued against the table, view, or synonym to which an Oracle Virtual Private Database security policy was applied. The VPD is deployed on the database itself, and forms an integral layer of the database that cannot be circumvented by external applications and query tools. A Database Administrator would typically create new policies to activate the Virtual Private Database.

Oracle Virtual Private Database enforces security, to a fine level of granularity, directly on database tables, views, or synonyms. Because you attach security policies directly to these database objects, and the policies are automatically applied whenever a user accesses data, there is no way to bypass security.

When a user directly or indirectly accesses a table, view, or synonym that is protected with an Oracle Virtual Private Database policy, Oracle Database dynamically modifies the SQL statement of the user. This modification creates a WHERE condition (called a predicate) returned by a function implementing the security policy. Oracle Database modifies the statement dynamically, transparently to the user, using any condition that can be expressed in or returned by a function. You can apply Oracle Virtual Private Database policies to SELECT, INSERT, UPDATE, INDEX, and DELETE statements.

For example, suppose a user performs the following query:

```
SELECT * FROM OE.ORDERS;
```

The Oracle Virtual Private Database policy dynamically appends the statement with a WHERE clause. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = 159;
```

In this example, the user can only view orders by Sales Representative 159.

If you want to filter the user based on the session information of that user, such as the ID of the user, then you can create the WHERE clause to use an application context. For example:

```
SELECT * FROM OE.ORDERS
 WHERE SALES_REP_ID = SYS_CONTEXT('USERENV','SESSION_USER');
```

However, Oracle Virtual Private Database does not support filtering for DDLs, such as TRUNCATE or ALTER TABLE statements.

2.2. Benefits of Using Oracle Virtual Private Database Policies

2.2.1. Basing Security Policies on Database Objects Rather Than Applications

Attaching Oracle Virtual Private Database security policies to database tables, views, or synonyms, rather than implementing access controls in all your applications, provides the following benefits:

- ***Security***. Associating a policy with a database table, view, or synonym can solve a potentially serious application security problem. Suppose a user is authorized to use an application, and then drawing on the privileges associated with that application, wrongfully modifies the database by using an ad hoc query tool, such as SQL*Plus. By attaching security policies directly to tables, views, or synonyms, fine-grained access control ensures that the same security is in force, no matter how a user accesses the data.

- ***Simplicity***. You add the security policy to a table, view, or synonym only once, rather than repeatedly adding it to each of your table-based, view-based, or synonym-based applications.

- ***Flexibility***. You can have one security policy for SELECT statements, another for INSERT statements, and still others for UPDATE and DELETE statements. For example, you might want to enable Human Resources clerks to have SELECT privileges for all employee records in their division, but to update only salaries for those employees in their division whose last names begin with A through F. Furthermore, you can create

multiple policies for each table, view, or synonym.

2.2.2. Controlling How Oracle Database Evaluates Policy Functions

Running policy functions multiple times can affect performance. You can control the performance of policy functions by configuring how Oracle Database caches the Oracle Virtual Private Database predicates. The following options are available:

- Evaluate the policy once for each query (static policies).
- Evaluate the policy only when an application context within the policy function changes (context-sensitive policies).
- Evaluate the policy each time it is run (dynamic policies).

2.3. Attaching a Policy to a Database Table, View, or Synonym

To attach a policy to a table, view, or synonym, you use the DBMS_RLS.ADD_POLICY procedure. You need to specify the table, view, or synonym to which you are adding a policy, and a name for the policy. You can also specify other information, such as the types of statements the policy controls (SELECT, INSERT, UPDATE, DELETE, CREATE INDEX, or ALTER INDEX).

Here is an example to show how to use DBMS_RLS.ADD_POLICY to attach an Oracle Virtual Private Database policy called secure_update to the HR.EMPLOYEES table. The function attached to the policy is check_updates.

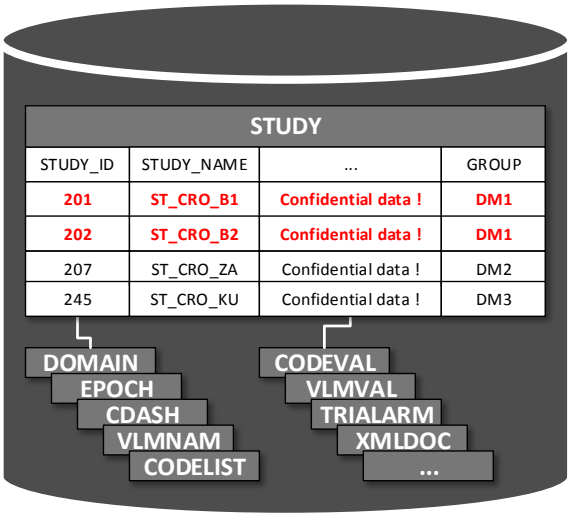
```
BEGIN
 DBMS_RLS.ADD_POLICY(
  object_schema     => 'hr',
  object_name       => 'employees',
  policy_name       => 'secure_update',
  policy_function   => 'check_updates',
  statement_types      =>  'SELECT,INDEX');

...
```

3. SECURING DATABASE

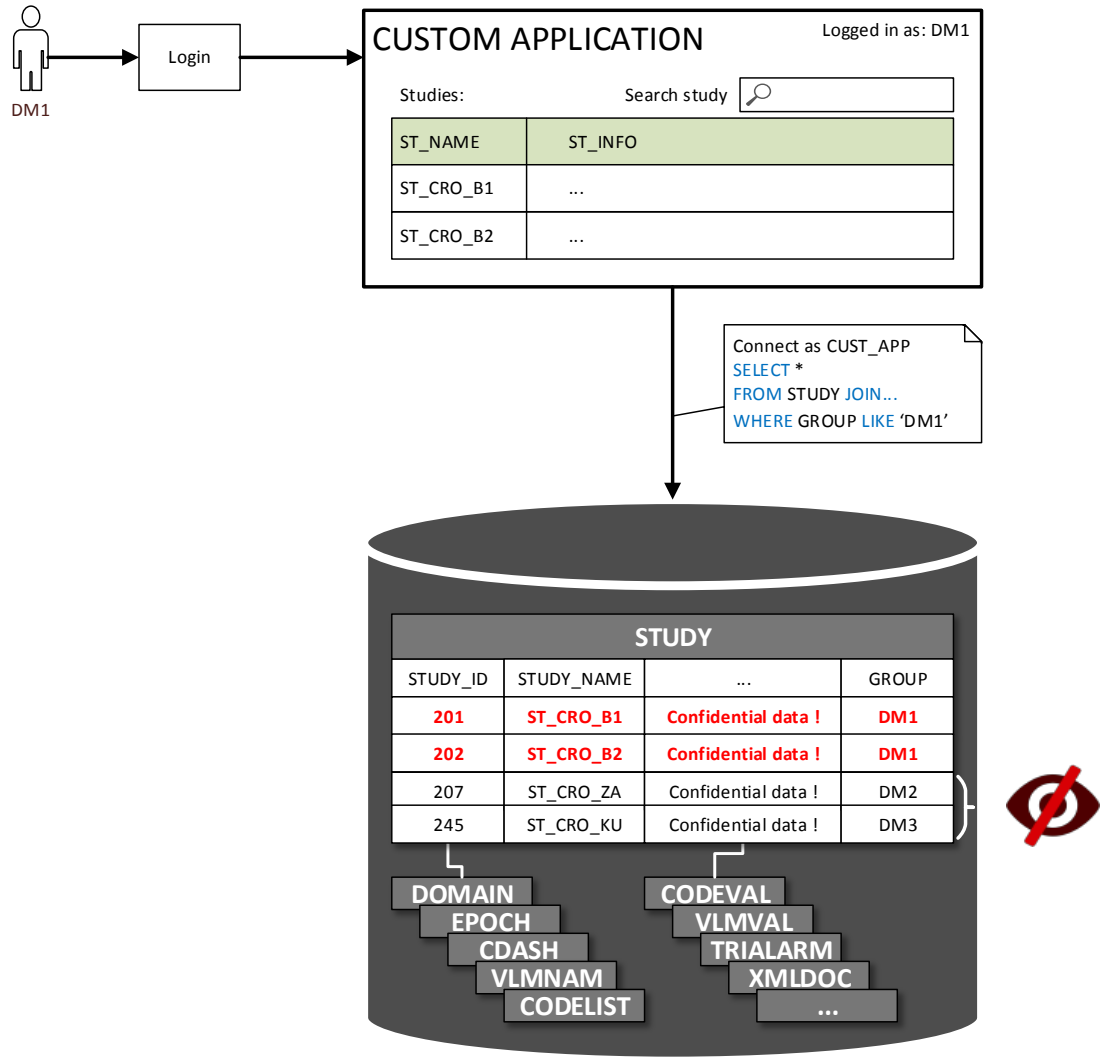
3.1. The problem

We will go through a practical simplified example that will motivate the reasons to use a Virtual Private Database. Suppose that in our database, we have 52 tables of data and metadata related to some clinical studies. Let's have a closer look at the single table that identifies these studies.



All tables are linked to the STUDY table. Ideally, in our example, the group DM1 should only access the data and metadata of the studies that they have created, which would be ST_CRO_B1 and ST_CRO_B2 and the related rows.

In order to do so, a program will be built to access the data in a controlled manner. Users will first authenticate in the application, which will handle the security and display the studies and related data that the user has access to.



This figure shows the user logging in. The application connects to the database to retrieve only the studies that the user has access to. In order to do so, the application connects using its own database user, in this case CUST_APP, which has full read/write access.

This solution introduces 2 major security issues:

1) The security is handled by the application, which can always be bypassed. The database itself is not protected. For example, suppose DM1 has a bit of knowledge in SQL and that the search box is not protected against SQL injection.
   In a normal case, the search box would be filled in with a study name such as **ST_CRO_ZA**, meaning that the user would like to retrieve all information about the study ST_CRO_ZA. Under the hood, a query that would look like this will be addressed to the database:

```
SELECT *
FROM STUDY
WHERE STUDY_NAME LIKE 'ST_CRO_ZA'
```

   The query means: from the table STUDY, filter all rows where the STUDY_NAME is equal to ST_CRO_ZA, and as a result, give back all columns (hence the star symbol).

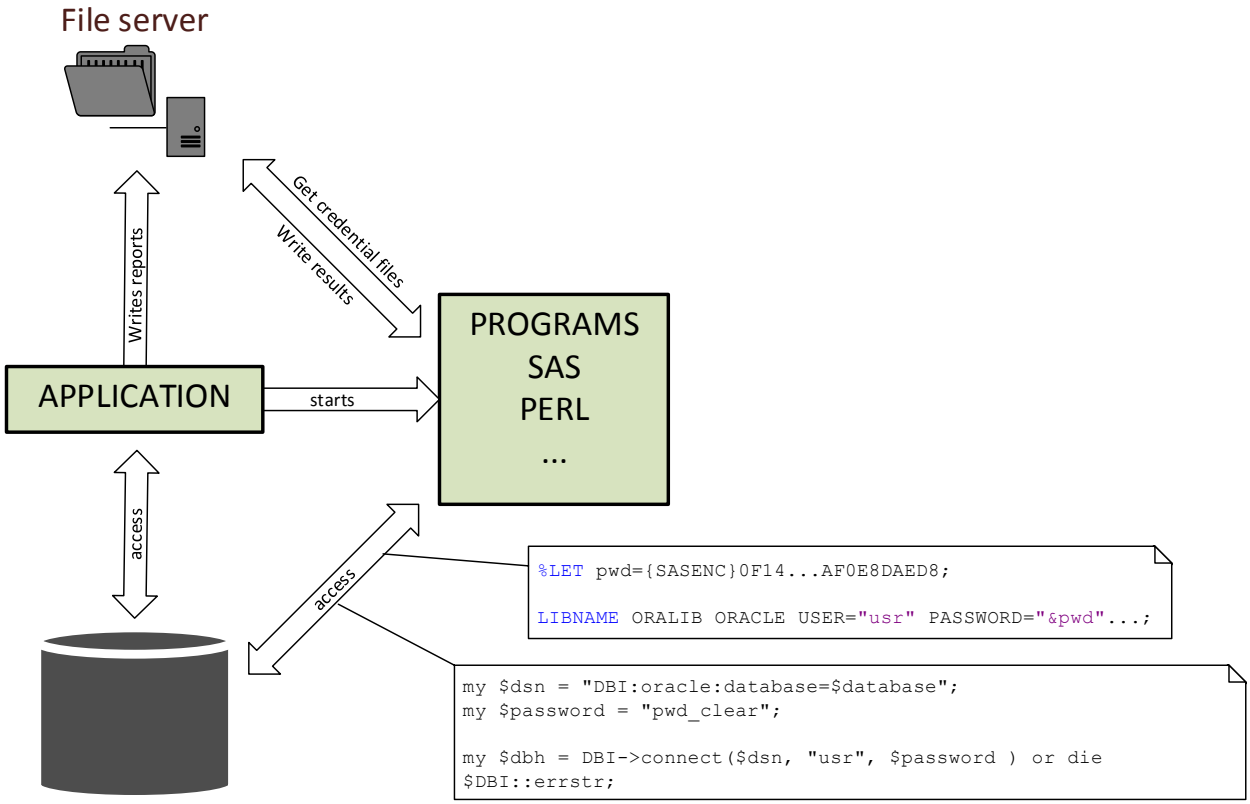
   Now what happens if, in the search box, DM1 would enter the string **' or 1=1--**? The request would become:

```
SELECT *
FROM STUDY
WHERE STUDY_NAME LIKE '' or 1=1--'
```

Then all studies are returned! Indeed, the request will search for studies that have an empty name OR 1=1 which will evaluate to true. The double *minus sign* is there just to comment the rest of the query (whose goal is to ignore the trailing *single quote sign to bypass a SQL error*).

And this only is the ABC of hacks, there are hundreds of security hacks that can be used, for which developers are often unaware of. The number of different technologies used nowadays to build application is only growing, directly correlated to security breaches. I would recommend to never trust the application's security as nothing is invulnerable.

2) What if a third party program is used, such as a SAS program or a script that needs to access the data? Applications nowadays integrate many pieces together, here is a real life example where the application not only connects to the database and writes reports to a fileserver, but also launches existing programs that themselves access the database (using credential files stored on a file server) to write back results to that fileserver:



A typical access plan for a program would be to call a macro that is stored in the fileserver, where the password is stored. The storage of that password is the weak point. Typically, with SAS, it would be PROC PWENCODEd. From the SAS documentation itself, a determined and knowledgeable person can decode the encoded passwords. Encryption can also be broken. The same goes for a PERL program. The end result is that DM1 knows the password of the database and will have access to unauthorized data.

This is why security should always be handled at the lowest level possible, here the database. Two different approaches are possible to enhance the security of our example.

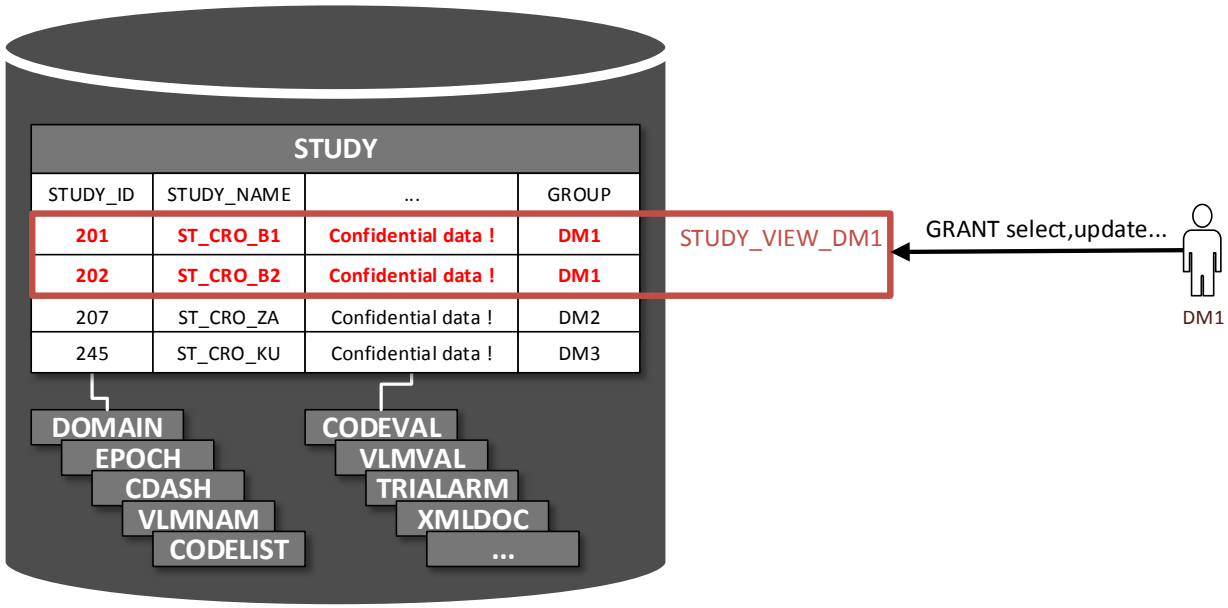
3.2. Views

As we have seen, views are a logical representation of one or more tables. Views could be generated for each table and role.

```
CREATE VIEW STUDY_VIEW_DM1 AS
  SELECT *
  FROM STUDY
  WHERE GROUP like 'DM1';

CREATE VIEW DOMAIN_VIEW_DM1 AS
  SELECT *
  FROM STUDY
    JOIN DOMAIN ON STUDY.STUDY_ID = DOMAIN.STUDY_ID
  WHERE STUDY.GROUP LIKE 'DM1';
```

Then we would need to create specific user access and grant an access for this user to the view.



Now if DM1 would be aware of the DM1 password to access the database, he would only still have access to the rows that he is allowed to see.

The drawback is however the number of views that we have to create.

3.3. Virtual Private Database

With Virtual Private Database, we would also need to create users and grant the access to the tables.

Virtual private database enables to enforce security, to a fine level of granularity, directly on tables, views, or synonyms. Because security policies are attached directly to tables, views, or synonyms and automatically applied whenever a user accesses data, there is no way to bypass security.

When a user directly or indirectly accesses a table, view, or synonym that is protected with a VPD policy, the server dynamically modifies the user's SQL statement. This can also affect highly powered accounts such as Database Administrators. The modification is based on a WHERE condition (known as a predicate) returned by a function which implements the security policy. The statement is modified dynamically, transparently to the user, using any condition which can be expressed in, or returned by a function. As VPD policies can be applied to SELECT, INSERT, UPDATE, INDEX, and DELETE statements, it allows a complete range of motion.

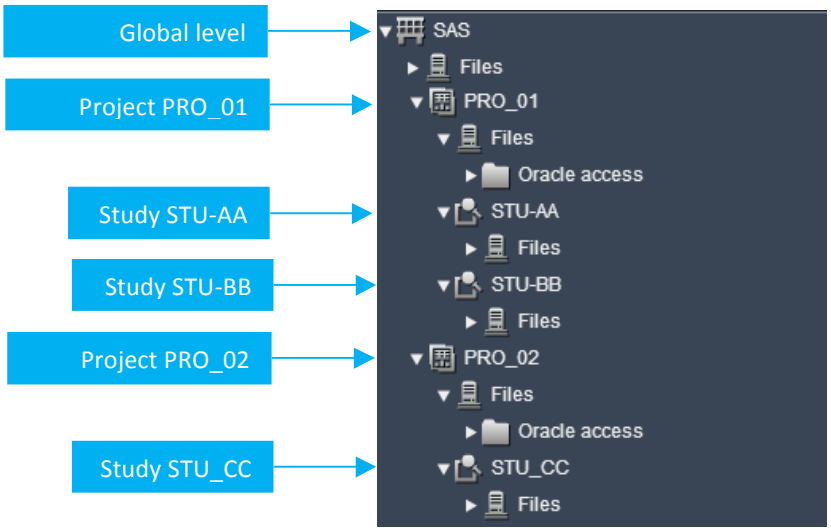
With Virtual Private Database, the data is protected at the source while views use a layered approach. This makes

it also easier to design and maintenance.

4. VPD WITH A FILE & PERMISSIONS SERVER

Here is an advanced example of how a VPD can be used in order to secure data along with a file server and a permissions server.
For the following case-study, we will use SAS Drug Development (SDD). Files are organized under 3 levels of hierarchy:

- Global level
- Projects
- Studies

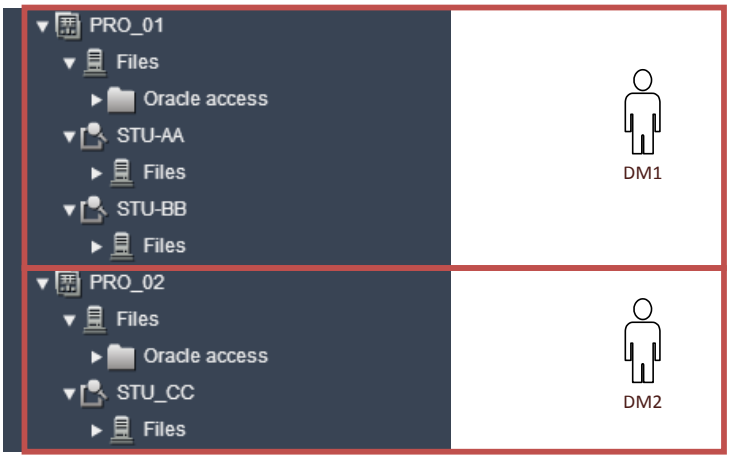


We would like to make sure:

1. That users who have access to PRO_01 cannot see data that are in another project
2. Secure the database

To ensure the first point, we will restrict the access to folders:

- First group of Data Managers (DM1) will only be able to access PRO_01
- Second group of Data Managers (DM2) will only be able to access PRO_02



Let's suppose that we have a SAS program called `create-report.sas` that needs to access the database and writes back a report in the study files area.

Notice the "Oracle access" folder for each project-level level files. We can store there a file that will initiate the connection to the database. The following example shows the SAS code saved under `PRO_1>Files>Oracle access`:

```
%LET usr=PRO_01;
%LET pwd={SASENC}2FA9941B0A836A9D5190A84535C02778;

/*
 * Set libname to oracle with the encrypted password
 */
LIBNAME ORALIB ORACLE USER="&usr" PASSWORD="&pwd" PATH="srv01" SCHEMA="app"
oracle_73=no;


OPTIONS NOCHARCODE;

/*
 * Check if the libname has been set correctly
 */
%MACRO CHECKLIB;
%IF &SYSLIBRC. NE 0 %THEN %DO;
   %ABORT CANCEL;
%END;
%MEND CHECKLIB;

%CHECKLIB;
```

This SAS access program will be the same under `PRO_2>Files>Oracle access`, except the credentials:

```
%LET usr=PRO_02;
%LET pwd={SASENC}4BC9941B390B5C5755357C7616AF6723;
```

This way, we ensure that DM1 will not have the same credentials as DM2. If DM1 would like to sneak in DM2's data, he could not simply change the `usr` variable as the passwords are not the same. Also, the permissions policy set on the file server prevents the DM2's access file to be seen by DM1.

In this example, we have secured the database using VPD.
First, we have to create a function that will be called upon access to tables.

```
CREATE OR REPLACE FUNCTION auth_study_function(
  schema_var IN VARCHAR2,
  table_var IN VARCHAR2
 )
 RETURN VARCHAR2
 IS
  return_val VARCHAR2 (400);
 BEGIN
    if UPPER(SYS_CONTEXT('USERENV', 'SESSION_USER')) = 'SUPERUSER'
    then
        return_val := '';
    else
        return_val := 'AUTH_GROUP = SYS_CONTEXT(''USERENV'', ''SESSION_USER'')';
    end if;
    RETURN return_val;
 END auth_ss_function;
/
```

What this function returns is a statement that will be added to the WHERE clause. When this function will be called, if the user is SUPERUSER, then nothing will be added to the WHERE clause. This handles the case where you would want some users to have a full access to the data. However if the user is, for example PRO_01, then the WHERE clause statement becomes: "AUTH_GROUP = PRO_01" which will filter out all the values where the column

AUTH_GROUP is not PRO_01.

We have to associate this function to a schema, tables or even actions.

```
BEGIN
       DBMS_RLS.ADD_POLICY (
              object_schema       => 'APP',
              object_name         => 'APP_STUDY',
              policy_name         => 'APP_STUDY_policy',
              function_schema     => 'app',
              policy_function     => 'auth_study_function',
              statement_types     => 'select'
       );
END;
/
```

Then finally grant the accesses.

```
GRANT SELECT ON APP.IDS_STUDY TO SUPERUSER;
GRANT SELECT ON APP.IDS_STUDY TO PRO_01;
GRANT SELECT ON APP.IDS_STUDY TO PRO_02;
```

This way, even if DM1 was to decrypt the password that is stored in `PRO_1>Files>Oracle access`, he could not use a third party program to explore unauthorized data. Indeed, even if would use a third party program, the dynamic WHERE clause would still be added by the Oracle kernel, filtering all results.

The obvious advantage over views is that it is easier to maintain. When a new project is created, only the GRANT clause needs to be added. If our security was based on views, we would have to create X new views and grant access to these X views. VPD however just requires a column to be added to the data model where a VPD filtering needs to be applied.

5. CONCLUSION

In conclusion, we've seen that in order to provide a good level of security you should protect your data at the lowest level possible – which is where the data is. Indeed, you should not trust your application's security. It is possible to add another layer of security by implementing views, however this is still an extra layer – leaving a possible security gap on the base tables. Oracle offers Virtual Private Database, which are easy to maintain, very flexible and offer a kernel protection.

REFERENCES

- SAS online documentation
- Oracle online documentation

SOFTWARE USED

- Oracle 11G R2
- SAS 9.3
- SAS Drug Development (SDD) 4.2

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Sébastien Roland
Business & Decision Life Sciences
Sint-Lambertusstraat 141 Rue Saint-Lambert
1200 Brussels
Email: sebastien.roland@businessdecision.com

Web: www.businessdecision-lifesciences.com

Brand and product names are trademarks of their respective companies.