Row level security using the Bell-Lapadula Model in Relational Databases

Trent Millar

February 25, 2013

Table of Contents

Executive Summary 3
Bell-Lapadula Basics 4
The BLP Database 6
    BLP Tables 6
    Audit Tables 6
    Agency Tables 6
ER Data Model 7
Flows in a BLP security model 12
SQL DDL statements - BLPDB creation 18
    Create 18
    Tables – BLP model 20
    Data Population – BLP model 21
    Tables – Agency model 23
    Data Population – Agency model 24
    Views – Agency model 25
    Tables – Audit Model 27
    Stored Procedures 27
    Triggers – Agency Model 28
Data – the BLPDB example 31
BLP examples in the BLPDB database 32
    Scenario, authentication 33
    Scenario, read operation 34
    Scenario – write operations - inserts 35
    Scenario – write operations – update 37
    Scenario – Audit records 38
Conclusion 38

Executive Summary

In a business, there are shutters over windows, locked doors, and alarm systems at the ready. In computer security, there are also multiple levels that need to ensure a network and its systems are secure, and most companies put every effort to achieve this. One overlooked area of security is the actual data the company maintains; it is rare that protection occurs at the data level. Maybe some database access controls are in place but beyond that an operator can typically view all the data within all the tables. At best the operator may only have access to the tables within a database that are necessary to do their job function, although a good first step this still does not solve the problem. The problem is data at the row level may contain different levels of sensitivity. Some rows may be unrestricted and are public knowledge, but some data may be so sensitive it could cause serious financial loss or even death if seen by the wrong people.

This concept is called multilevel security (MLS) and one long living implementation of this concept has been the Bell-Lapadula (BLP) security model. BLP is a secrecy based model that fits well with the relational database concept. Since most relational database management systems (RDBMS) revolve around reading and writing data, BLP adds a layer of security on top of this that is not found in most RDMS including Microsoft SQL Server, MySQL, Postgres, DB2, and countless other products.

The aim of this paper is to focus on the concept of how security can be applied at the row level.

Bell-Lapadula Basics

The BLP model was developed in the 1970s as a formal mandatory access control model. The model was developed specific to a military axiom and is focused on sensitivity of information between subjects and objects. A subject is an active entity that requests to read or write to an object. An object is a passive entity that represents a record within the database.

The model is focused on a hierarchical formulation of security levels or "classifications". Classifications are hierarchical in that; a Subject will inherit the properties of the classifications below it. The following classifications are common in many military and governments around the world:

TOP SECRET > CLASSIFIED > RESTRICTED > UNCLASSIFIED

To make the BLP security model more robust is to include a set of "compartments" which is similar to a category that groups like data. Compartments are specific to the organization but are typically named by project or military theater:

PROJECT X, DESERT SHIELD, OPERATION SMOOTH

BLP at its core applies both a "read down" and "write up" concept between subjects and objects. Technically these two concepts were defined by BLP as:

1. **Simple security property (ss-property),** subjects can only read an object if its classification is **equal to or less than the subject's classification**.
2. *** Property (star-property),** the subject is only allowed to write to an object, if the **subject classification is less than or equal to the object's** classification. This may not be expected relative to some security models in common use. This model prevents a highly classified subject from exposing objects to lower classifications.

The concept of subject and object applied to the data model can be interpreted as follows.

1. **Reading rows**; when the “user” (subject) selects “row(s)” (object(s)) from the database, the user must dominate the row(s) classification. All un-dominated row(s) will not be returned to the subject, and only rows that the user has dominance over will be included in the result set.
2. **Writing rows (update)**; when the user (subject) updates existing rows (objects) in the database an error may occur the row **does not** dominate the user. Since a user must only write to rows that dominate him, the database will “roll back” the atomic operation to the state prior to the operation.
3. **Writing rows (insert)**; the BLP database will handle new object creation in one of two ways. If the row has no parent, a user will apply their default classification level to the new row. If the row has a parent with a classification, then this classification is considered a hierarchically enforced classification, and the new row will use this level. In the latter case, if the user dominates the classification then the write will fail, and the database will “roll back” to the state before this operation started.

This is meant to be a quick introduction to both BLP and how the example BLP database (BLPDB) behaves. The next section will discuss the BLP data model and an explanation of how it works, then later sections will cover the working operation of the database using SQL statements to invoke telling scenarios.

BLPDB - The BLP Database

The BLPDB is developed in SQL Server 2008 but uses features common to most RDBMSs. The data model has two central cores; the first is the BLP security management tables and database objects, the second are the supporting tables to provide a real-world example of BLP within a RDBMS. The BLPDB contains eight tables within three logical sections, laid out in the following is the relational schema:

BLP Tables

USER (**USER_ID**, USER_NAME, USER_PASSWORD, *CLASSIFICATION_ID*)

CLASSIFICATION (**CLASSIFICATION_ID**, CLASSIFICATION_NAME)

COMPARTMENT (**COMPARTMENT_ID**, COMPARTMENT_NAME)

USER_COMPARTMENT_CLASSIFICATION (**COMPARTMENT_ID**, **CLASSIFICATION_ID**, **USER_ID**)

Audit Tables

AUDIT (AUDIT_TABLE_NAME, AUDIT _OPERATION, AUDIT _USER, AUDIT_TIME)

Agency Tables

SORTIE (**SORTIE_ID**, SORTIE_NAME, SORTIE_POPULATION, *CLASSIFICATION_ID. COMPARTMENT_ID*)

INFORMANT (**INFORMANT_ID**, INFORMANT_FIRSTNAME, INFORMANT_LASTNAME, INFORMANT_GENDER, INFORMANT_BIRTHDATE, C*LASSIFICATION_ID. COMPARTMENT_ID*)

SECRET (**SECRET_ID**, *INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID, COMPARTMENT_ID,* SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)

The “BLP tables” are specifically for driving the underlying Bell-Lapadula security model. The purpose of these tables is to define the organization’s classification levels,

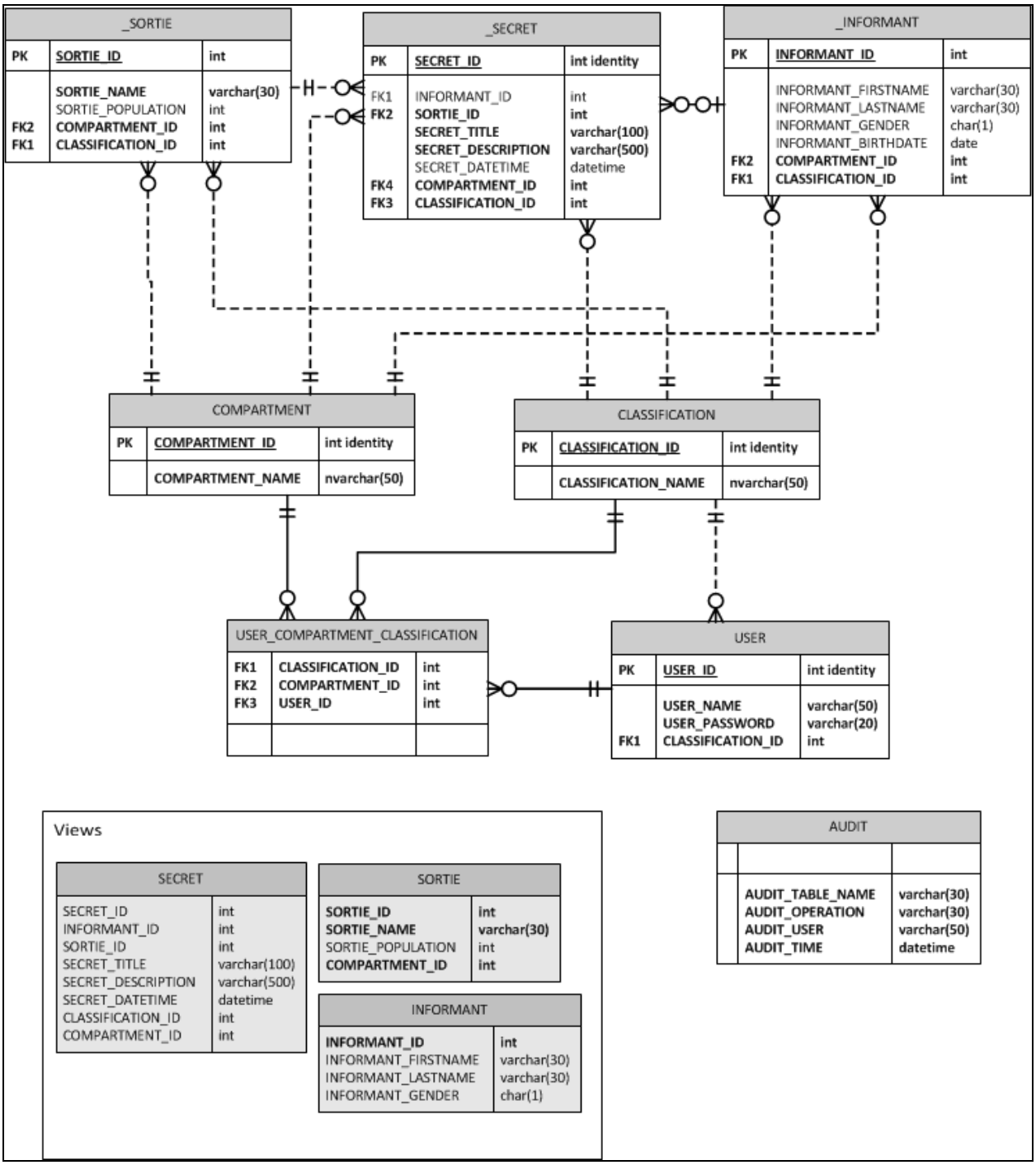
compartments (categories), and the users (subjects). The compartment id and classification id are used on the rows (objects) in the “Agency tables” to define access.

The “Audit tables” contain a single table called; Audit. This table is used for inserts that represent; Who, What, Where, and When. If a user reads or writes to an object, this is recorded in the table. It is meant for simple, primitive audit purposes and is not very robust.

ER Data Model

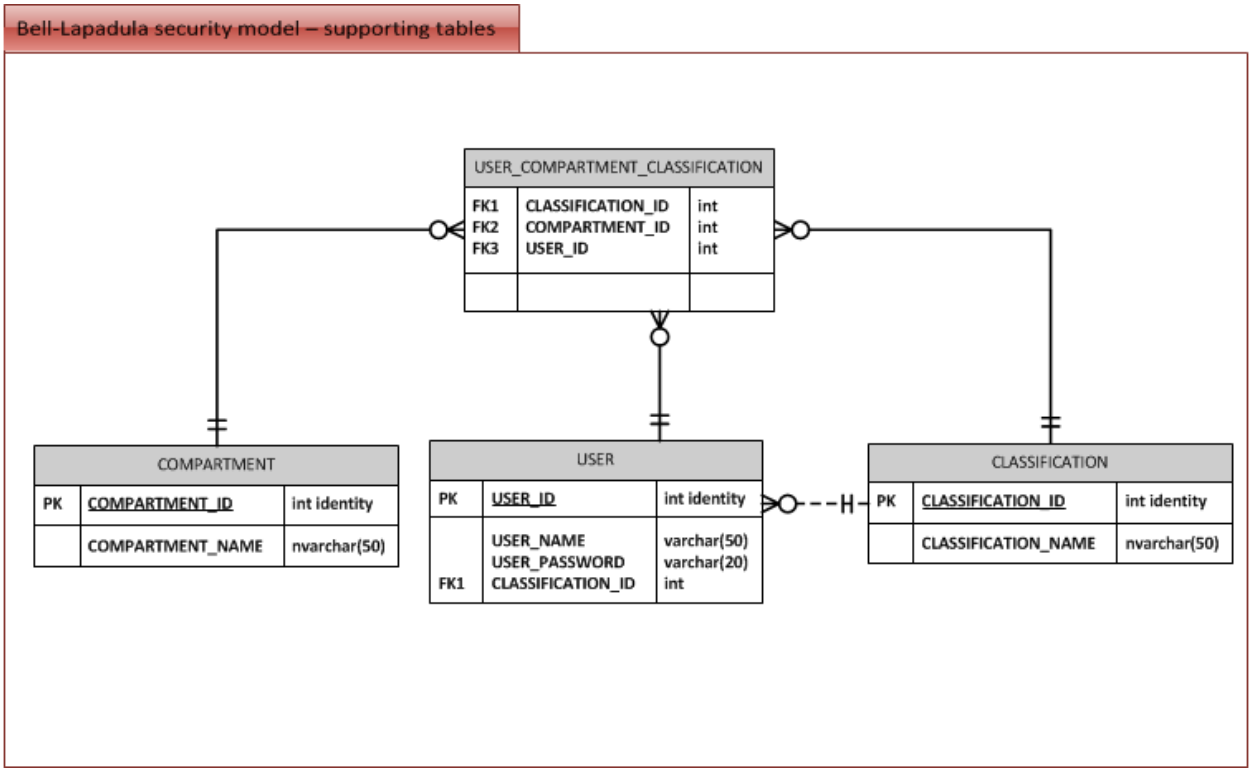
The pattern used to enforce BLP security is using the built-in RDBMSs’ security features to deny direct access to the business tables. Business tables are only accessed using views; this allows the BLP model present the data to the user based on their privileges. In the example data model, the tables; _SORTIE, _INFORMANT, and _SECRET are not accessed directly but instead through the views; SORTIE, INFORMANT, and SECRET.

BLP tables are used just to enforce the security model and should only be managed by high level administrators of the system. The Data Definition Language (DDL) included in the appendix prevents access to these tables using ‘DENY’ statements. These tables are pre-populated with data to demonstrate the BLP model in action.



Full ER model of BLPDB

The full ER model has three logical components that were previously described in the relational schema; Agency tables, BLP tables, and Audit tables. The following are the BLP tables;

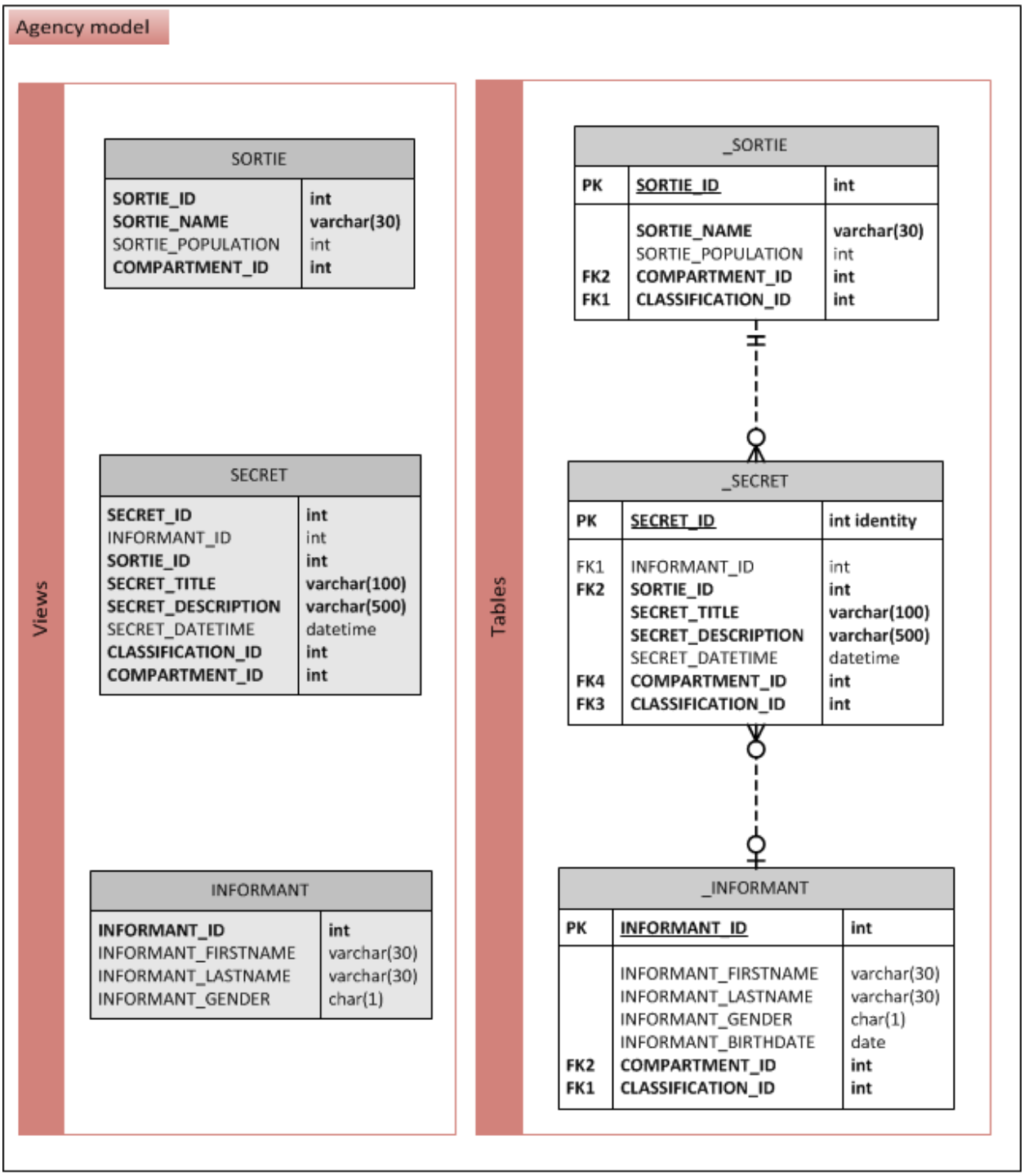


BLP Model

The table USER_COMPARTMENT_CLASSIFICATION is a weak type since it's identified by the other three strong type tables. The purposes of the BLP tables are:

- Define the compartments the organization can use.
- Define all allowable classifications in a hierarchical format.
- Define all the users within the system.
- Lastly, combine the above to define the security model of the entire system.

The BLP tables aren't exactly specific to the organization, however the "Agency tables" are.



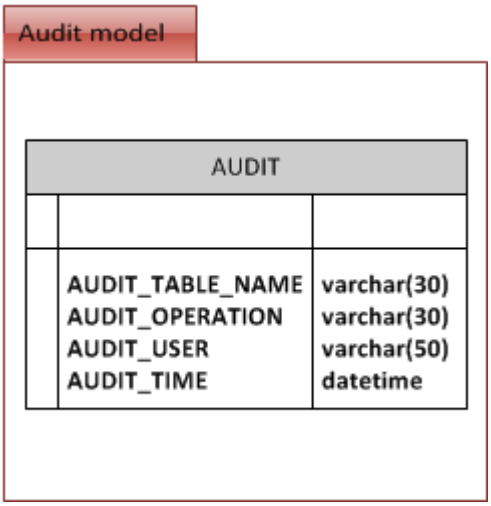
Agency Model

In the example BLPDB database, there are three agency tables; SORTIE, INFORMANT, and SECRET. These tables are used to illustrate and show off the capabilities of the BLP tables. The business descriptions of these three tables are:

- **_SORTIE**, represents a mission the agency is involved in; each sortie is a fictional location that the agency collects intelligence on over time.
- **_INFORMANT**, a person who feeds the agency secret intelligence related to the agency's sorties.
- **_SECRET**, over time the agency compiles secrets that it can use to help the success of its sorties. Secrets typically have an informant but not always, but they will always have a sortie.

All agency tables contain both a compartment and classification; these two attributes allow the security model to decide what operation, if any can be applied. All agency tables prevent direct access using the RDBMS security features; the tables are all fronted by views which wrap the security and audit features.

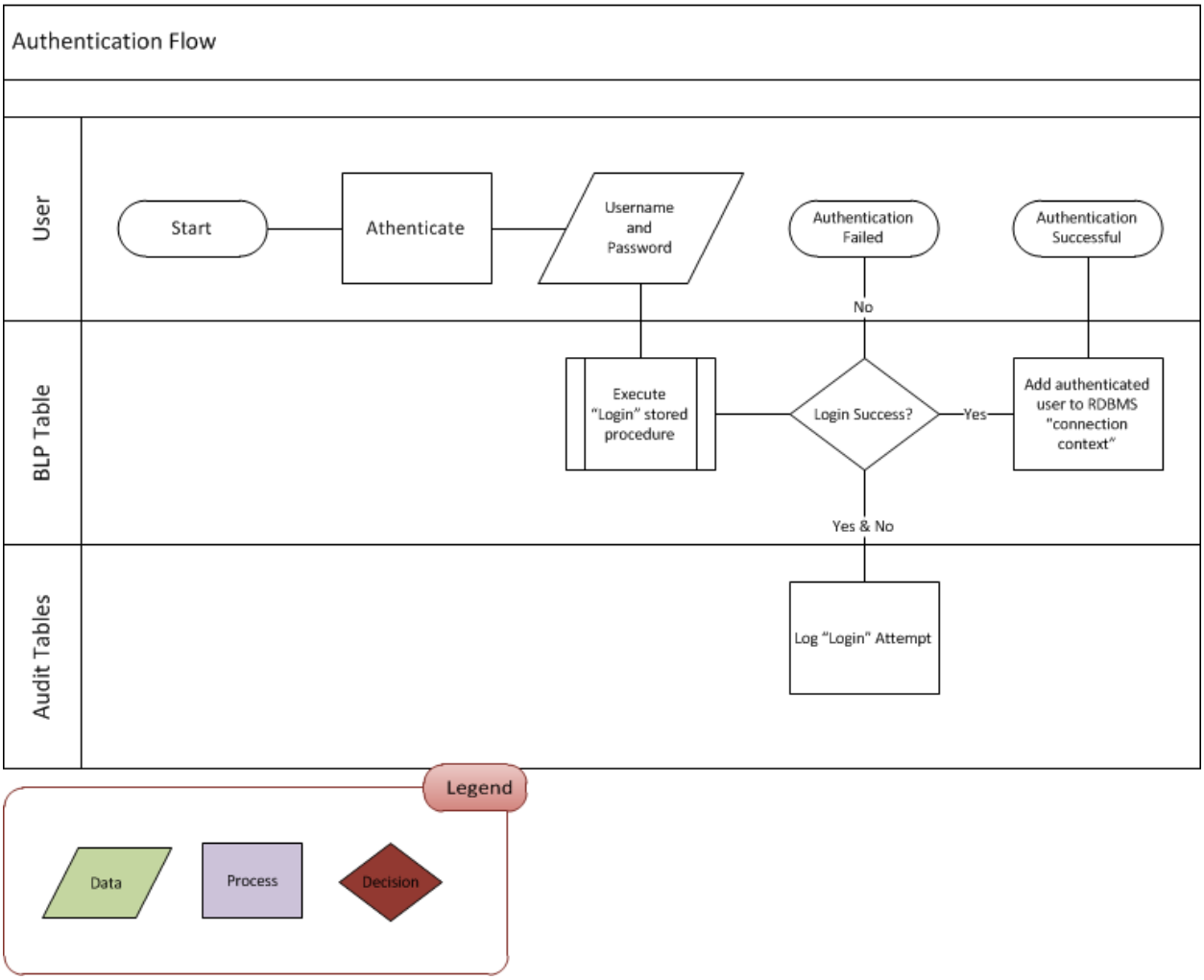
The "Audit tables" contains just a single table. The audit table provides primitive auditing by recording the table affected, the user, the time, and if that operation passed or failed. This audit strategy can easily be extended as required by the organization.



Audit Model

Flows in a BLP security model

This section will focus on describing the detailed BLP steps involved when data is; read, inserted, and updated in the database. The flows in general, and the flows specific to the example BLPDB database will be covered.

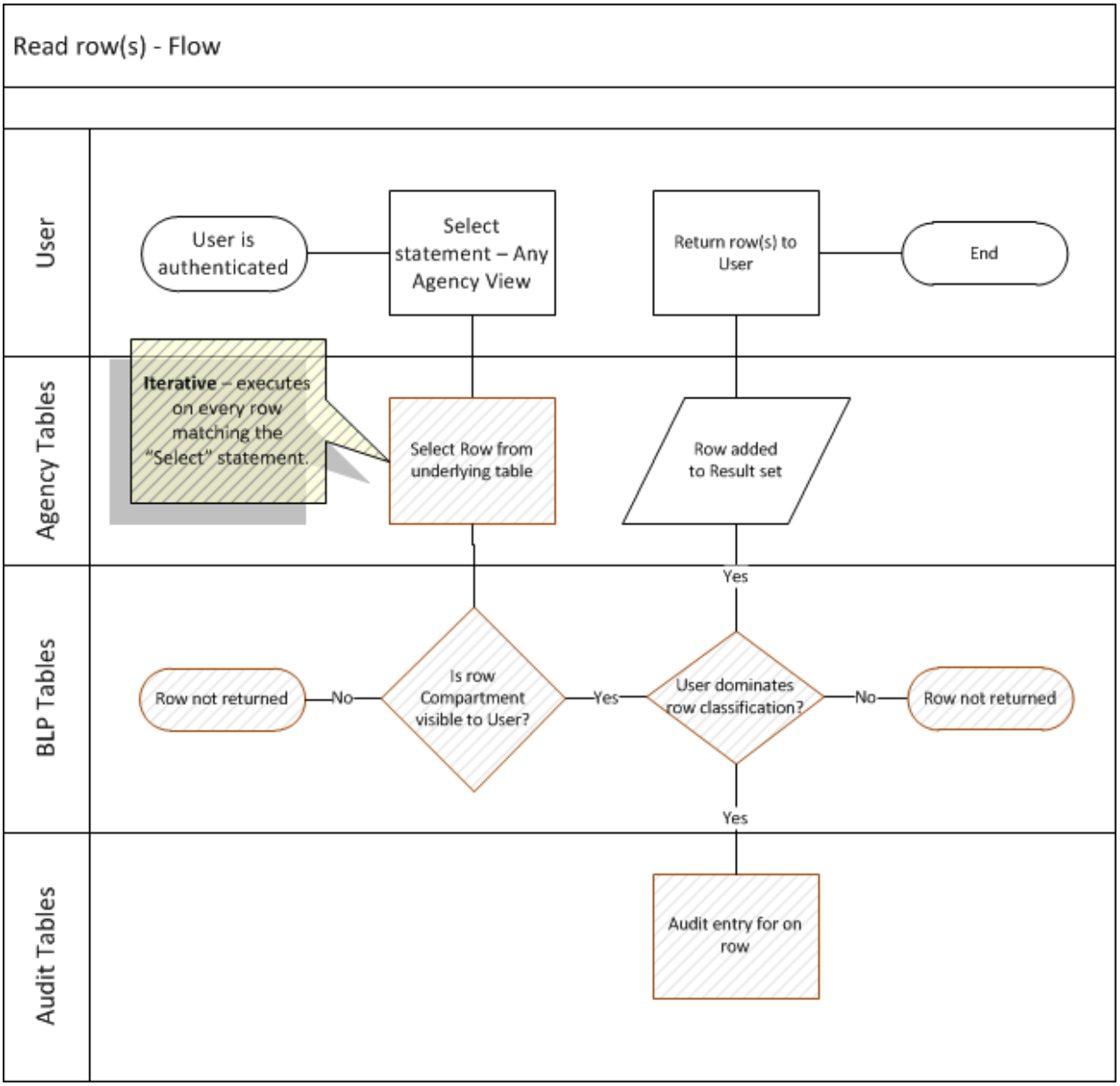


Authentication flow

The "authentication flow" is the first process that occurs when a user wants to access rows in the database. The RDBMS has the concept of users that can be added to the running context of the database. What this means is a user will communicate with the database using the same connection, this connection will transparently store the user in the background, allowing

the security model to make assertions at any time. This is the foundation of the BLP security model. The steps involved in the “authentication flow” are:

1. The first time a user accesses the BLPDB they need to authenticate themselves. This only has to be done once or until the database is restarted on the database server.
2. In the BLPDB, there is a stored procedure called “Login” that expects both the user’s username and password.
3. If the Login details are valid then the BLPDB adds this user to the user’s RDBMS connection session. If the Login details are invalid, then the user will receive an error and will not be able to access the BLPDB.
4. Regardless of what happens in step three, an audit entry, is added to the audit table specifying what happened.



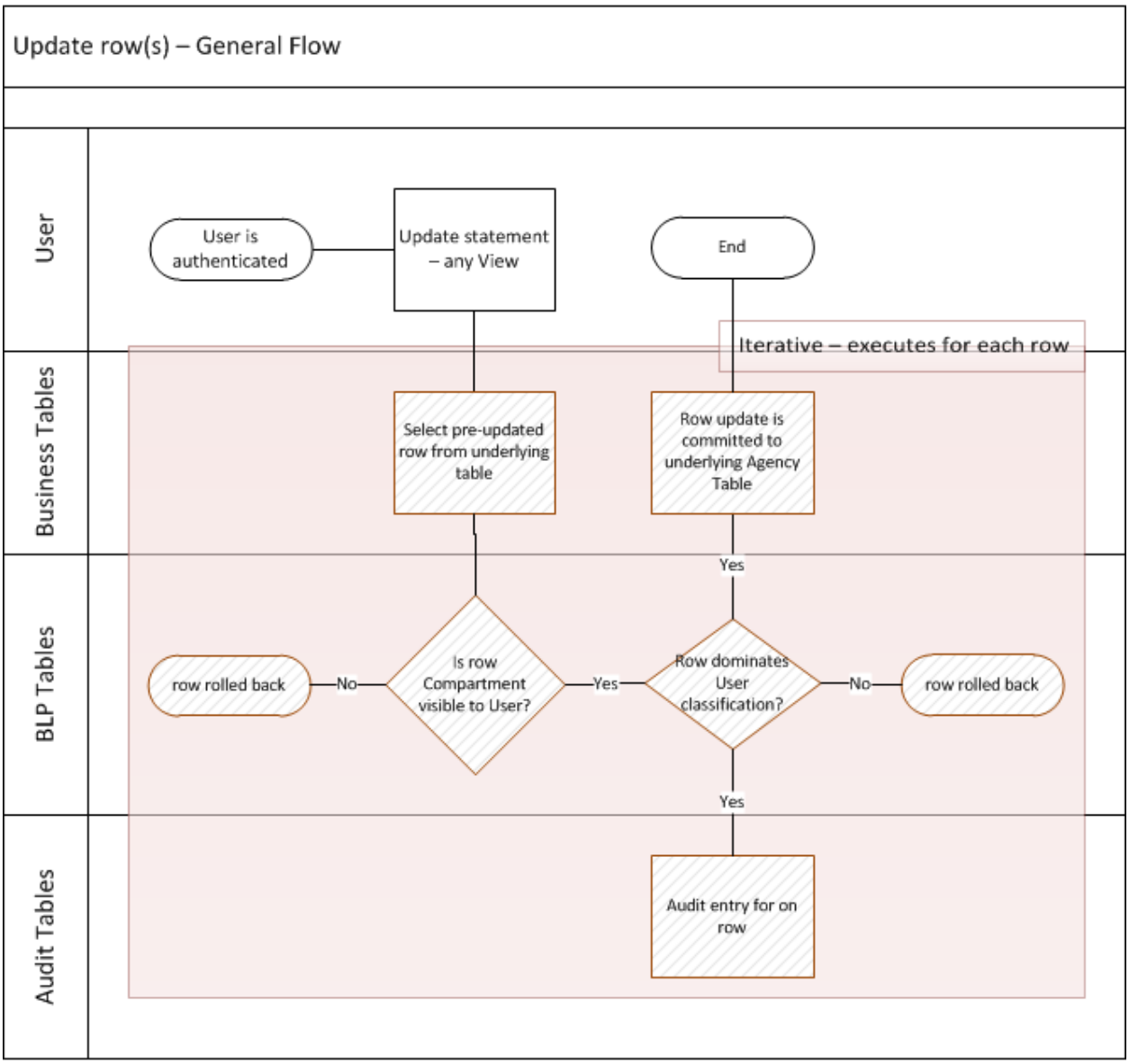
Read flow

The "read flow" is performed when a user attempts to select one or more rows from a protected table, in this case an Agency table. The approach used in the BLPDB is that all agency tables are fronted by a View; to a user this is no different from accessing a table. The steps performed when an authenticated user executes a "Select" statement against an agency view are as follows:

1. The user performs a select against a View, for example;

```
SELECT * FROM [agency].SECRET
```

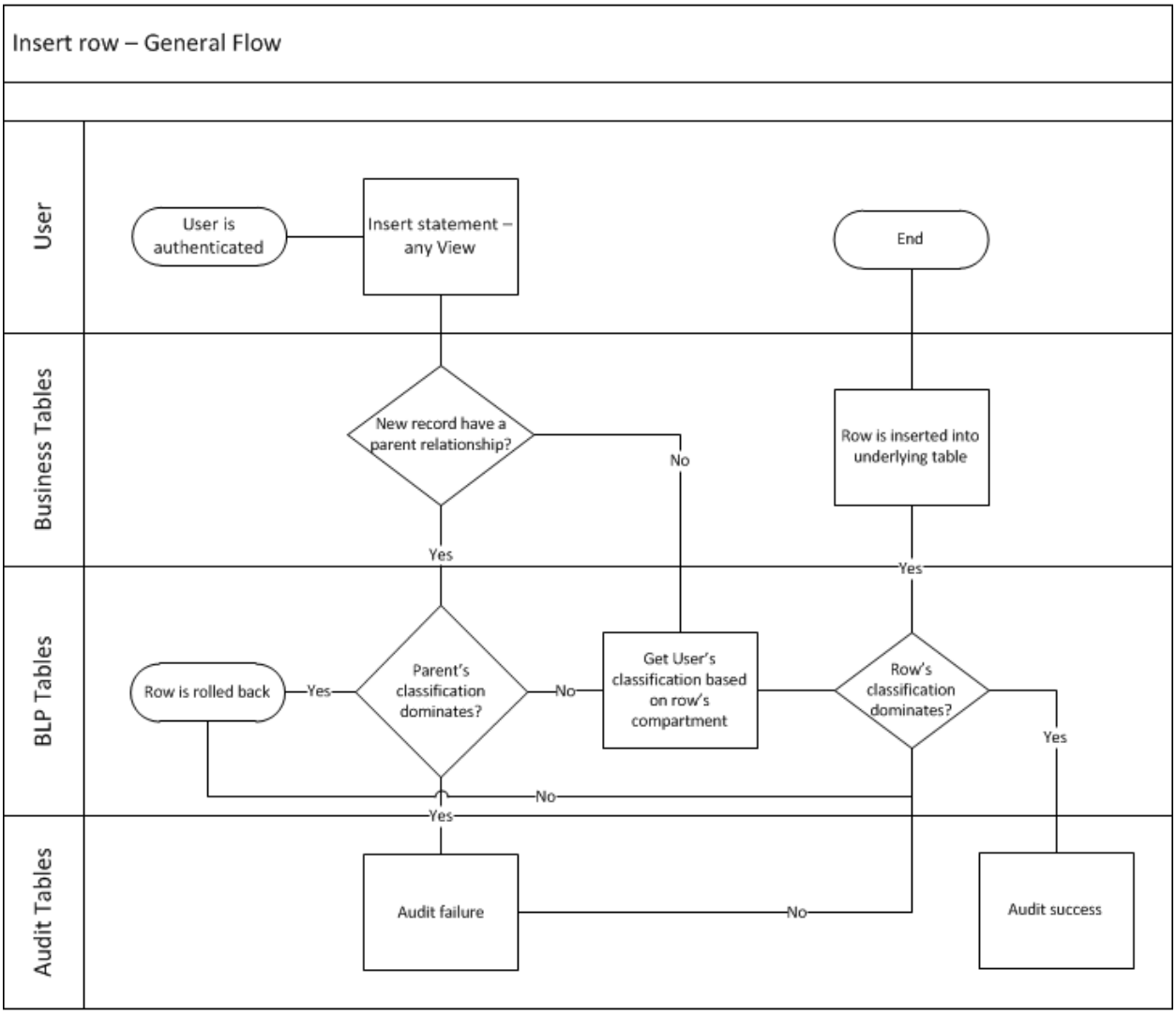
2. The BLPDB's `_SECRET` table contains both a compartment and classification; each row returned in the select statement will perform the following check.
    a. Does the user belong to the compartment? This is checked by querying the USER_COMPARTMENT_CLASSIFICATION table.
    b. Does the user dominate the classification of the row? Again this is in the USER_COMPARTMENT_CLASSIFICATION table.
3. If the previous step was **successful** on both checks, the row will be added to the result set and returned to the user. If **unsuccessful** then the row will not be added to the result set.



Update flow

The "update flow", like the "read flow" is applied to a view. Most RDBMS's allow views to be updated using a **Trigger,** and this is how the BLPDB allows the updates to flow down to the underlying agency tables. Updates in the BLP security model perform similar steps that the "read flow" executed. The real difference between a read and an update in the BLP model is; reads are user dominant, where the user can "read-down", while the updates are row dominant where the user can "write-up". The following will help break down the update flow:

1. An “update” statement is applied to a view, which then attempts to modify the underlying table rows matching the statement criteria.
2. For each row, the trigger will look at the row data prior to this update. Just like the read, if the user’s compartment is not the same as the rows then the update rolls back that row.
3. The domination check is next, unlike the read where a user must dominate the row, a write the row must dominate the user - the user must be of lower or equal classification.



Insert flow

"Insert flow" is similar to the update where the row dominates the user. It is also similar in that it uses a **trigger** to control the insert. In the BLPDB database, there is another check, which is to see if the row has a parent to enforce a classification down. The following are the steps that occur:

1. The inserted row will first check if there is a parent relationship and if so that the inserted row dominates the parent. A child row must not have less classification than what its parent has.
2. The next check will compare the user classification for of the inserted row compartment; if this classification is dominated by the row then the insert is successful.
3. Whether successful or not the operation is audited in the audit table.

SQL DDL statements - BLPDB creation

Before covering the scenarios in a later section; the database must be created including all the objects within it. The following sub-sections will cover each script and step that create the final BLPDB database:

Create

The following is the first script executed against a MS SQL Server 2008 database.

```
USE [master]
GO

SET NOCOUNT ON
GO

EXEC dbo.sp_addmessage
      @msgnum = 50001
      ,@severity = 16
      ,@msgtext = 'INVALID LOGIN'
      ,@lang = NULL
      ,@with_log = false
      ,@replace = 'replace'
```

```
GO

EXEC dbo.sp_addmessage
      @msgnum = 50002
      ,@severity = 16
      ,@msgtext = 'WRITE UP ERROR'
      ,@lang = NULL
      ,@with_log = false
      ,@replace = 'replace'
GO

-- Drop the database if it already exists
IF EXISTS (SELECT name FROM sys.databases WHERE name = N'BLPDB')
      ALTER DATABASE BLPDB SET SINGLE_USER WITH ROLLBACK IMMEDIATE
GO

IF EXISTS (SELECT name FROM sys.databases WHERE name = N'BLPDB')
      DROP DATABASE BLPDB
GO

CREATE DATABASE BLPDB
GO

USE BLPDB
GO

--CREATE '[agency]' SCHEMA
IF EXISTS (SELECT schema_name FROM information_schema.schemata WHERE
schema_name = '[agency]')
      EXEC('DROP SCHEMA [agency]')
GO

EXEC('CREATE SCHEMA [agency] AUTHORIZATION dbo')
GO

--SETUP SECURITY
PRINT 'Creating BLRole';
IF USER_ID(N'BLRole') IS NOT NULL
      DROP ROLE [BLRole]
CREATE ROLE [BLRole]
GO

PRINT 'Granting Execute on agency schema to BLRole';
GRANT EXECUTE ON SCHEMA :: [agency] TO [BLRole]
GO

PRINT 'Granting Select on [agency] schema to BLRole';
GRANT SELECT ON SCHEMA :: [agency] TO [BLRole]
```

This script performs the following:

1. Adds two “Messages” to the underlying RDBMS, these are used as messages when errors are returned to users.

2. The physical database is dropped and then created – this provides a clean slate database for the remaining scripts.
3. Many RDBMS's use schemas to encapsulate database objects, this provides a layer of security and makes it easy for the BLPDB to restrict access to all tables within the schema.
4. The role; "BLRole" is created – this is the role all users will run under while accessing the BLPDB database. (Similar concept to a group in Windows security)
5. The new role is granted access to select and execute all objects under the new schema – this is a quick way to give minimal access to the objects used.

Tables – BLP model

The next script creates the "BLP" tables and constraints between each other.

```
USE BLPDB
GO

SET NOCOUNT ON
GO

--CREATE TABLES - Security Model
CREATE TABLE [agency].[USER_COMPARTMENT_CLASSIFICATION](
      [CLASSIFICATION_ID] [int] NOT NULL,
      [COMPARTMENT_ID] [int] NOT NULL,
      [USER_ID] [int] NOT NULL,
      CONSTRAINT [UQ_COMP_CLASS] UNIQUE ([COMPARTMENT_ID],[USER_ID])
)
GO

CREATE TABLE [agency].[CLASSIFICATION](
      [CLASSIFICATION_ID] [int] IDENTITY(1,1) NOT NULL,
      [CLASSIFICATION_NAME] [nvarchar](50) NOT NULL
)
GO

CREATE TABLE [agency].[COMPARTMENT](
      [COMPARTMENT_ID] [int] IDENTITY(1,1) NOT NULL,
      [COMPARTMENT_NAME] [nvarchar](50) NOT NULL
)
GO

CREATE TABLE [agency].[USER](
```

```
      [USER_ID] [int] IDENTITY(1,1) NOT NULL,
      [USER_NAME] VARCHAR(50) NOT NULL,
      [USER_PASSWORD] VARCHAR(20) NOT NULL,
      [CLASSIFICATION_ID] INT NOT NULL DEFAULT(1)
)
GO

-- CREATE CONSTARINTS
ALTER TABLE [agency].[CLASSIFICATION] ADD PRIMARY KEY
CLUSTERED([CLASSIFICATION_ID] ASC)
GO

ALTER TABLE [agency].[COMPARTMENT] ADD PRIMARY KEY CLUSTERED
([COMPARTMENT_ID] ASC)
GO

ALTER TABLE [agency].[USER] ADD PRIMARY KEY CLUSTERED ([USER_ID] ASC )
GO

ALTER TABLE [agency].[USER] ADD FOREIGN KEY([CLASSIFICATION_ID])
      REFERENCES [agency].[CLASSIFICATION] ([CLASSIFICATION_ID])
GO

ALTER TABLE [agency].[USER_COMPARTMENT_CLASSIFICATION] ADD FOREIGN
KEY([CLASSIFICATION_ID])
      REFERENCES [agency].[CLASSIFICATION] ([CLASSIFICATION_ID])
GO

ALTER TABLE [agency].[USER_COMPARTMENT_CLASSIFICATION] ADD FOREIGN
KEY([COMPARTMENT_ID])
      REFERENCES [agency].[COMPARTMENT] ([COMPARTMENT_ID])
GO

ALTER TABLE [agency].[USER_COMPARTMENT_CLASSIFICATION] ADD FOREIGN
KEY([USER_ID])
            REFERENCES [agency].[USER] ([USER_ID])
GO
```

Data Population – BLP model

These are basic records used to stage the database in a usable form for testing and demonstrating scenarios.

```
USE BLPDB
GO

--POPULATE BELL-LAPADULA MODEL
SET IDENTITY_INSERT agency.[COMPARTMENT] ON
GO

--insert COMPARTMENT table
      INSERT INTO [agency].[COMPARTMENT] ([COMPARTMENT_ID],
[COMPARTMENT_NAME]) VALUES (1, 'None');
```

```
      INSERT INTO [agency].[COMPARTMENT] ([COMPARTMENT_ID],
[COMPARTMENT_NAME]) VALUES (2, 'Desert Shield');
      INSERT INTO [agency].[COMPARTMENT] ([COMPARTMENT_ID],
[COMPARTMENT_NAME]) VALUES (3, 'Operation SMOOTH');
      INSERT INTO [agency].[COMPARTMENT] ([COMPARTMENT_ID],
[COMPARTMENT_NAME]) VALUES (4, 'Project X');
GO

SET IDENTITY_INSERT agency.[COMPARTMENT] OFF
GO

SET IDENTITY_INSERT agency.[CLASSIFICATION] ON
GO

--insert CLASSIFICATION table
      INSERT INTO [agency].[CLASSIFICATION] ([CLASSIFICATION_ID],
[CLASSIFICATION_NAME]) VALUES (1, 'Unclassified'); -- WIDE OPEN
      INSERT INTO [agency].[CLASSIFICATION] ([CLASSIFICATION_ID],
[CLASSIFICATION_NAME]) VALUES (2, 'Restricted');
      INSERT INTO [agency].[CLASSIFICATION] ([CLASSIFICATION_ID],
[CLASSIFICATION_NAME]) VALUES (3, 'Classified');
      INSERT INTO [agency].[CLASSIFICATION] ([CLASSIFICATION_ID],
[CLASSIFICATION_NAME]) VALUES (4, 'Top Secret');
GO

SET IDENTITY_INSERT agency.[CLASSIFICATION] OFF
GO

SET IDENTITY_INSERT agency.[USER] ON
GO

--insert USER table
      INSERT INTO [agency].[USER] ([USER_ID], [USER_NAME], [USER_PASSWORD],
[CLASSIFICATION_ID]) VALUES (1, 'jdoe', 'p@ssw0rd1', 2);
      INSERT INTO [agency].[USER] ([USER_ID], [USER_NAME], [USER_PASSWORD])
VALUES (2, 'asmith', 'p@ssw0rd2');
      INSERT INTO [agency].[USER] ([USER_ID], [USER_NAME], [USER_PASSWORD])
VALUES (3, 'tanderson', 'p@ssw0rd3');
GO

SET IDENTITY_INSERT agency.[USER] OFF
GO

--adding them as users to the RDBMS
CREATE USER [jdoe] WITHOUT LOGIN WITH DEFAULT_SCHEMA = [agency]
CREATE USER asmith WITHOUT LOGIN WITH DEFAULT_SCHEMA = [agency]
CREATE USER tanderson WITHOUT LOGIN WITH DEFAULT_SCHEMA = [agency]

exec sp_addrolemember 'BLRole', 'jdoe'
exec sp_addrolemember 'BLRole', 'asmith'
exec sp_addrolemember 'BLRole', 'tanderson'
--exec sp_addrolemember 'BLRole', 'dbo'
GO

--insert USER_COMPARTMENT_CLASSIFICATION table
      INSERT INTO [agency].[USER_COMPARTMENT_CLASSIFICATION] ([USER_ID],
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (1, 1, 1);
```

```
      INSERT INTO [agency].[USER_COMPARTMENT_CLASSIFICATION] ([USER_ID],
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (2, 1, 1);
      INSERT INTO [agency].[USER_COMPARTMENT_CLASSIFICATION] ([USER_ID],
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (3, 1, 1);
      INSERT INTO [agency].[USER_COMPARTMENT_CLASSIFICATION] ([USER_ID],
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (1, 3, 3); -- jdoe under smooth has
Classified access
```

This script simply populates data, but there is a RDBMS specific security feature used in this script. The “create user” command allows the three users to also be created in the RDBMS; when the user logs in successfully, that user will always be on the RDBMS connection. This allows the BLP model to assert the user has access to rows when operations are executed.

Lastly the “sp_addrolemember” command is a stored procedure built into the RDBMS to add those users to the role created in the previous script.

Tables – Agency model

Next the “Agency” tables are created and reference the Classification and Compartment tables from the BLP model. The last part of the script “DENY”’s the role from performing “Select” statements against the tables. All access is done through views.

```
USE BLPDB
GO

-- Add business tables
CREATE TABLE [agency]._SORTIE (
  SORTIE_ID INT NOT NULL,
  SORTIE_NAME VARCHAR(30) NOT NULL,
  SORTIE_POPULATION INT CHECK (SORTIE_POPULATION >= 0),
  COMPARTMENT_ID INT NOT NULL DEFAULT(1),
  CLASSIFICATION_ID INT NOT NULL DEFAULT(1),
 PRIMARY KEY CLUSTERED (SORTIE_ID ASC),
 FOREIGN KEY (COMPARTMENT_ID) REFERENCES
[agency].[COMPARTMENT]([COMPARTMENT_ID]),
 FOREIGN KEY ([CLASSIFICATION_ID]) REFERENCES
[agency].[CLASSIFICATION]([CLASSIFICATION_ID])
);

CREATE TABLE [agency]._INFORMANT (
  INFORMANT_ID INT NOT NULL,
  INFORMANT_FIRSTNAME VARCHAR(30),
  INFORMANT_LASTNAME VARCHAR(30),
```

```
  INFORMANT_GENDER CHAR(1) CHECK (INFORMANT_GENDER IN ('M', 'F')),
  INFORMANT_BIRTHDATE DATE,
  COMPARTMENT_ID INT NOT NULL DEFAULT(1),
  CLASSIFICATION_ID INT NOT NULL DEFAULT(1),
 PRIMARY KEY CLUSTERED (INFORMANT_ID ASC),
 FOREIGN KEY (COMPARTMENT_ID) REFERENCES
[agency].[COMPARTMENT]([COMPARTMENT_ID]),
 FOREIGN KEY ([CLASSIFICATION_ID]) REFERENCES
[agency].[CLASSIFICATION]([CLASSIFICATION_ID])
);

CREATE TABLE [agency]._SECRET (
  SECRET_ID INT IDENTITY(1,1) NOT NULL,
  INFORMANT_ID INT,
  SORTIE_ID INT NOT NULL,
  SECRET_TITLE VARCHAR(100) NOT NULL,
  SECRET_DESCRIPTION VARCHAR(500) NOT NULL,
  SECRET_DATETIME DATETIME,
  COMPARTMENT_ID INT NOT NULL DEFAULT(1),
  CLASSIFICATION_ID INT NOT NULL DEFAULT(1),
  PRIMARY KEY CLUSTERED (SECRET_ID ASC),
  FOREIGN KEY (INFORMANT_ID) REFERENCES [agency]._INFORMANT(INFORMANT_ID),
  FOREIGN KEY (SORTIE_ID) REFERENCES [agency]._SORTIE(SORTIE_ID),
  FOREIGN KEY (COMPARTMENT_ID) REFERENCES
[agency].[COMPARTMENT]([COMPARTMENT_ID]),
  FOREIGN KEY ([CLASSIFICATION_ID]) REFERENCES
[agency].[CLASSIFICATION]([CLASSIFICATION_ID])
);
GO

DENY SELECT ON [agency].[_SECRET] TO [BLRole];
GO
DENY SELECT ON [agency].[_INFORMANT] TO [BLRole];
GO
DENY SELECT ON [agency].[_SORTIE] TO [BLRole];
GO
```

Data Population – Agency model

Simple records are staged into the Agency tables and are used for testing and demonstrations.

```
USE BLPDB
GO

-- POPULATE BUSINESS TABLES
INSERT INTO [agency]._SORTIE (SORTIE_ID, SORTIE_NAME, SORTIE_POPULATION,
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (1, 'Tim Buk Tu', 239000, 1, 1);
INSERT INTO [agency]._SORTIE (SORTIE_ID, SORTIE_NAME, SORTIE_POPULATION,
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (2, 'Shangri La', 430000, 1, 3);
INSERT INTO [agency]._SORTIE (SORTIE_ID, SORTIE_NAME, SORTIE_POPULATION,
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (3, 'Atlantis', 1980000, 3, 1);
INSERT INTO [agency]._SORTIE (SORTIE_ID, SORTIE_NAME, SORTIE_POPULATION,
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (4, 'El Dorado', 3210, 2, 4);
```

```
INSERT INTO [agency]._SORTIE (SORTIE_ID, SORTIE_NAME, SORTIE_POPULATION,
COMPARTMENT_ID, CLASSIFICATION_ID) VALUES (5, 'Xanadu', 282000, 2, 1);

INSERT INTO [agency]._INFORMANT (INFORMANT_ID, INFORMANT_FIRSTNAME,
INFORMANT_LASTNAME, INFORMANT_GENDER, INFORMANT_BIRTHDATE, CLASSIFICATION_ID)
      VALUES (1, 'Tom', 'Cruise', 'M', 'Jan 1, 1966', 4);
INSERT INTO [agency]._INFORMANT (INFORMANT_ID, INFORMANT_FIRSTNAME,
INFORMANT_LASTNAME, INFORMANT_GENDER, INFORMANT_BIRTHDATE, CLASSIFICATION_ID)
      VALUES (2, 'Amy', 'Adams', 'F', 'Mar 12, 1976', 2);
INSERT INTO [agency]._INFORMANT (INFORMANT_ID, INFORMANT_FIRSTNAME,
INFORMANT_LASTNAME, INFORMANT_GENDER, INFORMANT_BIRTHDATE, CLASSIFICATION_ID)
      VALUES (3, 'Jennifer', 'Lopez', 'F', 'Dec 10, 1970', 3);
INSERT INTO [agency]._INFORMANT (INFORMANT_ID, INFORMANT_FIRSTNAME,
INFORMANT_LASTNAME, INFORMANT_GENDER, INFORMANT_BIRTHDATE, CLASSIFICATION_ID)
      VALUES (4, 'Jude', 'Lay', 'M', 'Aug 22, 1966', 2);
INSERT INTO [agency]._INFORMANT (INFORMANT_ID, INFORMANT_FIRSTNAME,
INFORMANT_LASTNAME, INFORMANT_GENDER, INFORMANT_BIRTHDATE, CLASSIFICATION_ID)
      VALUES (5, 'Bob', 'Unimportant', 'M', 'Feb 22, 1983', 1);

INSERT INTO [agency]._SECRET (INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID,
SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)
      VALUES (1, 3, 3, 'High value target sighting', 'While filming Aquaman
2, I saw the leader of the resistance, "Salad Fingers" in town square.',
'Apr, 10, 2013 18:30:00');
INSERT INTO [agency]._SECRET (INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID,
SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)
      VALUES (4, 3, 4, 'Eureka!', 'Salad Fingers'' address is 1002 Poseidon
Street.', 'Apr, 13, 2013 13:00:00');
INSERT INTO [agency]._SECRET (INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID,
SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)
      VALUES (3, 1, 2, 'SAM site spotted', 'On the fourth tower there is a
sam site that looks active', 'Jun, 30, 2012 11:20:00');
INSERT INTO [agency]._SECRET (INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID,
SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)
      VALUES (5, 3, 1, 'Rude servers at IHOP', 'Don''t send anybody to this
place, they are really bad', 'Apr, 30, 2013 08:00:00');
```

Views – Agency model

The views that the user will access are created in this script. For the most part, the views reflect what is in the table below. The real change are the join statements, which check that the current user has access to view the data using the “Read-down” concept of the BLP security model.

```
USE BLPDB
GO

-- ADD BUSINESS VIEWS
CREATE VIEW [agency].SORTIE
AS
SELECT a.SORTIE_ID
      , a.SORTIE_NAME
      , a.SORTIE_POPULATION
      , a.COMPARTMENT_ID
```

```
FROM [agency]._SORTIE a
JOIN agency.USER_COMPARTMENT_CLASSIFICATION ucc ON
(
      (ucc.CLASSIFICATION_ID >= a.[CLASSIFICATION_ID] AND ucc.COMPARTMENT_ID
= a.COMPARTMENT_ID )
      or
      ( a.COMPARTMENT_ID = 1 AND ucc.COMPARTMENT_ID = 1 AND
ucc.CLASSIFICATION_ID >= a.[CLASSIFICATION_ID] )
)
JOIN agency.[USER] u ON
      u.USER_NAME = USER
      AND ucc.USER_ID = u.USER_ID
GO

GRANT SELECT ON [agency].SORTIE TO [BLRole];
GO

CREATE VIEW [agency].INFORMANT
AS
SELECT i.INFORMANT_ID
      ,i.INFORMANT_FIRSTNAME
      ,i.INFORMANT_LASTNAME
      ,i.INFORMANT_GENDER
FROM [agency]._INFORMANT i
JOIN agency.USER_COMPARTMENT_CLASSIFICATION ucc ON
(
      (ucc.CLASSIFICATION_ID >= i.[CLASSIFICATION_ID] AND ucc.COMPARTMENT_ID
= i.COMPARTMENT_ID )
      or
      ( i.COMPARTMENT_ID = 1 AND ucc.COMPARTMENT_ID = 1 AND
ucc.CLASSIFICATION_ID >= i.[CLASSIFICATION_ID] )
)
JOIN agency.[USER] u ON
      u.USER_NAME = USER
      AND ucc.USER_ID = u.USER_ID

GO

GRANT SELECT ON [agency].INFORMANT TO [BLRole];
GO

CREATE VIEW [agency].[SECRET]
AS
SELECT
      s.SECRET_ID
      ,s.INFORMANT_ID
      ,s.SORTIE_ID
      ,s.SECRET_TITLE
      ,s.SECRET_DESCRIPTION
      ,s.SECRET_DATETIME
      ,s.CLASSIFICATION_ID
      ,s.COMPARTMENT_ID
      FROM [agency]._SECRET s
      JOIN agency.USER_COMPARTMENT_CLASSIFICATION ucc ON
      (
            (ucc.CLASSIFICATION_ID >= s.[CLASSIFICATION_ID] AND
ucc.COMPARTMENT_ID = s.COMPARTMENT_ID )
```

```
            or
            ( s.COMPARTMENT_ID = 1 AND ucc.COMPARTMENT_ID = 1 AND
ucc.CLASSIFICATION_ID >= s.[CLASSIFICATION_ID] )
      )
      JOIN agency.[USER] u ON
            u.USER_NAME = USER
            AND ucc.USER_ID = u.USER_ID

GO

GRANT SELECT ON [agency].[SECRET] TO [BLRole];
GO
```

Tables – Audit Model

This is a very simple, single table creation to hold audit records.

```
USE BLPDB
GO

-- ADD BUSINESS AUDIT TABLES
CREATE TABLE [agency].[AUDIT](
      [AUDIT_TABLE_NAME] varchar(30) NOT NULL,
      [AUDIT_OPERATION] varchar(30) NOT NULL,
      [AUDIT_USER] varchar(50) NOT NULL,
      [AUDIT_TIME] [datetime] NOT NULL DEFAULT (GETDATE())
)
```

Stored Procedures

There are only two stored procedures; “Login” and “Insert_Audit_Entry”. The login takes the username and password, and if successfully found, adds the user to the RDBMS connection. The latter is simply to allow a quick insert into the audit table.

```
USE BLPDB
GO

-- ADD SPROCS
GO
CREATE PROCEDURE [agency].INSERT_AUDIT_ENTRY
@TABLE varchar(30),
@OPERATION varchar(30)
AS
BEGIN
INSERT INTO [agency].[AUDIT] ([AUDIT_TABLE_NAME], [AUDIT_OPERATION],
[AUDIT_USER], [AUDIT_TIME])
VALUES (@TABLE, @OPERATION, USER, GETDATE())
END
GO
```

```
CREATE PROCEDURE [agency].[LOGIN]
@USERNAME varchar(50),
@PASSWORD varchar(20)
AS
BEGIN

IF (SELECT COUNT(*) FROM agency.[USER] WHERE [USER_NAME] = @USERNAME AND
[USER_PASSWORD] = @PASSWORD) = 1
BEGIN
      SETUSER @USERNAME
      INSERT INTO [agency].[AUDIT] ([AUDIT_TABLE_NAME], [AUDIT_OPERATION],
[AUDIT_USER], [AUDIT_TIME])
      VALUES ('USER', 'LOGIN', USER, GETDATE())
END
ELSE
BEGIN
      DECLARE @Error as varchar(50) = 'User ' + @USERNAME + ' does not
exist';
      RAISERROR (50001, 16, 1, @Error);
END

END
```

Triggers – Agency Model

This script contains the “Insert” and “Update” triggers when a user performs these respective actions on a view. The insert trigger against the SECRET view will assert both the user classification and the parent classification are not dominant over the new row’s classification. This follows the BLP model’s “Write-up” concept covered in a previous section. The update trigger performs a similar check against the parent when updating the row, again the row must dominate.

```
USE BLPDB
GO

CREATE FUNCTION agency.[GetUserClassification]()
RETURNS int
AS
BEGIN
      DECLARE @CLASSIFICATION as INT;
      SELECT @CLASSIFICATION = CLASSIFICATION_ID
      FROM agency.[USER]
      WHERE [USER_NAME] = USER;
      RETURN @CLASSIFICATION;
END
```

```
GO

CREATE FUNCTION agency.[GetCompartmentClassification](@COMPARTMENT_ID as INT)
RETURNS INT
AS
BEGIN
	DECLARE @CLASSIFICATION_ID AS INT;
	SELECT @CLASSIFICATION_ID = ucc.CLASSIFICATION_ID
	FROM agency.USER_COMPARTMENT_CLASSIFICATION ucc
	JOIN agency.[USER] u ON
		ucc.USER_ID = u.USER_ID
		AND u.USER_NAME = USER
	WHERE ucc.COMPARTMENT_ID = @COMPARTMENT_ID

	IF @CLASSIFICATION_ID is null
	BEGIN
		SET @CLASSIFICATION_ID = 1
	END

	RETURN @CLASSIFICATION_ID;
END
GO

GRANT INSERT, UPDATE, DELETE ON [agency].[SECRET] TO BLRole
GO

CREATE TRIGGER agency.OnInsertSecret ON [agency].[SECRET]
WITH ENCRYPTION
INSTEAD OF INSERT
AS
-- DATA TO INSERT
	DECLARE @INFORMANT_ID INT, @SORTIE_ID INT,
		@COMPARTMENT_ID INT, @CLASSIFICATION_ID INT,
		@SECRET_TITLE VARCHAR(100), @SECRET_DESCRIPTION VARCHAR(500),
		@SECRET_DATETIME DATETIME, @SORTIE_CLASSIFICATION_ID INT

	SELECT
		@SORTIE_ID = inserted.SORTIE_ID
		,@INFORMANT_ID = inserted.INFORMANT_ID
		,@CLASSIFICATION_ID = inserted.CLASSIFICATION_ID
		,@SECRET_TITLE = inserted.SECRET_TITLE
		,@SECRET_DESCRIPTION = inserted.SECRET_DESCRIPTION
		,@SECRET_DATETIME = inserted.SECRET_DATETIME
	FROM inserted

	SELECT
		@COMPARTMENT_ID = COMPARTMENT_ID,
		@SORTIE_CLASSIFICATION_ID = CLASSIFICATION_ID
	FROM [agency]._SORTIE
	WHERE SORTIE_ID = @SORTIE_ID

	DECLARE @Error VARCHAR(50)

	--Write Up
	IF @SORTIE_CLASSIFICATION_ID > @CLASSIFICATION_ID
	BEGIN
```

```
            SET @Error = 'Inserting a secret that is less classified than
it''s parent is forbidden';
            EXEC agency.INSERT_AUDIT_ENTRY @TABLE = 'SECRET', @OPERATION =
'WRITE-FAIL'
            RAISERROR (50002, 16, 1, @Error);
      END

      DECLARE @USERS_COMPARTMENT_CLASSIFICATION_ID INT =
agency.[GetCompartmentClassification](@COMPARTMENT_ID);

      IF @USERS_COMPARTMENT_CLASSIFICATION_ID IS NULL OR
@USERS_COMPARTMENT_CLASSIFICATION_ID > @CLASSIFICATION_ID
      BEGIN
            SET @Error = 'User ' + USER + ' is too CLASSIFIED to insert into
this COMPARTMENT';
            EXEC agency.INSERT_AUDIT_ENTRY @TABLE = 'SECRET', @OPERATION =
'WRITE-FAIL'
            RAISERROR (50002, 16, 1, @Error);
      END
PRINT @COMPARTMENT_ID
      INSERT INTO [agency]._SECRET (INFORMANT_ID, SORTIE_ID,
SECRET_DESCRIPTION, SECRET_TITLE, SECRET_DATETIME, COMPARTMENT_ID,
CLASSIFICATION_ID)
      VALUES (@INFORMANT_ID, @SORTIE_ID, @SECRET_TITLE, @SECRET_DESCRIPTION,
@SECRET_DATETIME, @COMPARTMENT_ID, @CLASSIFICATION_ID)

      EXEC agency.INSERT_AUDIT_ENTRY @TABLE = 'SECRET', @OPERATION = 'WRITE-
PASS'
GO

CREATE TRIGGER agency.OnUpdateSecret
   ON [agency].[_SECRET]
   AFTER UPDATE
AS
      DECLARE @SORTIE_ID INT, @COMPARTMENT_ID INT, @CLASSIFICATION_ID INT,
            @SORTIE_COMPARTMENT_ID INT, @SORTIE_CLASSIFICATION_ID INT

      SELECT
            @SORTIE_ID = inserted.SORTIE_ID
            ,@COMPARTMENT_ID = inserted.COMPARTMENT_ID
            ,@CLASSIFICATION_ID = inserted.CLASSIFICATION_ID
      FROM inserted

      SELECT
            @SORTIE_COMPARTMENT_ID = COMPARTMENT_ID
            ,@SORTIE_CLASSIFICATION_ID = CLASSIFICATION_ID
      FROM agency._SORTIE
      WHERE
            SORTIE_ID = @SORTIE_ID

      IF @SORTIE_COMPARTMENT_ID <> @COMPARTMENT_ID OR
@SORTIE_CLASSIFICATION_ID < @CLASSIFICATION_ID
      BEGIN
            DECLARE @Error VARCHAR(50) = 'SORTIE security can not supercede
SECRET';
            EXEC agency.INSERT_AUDIT_ENTRY @TABLE = 'SECRET', @OPERATION =
'UPDATE-FAIL'
```

```
            RAISERROR (50002, 16, 1, @Error)
            ROLLBACK
            RETURN
      END
GO
```

These are all the scripts necessary to create the BLPDB database.

Data – the BLPDB example

The data used to test the BLP security model in the BLPDB database are quite light, but is enough to test the scenarios presented in the next sections. Starting with the BLP tables, the following shows the data stored in the tables (note, audit is empty):

USER:

	USER_ID	USER_NAME	USER_PASSWORD
1	1	jdoe	p@ssw0rd1
2	2	asmith	p@ssw0rd2
3	3	tanderson	p@ssw0rd3

COMPARTMENT:

	COMPARTMENT_ID	COMPARTMENT_NAME
1	1	None
2	2	Desert Shield
3	3	Operation SMOOTH
4	4	Project X

CLASSIFICATION:

	CLASSIFICATION_ID	CLASSIFICATION_NAME
1	1	Unclassified
2	2	Restricted
3	3	Classified
4	4	Top Secret

USER_COMPARTMENT_CLASSIFICATION:

	CLASSIFICATION_ID	COMPARTMENT_ID	USER_ID
1	1	1	1
2	1	1	2
3	1	1	3
4	3	3	1

Note, All users have “unclassified” & “none” conatianer access. User ‘jdoe’ also has “Classified” & “Operation SMOOTH” access.

_SORTIE (agency tables are prefixed with underscore – the view is SORTIE)**:**

	SORTIE_ID	SORTIE_NAME	SORTIE_POPULATION	COMPARTMENT_ID	CLASSIFICATION_ID
1	1	Tim Buk Tu	239000	1	1
2	2	Shangri La	430000	1	3
3	3	Atlantis	1980000	3	1
4	4	El Dorado	3210	2	4
5	5	Xanadu	282000	2	1

_INFORMANT (agency tables are prefixed with underscore – the view is INFORMANT)**:**

	INFORMANT_ID	INFORMANT_FIRSTNAME	INFORMANT_LASTNAME	INFORMANT_GENDER	INFORMANT_BIRTHDATE	COMPARTMENT_ID	CLASSIFICATION_ID
1	1	Tom	Cruise	M	1966-01-01	1	4
2	2	Amy	Adams	F	1976-03-12	1	2
3	3	Jennifer	Lopez	F	1970-12-10	1	3
4	4	Jude	Lay	M	1966-08-22	1	2
5	5	Bob	Unimportant	M	1983-02-22	1	1

_SECRET (agency tables are prefixed with underscore – the view is SECRET)**:**

	SECRET_ID	INFORMANT_ID	SORTIE_ID	SECRET_TITLE	SECRET_DESCRIPTION	SECRET_DATETIME	COMPARTMENT_ID	CLASSIFICATION_ID
1	1	1	3	High value target sighting	While filming Aquaman 2, I saw the leader of th...	2013-04-10 18:30:00.000	1	3
2	2	4	3	Eureka!	Salad Fingers' address is 1002 Poseidon Street.	2013-04-13 13:00:00.000	1	4
3	3	3	1	SAM site spotted	On the fourth tower there is a sam site that look...	2012-06-30 11:20:00.000	1	2
4	4	5	3	Rude servers at IHOP	Don't send anybody to this place, they are really...	2013-04-30 08:00:00.000	1	1

BLP examples in the BLPDB database

This section will walk through the common flows described in a previous section. Scenarios will use the data that is already provided, but will also write new data into the agency tables.

Scenario, authentication

The first flow that was covered is authentication, and is the first step a user must do before accessing the BLPDB database.

First attempt is a negative test case.

Attempt to select from the agency tables prior to logging in.

Execution:

```
SELECT * FROM [agency].SORTIE
SELECT * FROM [agency].INFORMANT
SELECT * FROM [agency].[SECRET]
```

Output:

Results | Messages

SORTIE_ID	SORTIE_NAME	SORTIE_POPULATION	COMPARTMENT_ID

INFORMANT_ID	INFORMANT_FIRSTNAME	INFORMANT_LASTNAME	INFORMANT_GENDER

SECRET_ID	INFORMANT_ID	SORTIE_ID	SECRET_TITLE	SECRET_DESCRIPTION	SECRET_DATETIME	CLASSIFICATION_ID	COMPARTMENT_ID

As expected, no results were returned since the **join** to the current user, which doesn't exist yet, will return zero results.

Next, another negative test case, this is an attempt to authenticate an unknown user.

Execution:

```
EXEC [agency].[LOGIN] @USERNAME = 'i-dont-exist', @PASSWORD = 'p@ssw0rd1'
```

Output:

Messages

```
Msg 50001, Level 16, State 1, Procedure LOGIN, Line 17
INVALID LOGIN
```

As expected, the RDBMS message 50001 added in the create script is returned; signaling an unsuccessful login using an unknown set of credentials.

Positive test case, successfully authenticating a known user.

Execution:

```
EXEC [agency].[LOGIN] @USERNAME = 'jdoe', @PASSWORD = 'p@ssw0rd1'
SELECT USER;
```

Output:

Results | Messages

	(No column name)	(No column name)	(No column name)
1	S-1-9-3-2978566146-1220861976-2231449736-2754614...	jdoe	jdoe

This user exists in the USER table and was successfully authenticated. The second line in the execution statement returns the RDBMS user on the connection, validating the login was successful.

Scenario, read operation

This scenario will verify the previously covered “Read flow”.

Negative test case; all the agency tables are locked down and will not allow direct reads.

Execution:

```
SELECT * FROM [agency]._SORTIE
SELECT * FROM [agency]._INFORMANT
SELECT * FROM [agency]._SECRET
```

Output:

Messages

```
Msg 229, Level 14, State 5, Line 1
The SELECT permission was denied on the object '_SORTIE', database 'BLPDB', schema 'agency'.
Msg 229, Level 14, State 5, Line 2
The SELECT permission was denied on the object '_INFORMANT', database 'BLPDB', schema 'agency'.
Msg 229, Level 14, State 5, Line 3
The SELECT permission was denied on the object '_SECRET', database 'BLPDB', schema 'agency'.
```

As expected, the user can not access the tables directly.

Positive test case; selecting the all records from the agency views should return the “Read-down” results.

Execution:

```
SELECT * FROM [agency].SORTIE
SELECT * FROM [agency].INFORMANT
SELECT * FROM [agency].[SECRET]
```

Output:

Results | Messages

	SORTIE_ID	SORTIE_NAME	SORTIE_POPULATION	COMPARTMENT_ID
1	1	Tim Buk Tu	239000	1
2	3	Atlantis	1980000	3

	INFORMANT_ID	INFORMANT_FIRSTNAME	INFORMANT_LASTNAME	INFORMANT_G
1	5	Bob	Unimportant	M

	SECRET_ID	INFORMANT_ID	SORTIE_ID	SECRET_TITLE	SECRET_DESCRIPT
1	4	5	3	Rude servers at IHOP	Don't send anybody t

Notice that all result sets are filtered based on the user ‘jdoe’.

‘jdoe’ has classification level 1 in compartment 1, and classification 3 in compartment 3:

Results | Messages

	CLASSIFICATION_ID	COMPARTMENT_ID	USER_ID
1	1	1	1
2	1	1	2
3	1	1	3
4	3	3	1

Based on this, the output above was returned correctly.

Scenario – write operations - inserts

These scenarios cover the “write flows” from the previous section.

Positive test case; inserting a row into the SECRET table.

Execution:

```
INSERT INTO [agency].SECRET (INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID,
SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)
      VALUES (5, 3, 3, 'TEST', 'Don''t send anybody to this place, they are
really bad', 'Apr, 30, 2013 08:00:00');
SELECT * FROM [agency].[SECRET]
```

Output:

Results | Messages

	SECRET_ID	INFORMANT_ID	SORTIE_ID	SECRET_TITLE	SECRET_DESCRIPTION	SECRET_DATETIME	C
1	4	5	3	Rude servers at IHOP	Don't send anybody to this place, they are rea...	2013-04-30 08:00:00.000	1
2	5	5	3	Don't send anybody to this place, they are rea...	TEST	2013-04-30 08:00:00.000	3

'jdoe' was able to insert a record into the SECRET table successfully as expected.

The following checks were made; the parent (SORTIE #3) did not dominate the row, and 'jdoe''s classification for compartment #3 was 3 which dominated or was equal to the row's classification.

Positive test case; insert a row that is more classified than the user.

Executed:

```
INSERT INTO [agency].SECRET (INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID,
SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)
      VALUES (5, 3, 4, 'ANOTHER TEST', 'Don''t send anybody to this place,
they are really bad', 'Apr, 30, 2013 08:00:00'); --classification 4!!!
SELECT * FROM [agency].SECRET
```

Output:

Results | Messages

	SECRET_ID	INFORMANT_ID	SORTIE_ID	SECRET_TITLE	SECRET_DESCRIPTION	SECRET_DATETIME	C
1	4	5	3	Rude servers at IHOP	Don't send anybody to this place, they are rea...	2013-04-30 08:00:00.000	1
2	5	5	3	Don't send anybody to this place, they are rea...	TEST	2013-04-30 08:00:00.000	3

The insert was successful, but the result set did not return the new row. Because the new row has a classification of “4” it was inserted using the “Write-up” rule, but violated the “Read-down” rule, and was not returned in the select statement.

Negative test case; insert a row that is less classified than the user.

Executed:

```
INSERT INTO [agency].SECRET (INFORMANT_ID, SORTIE_ID, CLASSIFICATION_ID,
SECRET_TITLE, SECRET_DESCRIPTION, SECRET_DATETIME)
      VALUES (5, 3, 2, 'BAD TEST', 'Don''t send anybody to this place, they
are really bad', 'Apr, 30, 2013 08:00:00');
SELECT * FROM [agency].SECRET
```

Output:

```
Messages

(1 row(s) affected)
Msg 50002, Level 16, State 1, Procedure OnInsertSecret, Line 43
WRITE UP ERROR
```

The insert was unsuccessful - the new row has a classification of “2” and ‘jdoe’ is classified as “3”. This volates the “Write-up” rule, and the WRITE UP ERROR was returned.

Scenario – write operations – update

These test cases will test the “update flow” previously described.

Negative test case; update a row that with lower classification than the user.

Executed:

```
UPDATE [agency].SECRET SET SECRET_TITLE = 'Nice servers at IHOP' WHERE
SECRET_ID = 4
```

Output:

```
Messages

(1 row(s) affected)
Msg 50002, Level 16, State 1, Procedure OnUpdateSecret, Line 26
WRITE UP ERROR
Msg 3609, Level 16, State 1, Line 1
The transaction ended in the trigger. The batch has been aborted.
```

The update was unsuccessful - the row has a classification of “1” and ‘jdoe’ is classified as “3”.

This volates the “Write-up” rule, and the WRITE UP ERROR was returned.

Scenario – Audit records

Based on the operations performed in these scenarios, the following audit records were collected.

Positive test case; view all audit logs.

Executed:

```
SELECT * FROM [agency].AUDIT
```

Output:

	AUDIT_TABLE_NAME	AUDIT_OPERATION	AUDIT_USER	AUDIT_TIME
1	USER	LOGIN	jdoe	2013-04-26 20:18:36.427
2	SECRET	WRITE-PASS	jdoe	2013-04-26 20:42:34.900
3	SECRET	WRITE-PASS	jdoe	2013-04-26 21:12:58.887
4	SECRET	WRITE-FAIL	jdoe	2013-04-26 21:21:24.773
5	SECRET	UPDATE-FAIL	jdoe	2013-04-26 21:21:24.773

Conclusion

The proposed BLP security model has a lot of potential securing data at a granular level. Some of the biggest corporate concerns lately have centered on data and information security. This concern is unlikely to disappear or even shrink in the near future, and all signs point to this problem becoming even bigger. To reach a strong, secure database of course the RDBMS

security features are required but the BLP security model is another step in securing extremely sensitive data.

References

Jajodia, S., & Sandhu, R. (n.d.). *Toward a multilevel secure relational data model*. Informally published manuscript, Center for Secure Information Systems, George Mason University, Fairfax, VA.

Bell, D., & LaPadula, L. (1973). *Secure computer systems: Mathematical foundations*