

DB2 Label-Based Access Control, a practical guide, Part 1: Understand the basics of LBAC in DB2

A step-by-step guide to protect sensitive data

Skill Level: Intermediate

[Carmen K. Wong \(ckmwong@ca.ibm.com\)](mailto:ckmwong@ca.ibm.com)

Software Developer

IBM

[Stan Musker \(smusker@ca.ibm.com\)](mailto:smusker@ca.ibm.com)

DB2 Information Developer

IBM

04 May 2006

Label-Based Access Control (LBAC) is a security feature introduced in the DB2® Viper release. With LBAC, administrators can control read and write access of a user at the table column and row level. This tutorial includes use case scenarios that demonstrate how users can apply LBAC to protect their data from illegal access, and yet have the flexibility of allowing users to access data restrictively. The tutorial provides a step-by-step guide to creating LBAC solutions based on use-case scenarios.

Section 1. Before you start

About this series

The tutorial is divided into two parts. The first part covers the basic row protection and column protection. The second part contains more complex scenarios, and introduces the use of exemptions.

About this tutorial

This tutorial provides a guide to using DB2's Label-Based Access Control (LBAC) security feature. LBAC controls access to table objects by attaching security labels to them. Users attempting to access an object must have its security label granted to them. When there's a match, access is permitted; without a match, access is denied. There are three types of security labels:

- Row security labels. A security label associated with a data row or record in a database table.
- Column security labels. A security label associated with a column in a database table.
- User security labels. A security label granted to a database user.

A security label is composed of one or more security label components. There are three types of security label components that you can use to build your security labels:

- Sets. A set is a collection of elements where the order in which those elements appear is not important. All elements are deemed equal.
- Arrays. An array is an ordered set that can be used to represent a simple hierarchy. In an array, the order in which the elements appear is important. For example, the first element ranks higher than the second element and the second higher than the third.
- Trees. A tree represents a more complex hierarchy that can have multiple nodes and branches. For example, trees can be used to represent organizational charts. You use a security policy to define the security label components that make up a particular security label.

DB2 Security Administrator (SECADM) is required to manipulate LBAC objects. SECADM authority can only be granted by SYSADM. A database manager (DBM) does not have SECADM by default.

This tutorial shows how to use security labels to control access to data at the row level, column level, and at a combination of both row and column. You will also learn how to determine which security label component is most appropriate when creating those security labels. And finally you will learn how to use a security policy to associate your security label components with your security labels. Using examples from the financial industry and the police services area, you will:

1. Analyze the required data restrictions.
2. Design the LBAC security solution.
3. Implement the LBAC security solution.

4. See your LBAC security solution in action.

Prerequisites

This tutorial is written for DB2 database developers and DB2 database administrators. You should understand the basic concepts of LBAC.

System requirements

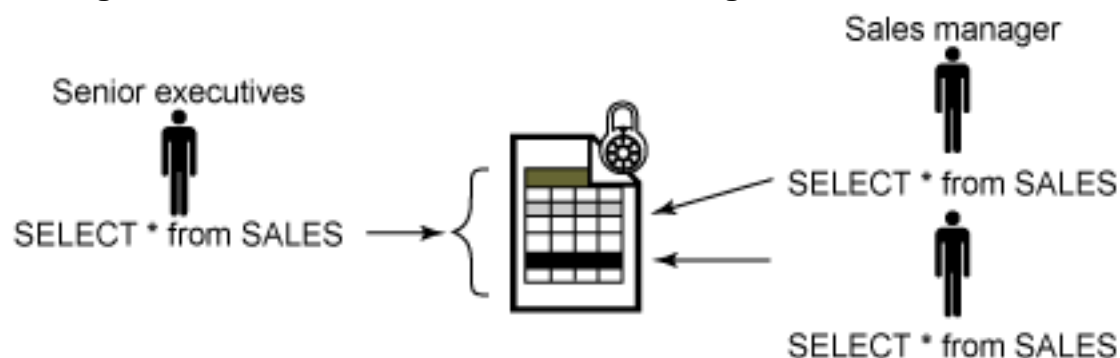
DB2 Viper for Linux®, UNIX®, and Windows®

Section 2. Lesson 1: Protecting rows

In this lesson, you will learn how to use row level protection to control access to the rows in a table.

The sales division of Global Life Financial is hosting a competition for its regional sales managers. The winner will be the regional manager whose area exceeds its sales target by the widest margin. Company executives are administering the contest and so will have access to all the sales records for all the regions, but in order to keep the competition exciting, regional managers will only be able to look at their own sales figures.

Figure 1. Senior executives are allowed to see all the records, while regional managers are restricted to the rows for their region



There are four sales regions participating in the competition. Two of the participating regions, Central-North and Central-South, are actually sub-regions of the Central region and take sales direction from its general manager. (**Note:** The Central region is not part of the competition, but its general manager is an interested observer.) The two senior executives who manage the Sales division are acting as the contest administrators. Those participating in the competition and their roles are summarized in the following table.

Table 1. Contest participants and administrators

Name	Position	Contest
Paul	Senior Vice President (senior executive)	Administrator
Bob	Vice President (senior executive)	Administrator
Sam	Sales manager for the East-Coast	Participant
Nick	Sales manager for the West-Coast	Participant
Sean	General manager for the Central Region	Observer
Becky	Sales manager for the Central-North	Participant
Marc	Sales manager for the Central-South	Participant

All data for the competition will be stored in the SALES table. This table must be created and will be similar to the existing PERFORMANCE table.

Table 2. Definition of the PERFORMANCE table

Column name	Type schema	Type name	Length	Scale	Nulls
SALES_DATE	SYSIBM	DATE	4	0	Yes
SALES_PERSON	SYSIBM	VARCHAR	15	0	Yes
REGION	SYSIBM	VARCHAR	15	0	Yes
SALES	SYSIBM	INTEGER	4	0	Yes
MARGIN	SYSIBM	INTEGER	4	0	Yes

In the next section you will analyze your security requirements.

Analyzing data restrictions

In this exercise, you need to determine how you will manage access to the SALES table. You need to enforce the following restrictions:

1. Regional sales managers are allowed READ/WRITE access only to the records for their region.
2. The general manager for the Central region can read records for the Central-North and Central-South regions.
3. Executives are allowed to read all records.

Based on this scenario, you summarize your security requirements as follows:

Table 3. Security requirements

Postition	READ access	WRITE access
Regional sales managers	Only records for their own region	Only records for their own region
General manager for the Central region	Only records for the Central-North and Central-South regions	No access
Executives	All records	No access

From your analysis, you decide to protect the sales data at the row level. For row level protection on a table, LBAC allows you to tag each row with a security label. You can then grant users a security label that allows them to access the appropriate table rows. In this case, you will create the SALES table based on the existing PERFORMANCE table (Table 2), but with an additional column for the row security label.

Table 4. A row security label column is required to control access

SALES_DATE	SALES_PERSC	REGION	SALES	MARGIN	Security Label
12/31/2004	LEE	East-Coast	2000	50	
12/31/2004	GOUNOT	West-Coast	1000	40	
01/29/2005	LUCCHESSI	Central-South	3000	30	
01/29/2005	LEE	Central-North	2000	45	

Next you will design an LBAC security solution based on your analysis.

Designing the security solution

In this exercise, you will design the security labels that will control access to the data rows in the SALES table. In designing security labels, you need to consider the following:

1. Row security labels that protect the rows.
2. User security labels that grant users the appropriate access.
3. Security label components that create the security labels.

Row security labels

From your analysis, you determine that each region requires a security label to control access to its data. So four security labels are needed, one for each sales region. The security label components for building these security labels can be constructed using the sales regions as the elements. Since all regions appear to be of equal importance, you might consider using a SET for this security label component.

User security labels for the regional managers

The appropriate security labels used to tag the rows will be granted to the regional managers so that they can access the data for their region. Their access is to this data is READ/WRITE.

User security label for general manager

The general manager of the Central region can read data for the Central-North and Central-South regions, so this security label should rank above the security labels used to protect the rows of those sub-regions. Because now there is a hierarchy, you might consider using an ARRAY or a TREE for another security label component.

Since the general manager is not allowed to write to the SALES table, some restriction should be imposed at the table level by revoking the INSERT, UPDATE, and DELETE privileges from this user. These types of restrictions will not be part of a security label component or security label, but will be imposed when you GRANT security labels to users.

User security label for executives

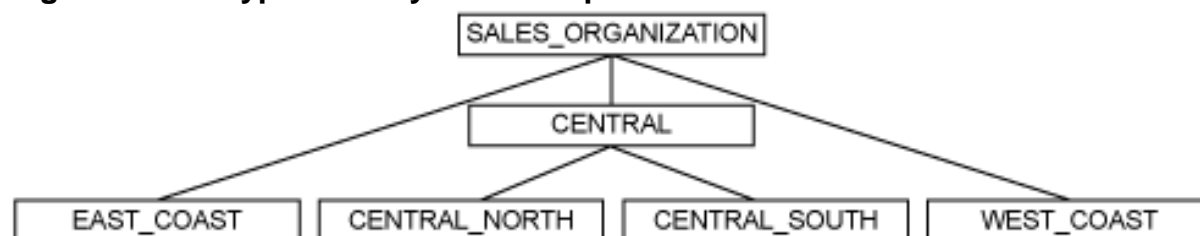
The executives can read all the sales data. One way to accomplish this is by granting all security labels to the executives, but this might not be the most efficient method. An alternative is to use a hierarchical structure, where the security label granted to the executives is higher than the security labels used to protect the rows. This hierarchy should match the organizational structure of the company and so is too complex for an ARRAY and you should consider using a TREE.

Like the general manager, the executives cannot write to the SALES table and so similar WRITE restrictions should be imposed.

Security label components

Since data access is based on Global Life Financial's geographic regions, the security label component can be constructed with a TREE structure with regions as the elements.

Figure 2. Tree type security label component



Using the nodes in the tree to make up the security label component, you can create four row security labels: one for each region in the competition. For example, the security label for tagging a sales record from 'West-Coast' can be constructed using the 'WEST_COAST' element from the security label component.

Allowing the executives to access the whole SALES table requires that their security label have a higher level of authority than that for the sales managers. A security label created using the 'SALES_ORGANIZATION' element means that a user granted that security label can access all the table records tagged with security labels that have been created with elements below it in the tree.

The next section shows you how to implement the solution that you have designed using SQL commands.

Implementing the security solution

Steps overview:

1. Defining the security policies and labels
 - Defining the security label component
 - Defining the security policy
 - Defining the security labels
2. Creating the protected SALES table by including a column that holds the security label and attaching the security policy to the table.
3. Granting the appropriate security labels to users.

Step 1: Defining the security policies and labels

Privilege and authority requirements

SECADM authority is required to execute commands for creating security policies and labels.

Step 1a: Defining the security label component

From your analysis, you have decided that a tree-type security label component can be used with the sales regions as elements. A security label component with a name SLC_REGION of tree type with the elements shown in Figure 2 can be created using the following command:

```
CREATE SECURITY LABEL COMPONENT SLC_REGION
TREE { 'SALES_ORGANIZATION' ROOT,
      'CENTRAL' UNDER 'SALES_ORGANIZATION',
      'CENTRAL_NORTH' UNDER 'CENTRAL',
      'CENTRAL_SOUTH' UNDER 'CENTRAL',
      'WEST_COAST' UNDER 'SALES_ORGANIZATION',
      'EAST_COAST' UNDER 'SALES_ORGANIZATION'
}
```

Step 1b: Defining the security policy

After the security label component is created, you need to create the security policy. A security policy with a name `SALES_REGION_POLICY` that uses the `SLC_REGION` component can be created using the following command:

```
CREATE SECURITY POLICY SALES_REGION_POLICY
  COMPONENTS SLC_REGION
  WITH DB2LBACRULES
  RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
```

Step 1c: Defining the security labels

From your analysis, you have decided that a security label is required for each sales region (four in total), a security label is required for the general manager, and a security label is required for the executives. Each security label is based on the `SALES_REGION_POLICY` security policy that you created in Step 1b. The required security labels are created using the following commands:

```
CREATE SECURITY LABEL SALES_REGION_POLICY.CENTRAL_SOUTH
  COMPONENT SLC_REGION 'CENTRAL_SOUTH'

CREATE SECURITY LABEL SALES_REGION_POLICY.CENTRAL_NORTH
  COMPONENT SLC_REGION 'CENTRAL_NORTH'

CREATE SECURITY LABEL SALES_REGION_POLICY.EAST_COAST
  COMPONENT SLC_REGION 'EAST_COAST'

CREATE SECURITY LABEL SALES_REGION_POLICY.WEST_COAST
  COMPONENT SLC_REGION 'WEST_COAST'

CREATE SECURITY LABEL SALES_REGION_POLICY.CENTRAL
  COMPONENT SLC_REGION 'CENTRAL'

CREATE SECURITY LABEL SALES_REGION_POLICY.SALES_ORG_READ
  COMPONENT SLC_REGION 'SALES_ORGANIZATION'
```

Step 2: Creating the protected SALES table

Privilege and authority requirements

Ability to create tables:

To secure the `SALES` table with row level protection, you will need a column of type `DB2SECURITYLABEL` to hold the security label and associate the table with the `SALES_REGION_POLICY` security policy.

```
CREATE TABLE SALES (SALES_DATE DATE,
  SALES_PERSON VARCHAR (15),
  REGION VARCHAR (15),
  SALES INTEGER,
  MARGIN INTEGER,
  REGION_TAG DB2SECURITYLABEL)
  SECURITY POLICY SALES_REGION_POLICY
```

Upon successful execution of the command, the `SALES` table is protected.

Table 5. Definition of the protected SALES table

Column name	Type schema	Type name	Length	Scale	Nulls
SALES_DATE	SYSIBM	DATE	4	0	Yes

SALES_PERSON	SYSIBM	VARCHAR	15	0	Yes
REGION	SYSIBM	VARCHAR	15	0	Yes
SALES	SYSIBM	INTEGER	4	0	Yes
MARGIN	SYSIBM	INTEGER	4	0	Yes
REGION_TAG	SYSIBM	DB2SECURITYLABEL		0	No

And when populated with the data from the PERFORMANCE table, the SALES table would look like this.

Table 6. Each row is tagged with a row security label that controls access based on the region

SALES_DATE	SALES_PERSON	REGION	SALES	MARGIN	Security Label
12/31/2004	LEE	East-Coast	2000	50	SALES_REGION_POLICY.EAST
12/31/2004	GOUNOT	West-Coast	1000	40	SALES_REGION_POLICY.WES
01/29/2005	LUCCHESI	Central-South	3000	30	SALES_REGION_POLICY.CENT
01/29/2005	LEE	Central-North	2000	45	SALES_REGION_POLICY.CENT

Step 3: Granting the security labels to users

Privilege and authority requirements

SECADM authority is required to execute commands for granting labels to users.

After the SALES table has been protected, no users can access the table until security labels are granted to them. To allow the regional managers access to the regional data of the SALES table, grant each manager the security label that corresponds to their sales region with READ/WRITE authority. For example, Nick is the sales manager of West-Coast, and so SALES_REGION_POLICY.WEST_COAST will be granted to Nick. The general manager, Sean, will be granted the security label to SALES_REGION_POLICY.CENTRAL with read access. The executives, Paul and Bob, will be granted SALES_REGION_POLICY.SALES_ORG_READ, with read access to all the records in the table.

```
GRANT SECURITY LABEL SALES_REGION_POLICY.EAST_COAST
  TO USER Sam FOR ALL ACCESS

GRANT SECURITY LABEL SALES_REGION_POLICY.WEST_COAST
  TO USER Nick FOR ALL ACCESS

GRANT SECURITY LABEL SALES_REGION_POLICY.CENTRAL_NORTH
  TO USER Becky FOR ALL ACCESS

GRANT SECURITY LABEL SALES_REGION_POLICY.CENTRAL_SOUTH
  TO USER Marc FOR ALL ACCESS

GRANT SECURITY LABEL SALES_REGION_POLICY.CENTRAL
  TO USER Sean FOR READ ACCESS

GRANT SECURITY LABEL SALES_REGION_POLICY.SALES_ORG_READ
  TO USER Paul FOR READ ACCESS

GRANT SECURITY LABEL SALES_REGION_POLICY.SALES_ORG_READ
```

```
TO USER Bob FOR READ ACCESS
```

The next section shows how well your LBAC security solution works.

Watching the solution in action

This section explains some possible runtime scenarios after the SALES table has been protected.

Example 1

Sam, the East-Coast sales manager, tries to insert into the SALES table:

```
INSERT into INSURANCE.SALES VALUES (  
    '2005-02-28',  
    'LUCCHESSI',  
    'East-Coast',  
    344,  
    40,  
    SECLABEL_BY_NAME('SALES_REGION_POLICY', 'SL_EAST_COAST'));
```

The command executes successfully. The resulting row contains:

('2005-02-28','LUCCHESSI','East-Coast',344,40,<SALES_REGION_POLICY.EAST_COAST>)

Example 2

Sam tries to insert sales data of another region:

```
INSERT into INSURANCE.SALES VALUES (  
    '2005-02-28',  
    'Brian',  
    'Central-South',  
    344,  
    28,  
    SECLABEL_BY_NAME ('SALES_REGION_POLICY', 'SL_CENTRAL_SOUTH'))
```

The command fails, because the security label SALES_REGION_POLICY.CENTRAL_SOUTH is not granted to Sam.

Note: If the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option is not specified in the CREATE SECURITY POLICY command, this command will execute successfully. However, Sam's security label for write access will be written in the last column instead of SL_CENTRAL_SOUTH.

Example 3

Sam tries to insert sales data using the security label granted to him:

```
INSERT into INSURANCE.SALES (  
    SALES_DATE, SALES_PERSON, REGION, SALES, MARGIN)  
VALUES (  
    '2005-02-12',  
    'Peter',  
    'East-Coast',  
    450,  
    40)
```

The command executes successfully. The two rows inserted in Example 1 and 2 are returned:

```
('2005-02-28','LUCCHESSI','East-Coast',344,40,<SALES_REGION_POLICY.EAST_COAST>)
('2005-02-12','Peter','East-Coast',450,40,<SALES_REGION_POLICY.SL_EAST_COAST>)
```

Example 4

Marc, the Central-South sales manager, tries to insert sales data into the table:

```
INSERT into INSURANCE.SALES (
  SALES_DATE, SALES_PERSON, REGION, SALES, MARGIN)
VALUES (
  '2005-02-25',
  'Brian',
  'Central-South',
  390,
  43,
  SECLABEL_BY_NAME('SALES_REGION_POLICY', 'SL_CENTRAL_SOUTH'));
```

The command executes successfully. The resulting row contains:

```
('2005-02-25','Brian','Central-South',390,43,<SALES_REGION_POLICY.SL_CENTRAL_SOUTH>)
```

Marc then tries to read the table by issuing:

```
SELECT sales_date, sales_person, region, sales, margin,
  varchar(SECLABEL_TO_CHAR('SALES_REGION_POLICY',REGION_TAG), 30)
from INSURANCE.SALES
```

The rows tagged with the SALES_REGION_POLICY.CENTRAL_SOUTH security label are returned:

```
('2005-02-25','Brian','Central-South',390,43,<SALES_REGION_POLICY.CENTRAL_SOUTH>)
```

Example 5

Sam tries to read sales data of other region by issuing:

```
SELECT sales_date, sales_person, region, sales, margin,
  varchar(SECLABEL_TO_CHAR('SALES_REGION_POLICY',REGION_TAG), 30)
from INSURANCE.SALES where REGION='Central-South'
```

No row is returned. The row with Central-South in the region column is protected by security label SALES_REGION_POLICY.SL_CENTRAL_SOUTH. The read access security label that Sam holds is not authorized to read that row.

Example 6

Sam tries to read sales data from the INSURANCE.SALES table by issuing:

```
SELECT sales_date, sales_person, region, sales, margin,
  varchar(SECLABEL_TO_CHAR('SALES_REGION_POLICY',REGION_TAG), 30)
from INSURANCE.SALES;
```

The row with SALES_REGION_POLICY.SL_EAST_COAST as the value in the REGION_TAG is returned:

```
('2005-02-28','LUCCHESSI','East-Coast',344,40,<SALES_REGION_POLICY.SL_EAST_COAST>)
('2005-02-12','Peter','East-Coast',450,40,<SALES_REGION_POLICY.SL_EAST_COAST>)
```

Example 7

Becky, the Central-North sales manager, tries to insert sales data for another region with the security label granted to her:

```
INSERT into INSURANCE.SALES (
    SALES_DATE, SALES_PERSON, REGION, SALES, MARGIN)
VALUES ( '2005-01-28',
    'Owen',
    'Central-North',
    300,
    36,
    SECLABEL_BY_NAME( 'SALES_REGION_POLICY', 'SL_CENTRAL_NORTH' ) );
```

The command executes successfully. The resulting row contains:

```
('2005-01-28','Owen','Central-North',300,36,<SALES_REGION_POLICY.SL_CENTRAL_NORTH>)
```

Becky, then tries to read the table by issuing:

```
SELECT sales_date, sales_person, region, sales, margin,
    varchar(SECLABEL_TO_CHAR( 'SALES_REGION_POLICY',REGION_TAG), 30)
from INSURANCE.SALES
```

The rows tagged with the SALES_REGION_POLICY.SL_CENTRAL_NORTH security label are returned:

```
('2005-01-28','Owen','Central-North',300,36,SL_CENTRAL_NORTH)
```

Example 8

Sean, the Central region general manager, tries to read the sales data for the regions:

```
SELECT sales_date, sales_person, region, sales, margin,
    varchar(SECLABEL_TO_CHAR( 'SALES_REGION_POLICY',REGION_TAG), 30)
from INSURANCE.SALES
```

The command returns the rows tagged with the SALES_REGION_POLICY.SL_CENTRAL_SOUTH or SALES_REGION_POLICY.SL_CENTRAL_NORTH security labels:

```
('2005-02-25','Brian','Central-South',390,43,<SALES_REGION_POLICY.SL_CENTRAL_SOUTH>)
('2005-01-28','Owen','Central-North',300,36,<SALES_REGION_POLICY.SL_CENTRAL_NORTH>)
```

Example 9

Paul, a senior executive, tries to read the sales data by issuing:

```
SELECT sales_date, sales_person, region, sales, margin,  
       varchar(SECLABEL_TO_CHAR('SALES_REGION_POLICY',REGION_TAG), 30)  
from INSURANCE.SALES
```

The command executes successfully and returns all the rows in the INSURANCE.SALES table:

```
('2005-02-28','LUCCHESSI','East-Coast',344,40,<SALES_REGION_POLICY.EAST_COAST>)  
( '2005-02-12','Peter','East-Coast',450,40,<SALES_REGION_POLICY.SL_EAST_COAST>)  
( '2005-02-25','Brian','Central-South',390,43,<SALES_REGION_POLICY.SL_CENTRAL_SOUTH>)  
( '2005-01-28','Owen','Central-North',300,36,<SALES_REGION_POLICY.SL_CENTRAL_NORTH>)
```

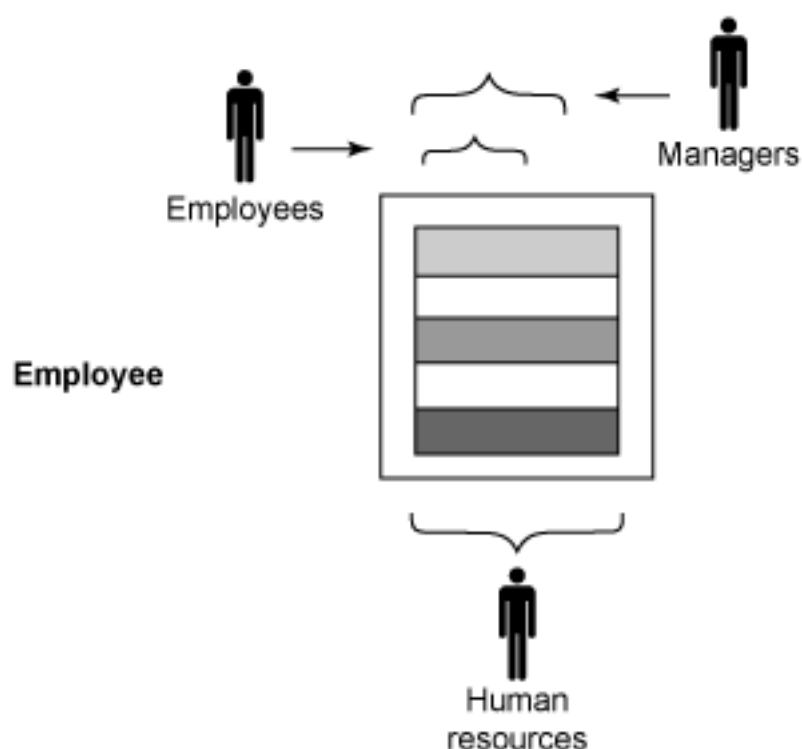
Section 3. Lesson 2: Protecting column

In this lesson, you will learn how to use column level protection to control access to the columns in a table.

The Human Resource department of Global Life Financial would like to allow employees, managers, and HR staff to access data in the EMPLOYEE table. This table contains information with different levels of sensitivity and so some restrictions should be made with regards to access:

- Name, gender, department, and phone number are considered to be unclassified information and can be available to all employees.
- Employee number, hire date, job, and education level are confidential and should be restricted to managers and HR staff.
- Birth date, salary, bonus, and commission are highly confidential information HR staff.

Figure 3. Levels of access for the EMPLOYEE table



Some of the users who access the table are summarized in the following table.

Table 7. Staff who access the EMPLOYEE table.

Name	Position
Jen	HR staff
Noel	Manager
Sunny	Regular employee

The existing EMPLOYEE table will be tagged with security labels that indicate a column's level of sensitivity.

Table 8. Classifying the columns of the existing EMPLOYEE table.

Column name	Type schema	Type name	Length	Scale	Nulls
(C) EMPNO	SYSIBM	CHARACTER	6	0	No
(U) FIRSTNME	SYSIBM	VARCHAR	12	0	No
(U) MIDINIT	SYSIBM	CHARACTER	1	0	No
(U) LASTNAME	SYSIBM	VARCHAR	15	0	No
(U) WORKDEPT	SYSIBM	CHARACTER	3	0	Yes
(U) PHONENO	SYSIBM	CHARACTER	4	0	Yes
(U) GENDER	SYSIBM	CHARACTER	1	0	Yes
(U) GENDER	SYSIBM	CHARACTER	1	0	Yes
(C) HIREDATE	SYSIBM	DATE	4	0	Yes

(C) JOB	SYSIBM	CHARACTER	8	0	Yes
(C) EDLEVEL	SYSIBM	SMALLINT	2	0	No
(H) BIRTHDATE	SYSIBM	DATE	4	0	Yes
(H) SALARY	SYSIBM	DECIMAL	9	0	Yes
(H) BONUS	SYSIBM	DECIMAL	9	0	Yes
(H) COMM	SYSIBM	DECIMAL	9	0	Yes
Column security class: (U) Unclassified, (C) Confidential, (H) Highly Confidential					

In the next section you will analyze your security requirements.

Analyzing data restrictions

In this exercise, you need to determine how you will manage access to the columns of the EMPLOYEE table. You need to enforce the following restrictions:

1. Anyone with access to the EMPLOYEE table can read unclassified columns.
2. Managers can also read all confidential columns.
3. HR staff have READ/WRITE access to all columns in the table.

Based on this scenario, you summarize your security requirements as follows:

Table 9. Security requirements

Position	READ access	WRITE access
HR staff	All	All
Managers	Confidential and Unclassified columns	No access
Regular employees	Unclassified columns	No access

Next you will design an LBAC security solution based on your analysis.

Designing the security solution

In this exercise, you will design the security labels that will control access to the columns in the EMPLOYEE table. In designing security labels, you need to consider the following:

1. Column security labels that protect the different levels of sensitivity.
2. User security labels that grant users the appropriate access.

3. Security label components that create the security labels.

Column security labels

From your analysis, you determine that each column requires a security label based on its sensitivity. And so, three security labels are needed, one for each level of sensitivity: HIGHLY CONFIDENTIAL, CONFIDENTIAL, and UNCLASSIFIED. This seems to be a simple hierarchy and you consider using an ARRAY for the security label component.

User security labels for employees

Regular employees can only access unclassified information. If unclassified columns are protected with a security label of UNCLASSIFIED, then that label should be granted to regular employees.

Since regular employees are not allowed to write to the EMPLOYEE table, some restriction should be imposed at the table level by revoking the INSERT, UPDATE and DELETE privileges from these users when you GRANT the security label.

User security labels for managers

Managers can access unclassified and confidential information. If confidential columns are protected with a security label of CONFIDENTIAL, then that label should be granted to managers. An ARRAY still seems to be appropriate for the security label. If the order of elements in the array is (CONFIDENTIAL, UNCLASSIFIED), then managers granted a CONFIDENTIAL security label would have access to all information at the CONFIDENTIAL level and any levels below (in this case UNCLASSIFIED).

Since managers are also not allowed to write to the EMPLOYEE table, some restriction should be imposed at the table level by revoking the INSERT, UPDATE and DELETE privileges from these users.

User security labels for human resources

HR has the highest level of access to the EMPLOYEE table and can access all information. If highly confidential columns are protected with a security label of HIGHLY CONFIDENTIAL, then that label should be granted to the HR staff. An ARRAY still seems to be appropriate for the security label. If the order of elements in the array is (HIGHLY CONFIDENTIAL, CONFIDENTIAL, UNCLASSIFIED), then HR members granted a HIGHLY CONFIDENTIAL security label would have access to all information at the HIGHLY CONFIDENTIAL level and below (in this case CONFIDENTIAL and UNCLASSIFIED).

Their access to data in the EMPLOYEE table should be READ/WRITE.

Security label components

Since data access is based on a linear hierarchy, the security label component can

be constructed with an ARRAY ordered as HIGHLY CONFIDENTIAL, CONFIDENTIAL, UNCLASSIFIED.

The next section shows you how to implement the solution that you have designed using SQL commands.

Implementing the security solution

Steps overview:

1. Defining the security policies and labels
 - Defining the security label component
 - Defining the security policy
 - Defining the security labels
2. Altering the EMPLOYEE table by protecting all columns with security labels and attaching the security policy to the table.
3. Granting the appropriate security labels to users.

Step 1: Defining the security policies and labels

Privilege and authority requirements

Requires SECADM authority to execute commands for creating security policies and labels.

Step 1a: Defining the security label component

From your analysis, you have decided that an array type security label component can be used for this ordered set of elements: HIGHLY CONFIDENTIAL, CONFIDENTIAL, and UNCLASSIFIED. The security label component can be created using the following command:

```
CREATE SECURITY LABEL COMPONENT SLC_LEVEL  
ARRAY [ 'HIGHLY CONFIDENTIAL', 'CONFIDENTIAL', 'UNCLASSIFIED' ]
```

Step 1b: Defining the security policy

After the security label component is created, you need to create the security policy. A security policy with a name ACCESS_EMPLOYEE_POLICY that uses the SLC_LEVEL component can be created using the following command:

```
CREATE SECURITY POLICY ACCESS_EMPLOYEE_POLICY  
COMPONENTS SLC_LEVEL  
WITH DB2LBACRULES  
RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
```

Step 1c: Defining the security labels

From your analysis, you have decided that a security label is required for each classification type (three in total). Each security label is based on the `ACCESS_EMPLOYEE_POLICY` security policy that you created in Step 1b. The required security labels are created using the following commands:

```
CREATE SECURITY LABEL ACCESS_EMPLOYEE_POLICY.HIGHCONFIDENTIAL
  COMPONENT SLC_LEVEL 'HIGHLY CONFIDENTIAL'

CREATE SECURITY LABEL ACCESS_EMPLOYEE_POLICY.CONFIDENTIAL
  COMPONENT SLC_LEVEL 'CONFIDENTIAL'

CREATE SECURITY LABEL ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
  COMPONENT SLC_LEVEL 'UNCLASSIFIED'
```

Step 2: Altering the EMPLOYEE table

Privilege and authority requirements

ALTER table privilege on the `EMPLOYEE` table:

To secure the `EMPLOYEE` table with column level protection, you will need to alter the columns by attaching the appropriate security labels and associate the table with the `ACCESS_EMPLOYEE_POLICY` security policy.

```
ALTER TABLE EMPLOYEE
  ALTER EMPNO SECURED WITH ACCESS_EMPLOYEE_POLICY.CONFIDENTIAL
  ALTER FIRSTNAME SECURED WITH ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
  ALTER MIDINIT SECURED WITH ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
  ALTER LASTNAME SECURED WITH ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
  ALTER WORKDEPT SECURED WITH ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
  ALTER PHONENO SECURED WITH ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
  ALTER GENDER SECURED WITH ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
  ALTER HIREDATE SECURED WITH ACCESS_EMPLOYEE_POLICY.CONFIDENTIAL
  ALTER JOB SECURED WITH ACCESS_EMPLOYEE_POLICY.CONFIDENTIAL
  ALTER EDLEVEL SECURED WITH ACCESS_EMPLOYEE_POLICY.CONFIDENTIAL
  ALTER BIRTHDATE SECURED WITH ACCESS_EMPLOYEE_POLICY.HIGHCONFIDENTIAL
  ALTER SALARY SECURED WITH ACCESS_EMPLOYEE_POLICY.HIGHCONFIDENTIAL
  ALTER BONUS SECURED WITH ACCESS_EMPLOYEE_POLICY.HIGHCONFIDENTIAL
  ALTER COMM SECURED WITH ACCESS_EMPLOYEE_POLICY.HIGHCONFIDENTIAL
  ADD SECURITY POLICY ACCESS_EMPLOYEE_POLICY
```

Upon successful execution of the commands, the `EMPLOYEE` table is protected.

Step 3: Granting the security labels to users

Privilege and authority requirements

This requires `SECADM` authority to execute commands for granting labels to users.

After the `EMPLOYEE` table has been protected, no users can access the table until security labels are granted to them. To allow the staff access to the protected `EMPLOYEE` table, the following labels should be granted:

```
GRANT SECURITY LABEL ACCESS_EMPLOYEE_POLICY.HIGHCONFIDENTIAL
  TO USER Jen
  FOR ALL ACCESS

GRANT SECURITY LABEL ACCESS_EMPLOYEE_POLICY.CONFIDENTIAL
```

```
TO USER Noel
FOR READ ACCESS

GRANT SECURITY LABEL ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED
TO USER Sunny
FOR READ ACCESS
```

The next section shows how well your LBAC security solution works.

Watching the solution in action

This section explains some possible runtime scenarios after the SALES table has been protected.

Example 1

Jen, from HR, tries to read the EMPLOYEE table by issuing:

```
SELECT * from EMPLOYEE;
```

The command is successful.

Example 2

Noel, a manager, tries to read the EMPLOYEE table by issuing:

```
SELECT * from HR.EMPLOYEE;
```

The command fails because it violates the read access restrictions for the highly confidential columns.

Noel then tries to read only the unclassified and confidential columns:

```
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT,
       PHONENO, HIREDATE, JOB, EDLEVEL
from HR.EMPLOYEE;
```

The command executes successfully.

Example 3

Sunny, a regular employee, tries to read the EMPLOYEE table by issuing:

```
SELECT FIRSTNME, LASTNAME, PHONENO from HR.EMPLOYEE;
```

The command succeeds because Sunny holds the security label ACCESS_EMPLOYEE_POLICY.UNCLASSIFIED and is only trying to read unclassified information.

Sunny then tries to read the confidential columns:

```
SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
       WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL
```

```
from HR.EMPLOYEE;
```

The command fails because it violates the read access restrictions.

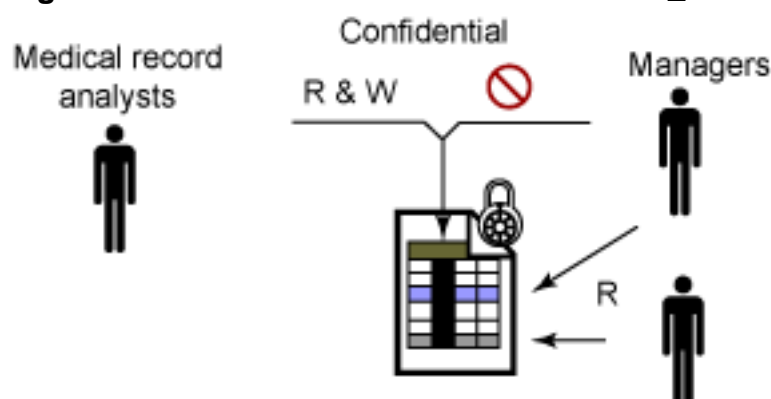
Section 4. Lesson 3: Protecting rows and columns

In this lesson, you will learn how to use a combination of row level and column level protection to protect sensitive data in a table.

Global Life Financial requires all life insurance applicants to submit a Medical History report for analyzing the eligibility of their application. An applicant's medical history is highly confidential and should only be handled by the Medical Record Analysis department. Upon analyzing the report, that department comes up with a rating in form of a risk index. Department managers can then refer to the insurance risk index when they consider approving an insurance application. To maintain confidentiality, department managers cannot access the information related to the medical history of a client's record.

Medical histories and risk index are stored in the existing MEDICAL_RECORD table.

Figure 4. Levels of access for the MEDICAL_RECORD table.



Those who access the table are summarized in the following table.

Table 10. Some of the staff who could access the MEDICAL_RECORD table.

Name	Position
Peter	Medical record analyst
Andrea	Manager of department K01
Sara	Manager of department K02
Kevin	Manager of department S01
Joseph	Manager of department S02

In the MEDICAL_RECORD table, the confidential column can be accessed by medical record analysts only. Manager access is restricted to the client records for

their department.

Table 11. Definition for the existing MEDICAL_RECORD table.

Column name	Type schema	Type name	Length	Scale	Nulls
RECORDID	SYSIBM	CHARACTER	6	0	No
CLIENTNO	SYSIBM	CHARACTER	6	0	No
DEPTNO	SYSIBM	CHARACTER	6	0	No
APPLICATION_ DATE	SYSIBM	DATE	4	0	Yes
LAST_ UPDATE	SYSIBM	DATE	4	0	No
MEDICAL_ HISTORY	SYSIBM	CLOB	1048576	0	No
RISK_ INDEX	SYSIBM	SMALLINT	1	0	No
Note: The shaded column contains confidential information.					

In the next section you will analyze your security requirements.

Analyzing data restrictions

In this exercise, you need to determine how to manage access to the confidential column and restrict record access by group. You need to enforce the following restrictions:

- Columns
 - Medical record analysts have READ/WRITE access to confidential columns.
 - Managers have READ access to non-confidential columns.
- Rows:
 - Medical record analysts can read and update all the records.
 - Managers can read but not update client records for their department.

Based on this scenario, you summarize your security requirements as follows:

Table 12. Security requirements

Postition	READ access	WRITE access
Medical record analyst	All records	All records
Managers	Client records for their department and only non-confidential columns	No access

To restrict access to the column that is confidential, the column can be protected with a security label. To restrict managers' access to only the records for their department, each row can be tagged with a security label that indicates the department. The write restriction for managers can be implemented by revoking their write privileges to the table. In Table 13, below, see how a column security label is added to the MEDICAL_HISTORY column to control access based on job category.

Table 13. A row security label is added to control access by department.

RECORDID	CLIENTNO	DEPTNO	ALLOCATION DATE	LAST_UPDATE	MEDICAL_HISTORY	RISK_FACTOR	Row Security Label
000010	K108341	K01	2005/01/05	2005/01/15	...	0	
000020	K181245	K01	2005/02/09	2005/02/19	...	2	
000030	S245987	S02	2005/02/11	2005/02/21	...	1	
000050	S231674	S02	2005/03/23	2005/04/04	...	1	

Next you will design an LBAC security solution based on your analysis.

Designing the security solution

In this exercise, you will design the security labels that will control access to the data in the MEDICAL_RECORD table. In designing a security solution, you need to consider:

1. Row and column security labels that protect the columns and rows.
2. User security labels that grant users the appropriate access.
3. Security label components that create the security labels.

Column security labels

For column protection, a column security label is required for the confidential column. The security label component for building this security label could simply be an element called CONFIDENTIAL. Using a SET seems appropriate because there is only one element.

Row security labels

From your analysis, you determine that each department requires a security label to control access to its data. And so, four security labels are needed, one for each department. Since all departments appear to be of equal importance you might consider using a SET for this security label component.

User security label for the medical record analyst

The medical record analyst is allowed READ/WRITE access to the records for all

departments, including the confidential column. So this security label should include the elements from both the column security label and the row security label. Because the medical record analyst can access all records for all departments, this indicates that there is a hierarchy and that a TREE might be a better alternative for the row security label.

User security label for the managers

The appropriate security labels used to tag the rows will be granted to the department managers so that they can access the records for their department. Their access to this data is READ except for the confidential column.

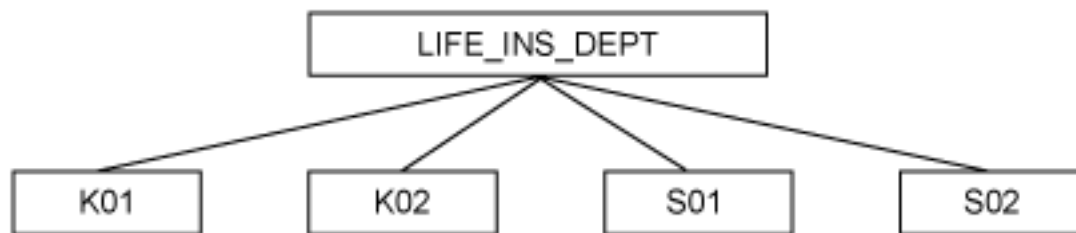
Security label components

Two security label components are required.

A SET type security label component is required for the column security label, with one element 'CONFIDENTIAL'.

A TREE type security label component is required for the row security label, with 'LIFE_INS_DEPT' as the root, and the department names 'K01', 'K02', 'S01', 'S02' as the child elements.

Figure 5. Tree type security label component.



The next section shows you how to implement the solution that you have designed using SQL commands.

Implementing the security solution

Steps overview:

1. Defining the security policies and labels
2. a. Defining the security label component
3. b. Defining the security policy
4. c. Defining the security labels
5. Altering the MEDICAL_RECORD table by adding a security label column for row level protection, marking the confidential column as protected, and attaching the security policy to the table.

6. Updating the MEDICAL_RECORD table security label column.
7. Granting the appropriate security labels to users.

Step 1: Defining the security policies and labels

Privilege and authority requirements

Requires SECADM authority to execute commands for creating security policies and labels.

Step 1a: Defining the security label components

From your analysis, two security label components are required. The security label components can be created using the following commands:

```
CREATE SECURITY LABEL COMPONENT SLC_LEVEL
SET {'CONFIDENTIAL'}

CREATE SECURITY LABEL COMPONENT SLC_LIFEINS_ORG
TREE {'LIFE_INS_DEPT' ROOT,
      'K01' UNDER 'LIFE_INS_DEPT',
      'K02' UNDER 'LIFE_INS_DEPT',
      'S01' UNDER 'LIFE_INS_DEPT',
      'S02' UNDER 'LIFE_INS_DEPT'
}
```

Step 1b: Defining the security policy

After the security label components are created, the next step is to create the security policy. A security policy with a name MEDICAL_RECORD_POLICY that uses the SLC_LEVEL and SLC_LIFEINS_ORG security label components can be created using the following command:

```
CREATE SECURITY POLICY MEDICAL_RECORD_POLICY
COMPONENTS SLC_LEVEL, SLC_LIFEINS_ORG
WITH DB2LBACRULES
RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
```

Step 1c: Defining the security labels

From your analysis, the following security labels are needed:

1. A column security label.
2. Four security labels for row protection.
3. A user security label for medical record analysts.

The column security label is created with the following command:

```
CREATE SECURITY LABEL MEDICAL_RECORD_POLICY.MED_RECORD
COMPONENT SLC_LEVEL 'CONFIDENTIAL'
```


The row security labels are created with the following commands:

```
CREATE SECURITY LABEL MEDICAL_RECORD_POLICY.LIFEINS_DEPT_K01
  COMPONENT SLC_LIFEINS_ORG 'K01'

CREATE SECURITY LABEL MEDICAL_RECORD_POLICY.LIFEINS_DEPT_K02
  COMPONENT SLC_LIFEINS_ORG 'K02'

CREATE SECURITY LABEL MEDICAL_RECORD_POLICY.LIFEINS_DEPT_S01
  COMPONENT SLC_LIFEINS_ORG 'S01'

CREATE SECURITY LABEL MEDICAL_RECORD_POLICY.LIFEINS_DEPT_S02
  COMPONENT SLC_LIFEINS_ORG 'S02'
```

The security label for the medical analyst is created with the following command:

```
CREATE SECURITY LABEL MEDICAL_RECORD_POLICY.MEDICAL_ANALYST
  COMPONENT SLC_LEVEL 'CONFIDENTIAL',
  COMPONENT SLC_LIFEINS_ORG 'K01', 'K02', 'S01', 'S02'
```

Step 2: Creating the protected SALES table

Privilege and authority requirements

ALTER table privilege on the MEDICAL_RECORD table.

A database administrator should be granted a security label that is associated with the MEDICAL_RECORD_POLICY security policy. Since administration activities may require working with most rows and columns, you might consider granting the highest security label: MEDICAL_ANALYST. Initially the administrator's security label will be used as the default value for the new DEPARTMENT_TAG column (see Step 3).

```
GRANT SECURITY LABEL MEDICAL_RECORD_POLICY.MEDICAL_ANALYST
  TO USER <administrator_auth_id>
  FOR ALL ACCESS
```

To secure the MEDICAL_RECORD table, all confidential columns need to be secured with the MEDICAL_RECORD_POLICY.MED_RECORD security label. To secure the rows, a new column will be added to hold the row security label. The table can be protected by executing the following command:

```
ALTER TABLE MEDICAL_RECORD
  ALTER COLUMN MEDICAL_HISTORY
    SECURED WITH MEDICAL_RECORD_POLICY.MED_RECORD
  ADD COLUMN DEPARTMENT_TAG DB2SECURITYLABEL
  ADD SECURITY POLICY MEDICAL_RECORD_POLICY
```

Upon successful execution of the command, the MEDICAL_RECORD table is protected.

Table 14. Definition for the protected MEDICAL_RECORD table

Column name	Type schema	Type name	Length	Scale	Nulls
RECORDID	SYSIBM	CHARACTER	6	0	No
CLIENTNO	SYSIBM	CHARACTER	6	0	No

DEPTNO	SYSIBM	CHARACTER	6	0	No
APPLICATION_	SYSIBM	DATE	4	0	Yes
DATE					
LAST_	SYSIBM	DATE	4	0	No
UPDATE					
MEDICAL_	SYSIBM	CLOB	1048576	0	No
HISTORY					
RISK_	SYSIBM	SMALLINT	1	0	No
INDEX					
DEPARTMENT_	SYSIBM	DB2SECURITY	0	0	No
TAG		LABEL			
The shading shows the protected MEDICAL_HISTORY column and the added DEPARTMENT_TAG column.					

Step 3: Updating the security label column

Privilege and authority requirements

UPDATE privilege on the MEDICAL_RECORD table:

The database administrator can either be granted a security label that can be used for updates (you did this in Step 2), or you could grant an EXEMPTION for WRITE access.

```
GRANT EXEMPTION ON RULE DB2LBACWRITETREE
FOR MEDICAL_RECORD_POLICY
TO USER <administrator_auth_id>
```

Initially the DEPARTMENT_TAG column will be populated with the administrator's security label (MEDICAL_RECORD_POLICY.MEDICAL_ANALYST), when the table is altered. Next that column needs to be updated with the appropriate security label for each record using the following commands:

```
UPDATE MEDICAL_RECORD
  set DEPARTMENT_TAG= SECLABEL_BY_NAME ('MEDICAL_RECORD_POLICY', 'DEPT_K01')
  where DEPTNO='K01'

UPDATE MEDICAL_RECORD
  set DEPARTMENT_TAG= SECLABEL_BY_NAME ('MEDICAL_RECORD_POLICY', 'DEPT_K02')
  where DEPTNO='K02'

UPDATE MEDICAL_RECORD
  set DEPARTMENT_TAG= SECLABEL_BY_NAME ('MEDICAL_RECORD_POLICY', 'DEPT_S01')
  where DEPTNO='S01'

UPDATE MEDICAL_RECORD
  set DEPARTMENT_TAG= SECLABEL_BY_NAME ('MEDICAL_RECORD_POLICY', 'DEPT_S02')
  where DEPTNO='S02'
```

The updated MEDICAL_RECORD table would look like this.

Table 15. Updated MEDICAL_RECORD table

RECORDID	CLIENTNO	DEPTNO	ALLOCATION	LAST_	MEDICAL_	RISK_	Row
			DATE	UPDATE	HISTORY	FACTOR	Security

							Label
000010	K108341	K01	2005/01/05	2005/01/15	...	0	MEDICAL_RECORD_POLICY.DEPT_K01
000020	K181245	K01	2005/02/09	2005/02/19	...	2	MEDICAL_RECORD_POLICY.DEPT_K01
000030	S245987	S02	2005/02/11	2005/02/21	...	1	MEDICAL_RECORD_POLICY.DEPT_S02
000050	S231674	S02	2005/03/23	2005/04/04	...	1	MEDICAL_RECORD_POLICY.DEPT_S02

Step 4: Granting the security labels to users

Privilege and authority requirements

This requires SECADM authority to execute commands for granting security labels to users.

After the MEDICAL_RECORD table has been protected, no user can access the table until security labels are granted to them. To allow Peter, Andrea, and Joseph access to the table, grant them security labels with the following commands:

```
GRANT SECURITY LABEL MEDICAL_RECORD_POLICY. MEDICAL_ANALYST
TO USER PETER
FOR ALL ACCESS

GRANT SECURITY LABEL MEDICAL_RECORD_POLICY.DEPT_K01
TO USER Andrea
FOR ALL ACCESS

GRANT SECURITY LABEL MEDICAL_RECORD_POLICY.DEPT_S02
TO USER Joseph
for ALL ACCESS
```

The next section shows how well your LBAC security solution works.

Watching the solution in action

This section explains some possible runtime scenarios after the MEDICAL_RECORD table has been protected.

Example 1

Peter, the medical records analyst, tries to read the MEDICAL_RECORD table by issuing:

```
SELECT * from INSURANCE.MEDICAL_RECORD;
```

Command executes successfully. All records of the MEDICAL_RECORD are returned.

Example 2

Andrea, the manager of department K01, tries to read the MEDICAL_RECORD table by issuing:

```
SELECT RECORDID, CLIENTNO, DEPTNO, APPLICATION_DATE, LAST_UPDATE, MEDICAL_HISTORY, RISK_INDEX
from INSURANCE.MEDICAL_RECORD;
```

The command execution fails because it violates the READ access rule on the protected MEDICAL_HISTORY column.

Example 3

Joseph, the manager of department S02, tries to manipulate a record in the MEDICAL_RECORD table. First he queries for the record:

```
SELECT RISK_INDEX from INSURANCE.MEDICAL_RECORD
WHERE CLIENTNO='S231674';
```

Command executes successfully. The value of RISK_INDEX is returned.

Joseph then tries to update the value of the RISK_INDEX.

```
UPDATE INSURANCE.MEDICAL_RECORD
SET RISK_INDEX = 0
WHERE CLIENTNO='S231674'
```

The command fails because it violates the write access rule. The same command executed by Peter would be successful since Peter has write access to the records.

Example 4

Joseph tries to update one of the confidential columns in the MEDICAL_RECORD table.

```
UPDATE INSURANCE.MEDICAL_RECORD
SET MEDICAL_HISTORY = ''
WHERE CLIENTNO='S231674'
```

The command fails because it violates the access rule for the protected column.

Conclusion

This completes Part 1 of the LBAC tutorial. You should now be familiar with the basics of row and column protection and be able to design a LBAC solution to protect your data. In Part 2, you will work through more complex scenarios and learn how to apply exemptions to your security controls.

Resources

Learn

- Refer to the [LBAC section of the DB2 Information Center](#) for more information on this topic.
- The article [What's new in DB2 Viper](#) introduces you to many of DB2 Viper's features.
- Visit the [developerWorks DB2 for Linux, UNIX, and Windows page](#) to expand your DB2 skills.
- Visit the [developer Information management zone](#) to expand skills on all the IBM Information Management products.

Get products and technologies

- Download the test drive version of [DB2 Viper](#) to try out the concepts in this tutorial.

Discuss

- [Participate in the discussion forum for this content.](#)

About the authors

Carmen K. Wong

Carmen Wong has worked in the DB2 Administration Tools Team as a software developer for five years. Her experiences include designing and implementing Java GUI using Swing for DB2 Administration Tools, specializing in monitoring tools (Visual Explain and Event Monitor). She was also involved in the LBAC project in the DB2 Viper release. Carmen is the co-author of a DB2 security book, which will be released in fall 2006.

Stan Musker

Stan Musker has worked as an Information Developer for 18 years, the last 8 years in Information Management. He currently leads the team that creates the documentation for the DB2 Administration Tools. In addition, he has helped develop product videos, tutorials, and eBooks.