DBA Certification Course

(Summer 2008)

Chapter 2: Security in DB2

- Authentication
- DB2 Authorities
- Privileges
- Label-Based Access Control

Objectives

After completing this chapter, you should be able to:

- explain where authentication is done (client or server), depending on configuration parameters.
- explain the rights of different server authorizations
- use `GRANT` and `REVOKE` for object privileges

  This is mostly a short repetition from "Databases I".

Literature

- Graham G. Milne: DB2 9 Fundamentals exam 730 prep, Part 2: Security

  [http://www.ibm.com/developerworks/edu/dm-dw-db2-cert7302.html]
- Carmen K. Wong, Stan Musker: DB2 Label-Based Access Control, a practical guide, Part 1

  [http://.../developerworks/edu/dm-dw-dm-0605wong-i.html]
- Clara Liu, Raul Chong, Dwaine Snow, Sylvia Qi: Understanding DB2: Learning Visually with Examples

  IBM Press/Pearson, 2005, ISBN 0-13-185916-1, 895 pages.

Overview

Security Components (1)

- There are two main security components:
  - ◇ Authentication: Checking the identity of the user (is he really the person he claims to be)?
  - ◇ Authorization: Determining whether a user is allowed to perform the operation he wants to.
- Authentication is done externally to DB2, by:
  - ◇ the operating system or
  - ◇ a network security service like Kerberos or
  - ◇ a GSS-API plugin.
- Thus, passwords are not stored inside DB2.

Security Components (2)

- Authorization is done via
  - ◇ DB2 authorities / authority levels (for admins): `SYSADM`, `SYSCTRL`, `SYSMAINT`, `DBADM`, `LOAD`, ...
  - ◇ database privileges, e.g. `CONNECT`, `CREATETAB`.
  - ◇ object privileges, e.g. `GRANT SELECT, INSERT ON MY_TABLE TO BRASS`
  - ◇ Label-based access control (LBAC).

    This is new in Version 9.
- From top to bottom, these mechanisms permit a more fine-granular decision about access rights.

Overview

Authentication (1)

- A typical case is that the authentication is done on the server (by the operating system).
- When the user is logged in on the server, he/she can connect to the database without password:

```
connect to sample
```

- If the application program runs on a different machine (client), and a network connection to the server is used, it might be necessary to specify user and password for the server:

```
connect to sample user scott using tiger
```

Authentication (2)

- Several configuration parameters determine:
  - ◇ Where authentication is done (client or server),
  - ◇ with what mechanism (OS, Kerberos, ...), and
  - ◇ whether the communication is encrypted or not.
- An important parameter is `AUTHENTICATION`.
  - ◇ The current value appears in the output of

    ```
    get dbm cfg
    ```
  - ◇ It can be changed with

    ```
    update dbm cfg using authentication KERBEROS
    ```


Authentication (3)

- Possible values of the DBM configuration parameter `AUTHENTICATION` are (continued on next slide):
  - ◇ `SERVER`: OS on the server (default).
  - ◇ `SERVER_ENCRYPT`: OS on the server, password is sent encrypted over the net.
  - ◇ `DATA_ENCRYPT`: As `SERVER_ENCRYPT`, but also data are sent encrypted over the net.
  - ◇ `DATA_ENCRYPT_CMP`: As `DATA_ENCRYPT`, but older clients may connect with `SERVER_ENCRYPT`.

    `DATA_ENCRYPT` was new in Version 9.

Authentication (4)

- Parameter `AUTHENTICATION`, continued:
  - ◇ `CLIENT`: OS on the client (with exceptions).

    See configuration parameters `TRUST_ALLCLNTS`, `TRUST_CLNTAUTH`.
  - ◇ `KERBEROS`: Kerberos server.
  - ◇ `KRB_SERVER_ENCRYPT`: If possible, Kerberos server, otherwise as `SERVER_ENCRYPT`.

    It might be that a specific client does not support Kerberos.
  - ◇ `GSSPLUGIN`: External procedures via GSS-API.
  - ◇ `GSS_SERVER_ENCRYPT`: As `GSSPLUGIN`, if client supports it, otherwise as `SERVER_ENCRYPT`.

Authentication (5)

- The parameter `AUTHENTICATION` is set at two places:
  - ◇ on the server (with `update dbm cfg`)
  - ◇ on the client (with `catalog database`).
- In general, the two settings must be identical, with exception of the modes that permit alternatives. Then one of the alternatives must be chosen on the client.

  Possible settings on the client are: `SERVER`, `SERVER_ENCRYPT`, `CLIENT`, `KERBEROS` (with `TARGET PRINCIPAL` ⟨Name⟩), `SQL_AUTHENTICATION_DATAENC`, `SQL_AUTHENTICATION_DATAENC_CMP`, `GSSPLUGIN`.

Authentication (6)

- Suppose `AUTHENTICATION` is set to `CLIENT`.
- There are
  - ◇ trusted clients (e.g., UNIX, Windows 2000),

    A special case of trusted clients are host clients (e.g., z/OS).
  - ◇ untrusted clients (e.g. Windows 98): without reliable security features.

    Windows 98/ME is no longer supported in V9, but V8-clients can connect to a V9 server.
- `TRUST_ALLCLNTS` determines whether a client is really allowed to perform authentication (only username is sent to server) or not (password is sent, too).

Authentication (7)

- The parameter `TRUST_ALLCLNTS` can be:
  - ◇ `YES`: Even the authentication from a Windows 98 client is trusted.

    This is the default!
  - ◇ `NO`: Only the authentication of trusted clients is accepted.
  - ◇ `DRDAONLY`: Only the authentication from host clients is accepted (client running on mainframe).

    The client must run on iSeries, zSeries, VM, VSE. The name is for historical reasons only. As of Version 8, all clients connect to the server using the DRDA database communication protocol ("distributed relational database architecture").

Authentication (8)

- Suppose a user is logged in at the client as user A, but wants to connect to the server as user B.
- Then one specifies user ID and password in the connect statement.
- The parameter `TRUST_CLNTAUTH` determines where authentication is done in this case. It can be:
  - ◇ `SERVER`: On the server.
  - ◇ `CLIENT`: On the client.

    Note that this is done only if `AUTHENTICATION` is set to `CLIENT`, and the client is trusted (or `TRUST_ALLCLNTS=YES`). I.e. the other parameters have higher priority.

Authentication (9)

- One must distinguish between
  - ◇ connections to a database, and
  - ◇ a "login" (attachment) to the instance.

    This can happen implicitly, e.g. with "`get dbm cfg`", "`create db`", or explicitly with `attach to` ⟨`node`⟩, e.g. `attach to db2`.
- The parameter `AUTHENTICATION` is actually the setting for instance attachment.
- The parameter `SRVCON_AUTH` is responsible for database connection (probably new in Version 8.2).
- However, `SRVCON_AUTH` is by default `NOT_SPECIFIED`, in which case the value of `AUTHENTICATION` is used.

Overview

DB2 Authorities (1)

- DB2 Authorities are rights that usually only administrators have. One must distinguish between:
  - ◇ Instance-level authorities:
    `SYSADM`, `SYSCTRL`, `SYSMAINT`, `SYSMON`.
  - ◇ Database-level authorities for a particular DB:
    `DBADM`, `LOAD`, `SECADM`.
- Furthermore, there are database-level privileges, which are sometimes also listed as authorities.

  But these are not only for administrators: `CONNECT`, `CREATETAB`, `BINDADD`, `CREATE_EXTERNAL_ROUTINE`, `CREATE_NOT_FENCED`, `IMPLICIT_SCHEMA`, `LOAD`, `QUIESCE_CONNECT`.

DB2 Authorities (2)

- Authorizations of the current user are printed with

  `get authorizations`
- Instance-level authorities can only be assigned to a group by changing DBM CFG parameters, e.g.

  `update dbm cfg using SYSADM_GROUP db2grp1`

  A change is effective only after `db2stop/db2start`.

  Since the instance is not running for certain instance-level commands (like `db2start`), `GRANT` cannot be used (needs data dictionary).
- Database level authorities are given with the `GRANT` command to a user or group.

DB2 Authorities (3)

- `SYSADM` is the most powerful authority:
  - ◇ Only user that can change DBM CFG.

    Who can change `SYSADM_GROUP`, could get this authority.
  - ◇ Can execute any command to the instance.
  - ◇ Can execute any command to any database.
  - ◇ Can read all data.
- On UNIX/Linux, by default, the primary group of the instance owner has all `SYS*`-authorities.

  On Windows, by default the `SYS*GROUP` parameters are set to null, and the Windows administrators group has these authorities.

DB2 Authorities (4)

- `SYSCTRL`:
  - Can create/drop databases, create/drop tablespaces, change DB CFG.
  - Can perform standard maintenance commands, e.g. start/stop instance, force users off the database, perform backup&recovery, use runstats, obtain monitor snapshots.
  - Cannot change or look at table data.

    Unless, of course, he/she was explicitly granted the corresponding privilege for the specific table.

DB2 Authorities (5)

- `SYSMAINT`:
  - Cannot create/drop databases or tablespaces, but can change DB CFG.
  - Can perform standard maintenance commands (except forcing users off the database).
- `SYSMON`:
  - Can use monitor switches or snapshot data.

    The following commands are allowed: `GET DATABASE MANAGER MONITOR SWITCHES`, `GET MONITOR SWITCHES`, `UPDATE MONITOR SWITCHES`, `RESET MONITOR`, `GET SNAPSHOT`, `LIST ACTIVE DATABASES`, `LIST APPLICATIONS`, `LIST DCS APPLICATIONS`.

DB2 Authorities (6)

- `DBADM`:
  - Has all privileges to all tables in the DB.
  - Can grant/revoke any privilege for a DB object
  - Can drop any table.
  - Cannot change the DB CFG.
  - Cannot create/drop a tablespace.
  - Cannot perform backup&recovery.
- The user who created a database gets `DBADM` for it.
- `SYSADM` can perform (after connecting to the DB):

  `grant dbadm for database to user ⟨username⟩`

DB2 Authorities (7)

- `LOAD`:
  - Can load data into a table of the database.

    Only if he/she has `insert/delete` privileges for the table. Not every user can perform `LOAD`, because table access is restricted during the load. It may also switch off constraints (?).
  - Can use `runstats` on any table.

    After the load, statistics for the optimizer should be updated.
- `SECADM`:
  - Can configure label-based access control.
  - Not even `SYSADM` can do this.

    But `SYSADM` can grant the `SECADM` right.

Database Privileges (1)

- `CONNECT`: Right to log into the database.
- `QUIESCE_CONNECT`: Login during maintenance.
- `CREATETAB`: Right to create tables.

  No right is required for view creation.
- `BINDADD`: Right to create packages.
- `IMPLICIT_SCHEMA`: Create tables in new schemas.
- `CREATE_NOT_FENCED`: Extend DBMS (add functions).
- `CREATE_EXTERNAL_ROUTINE`: Create procedure for use in applications or by other users.

Database Privileges (2)

`CONNECT`:

- An operating system user can connect to a database if he/she has the "`CONNECT`" authority.
- E.g. an administrator can "create" a user in the database by issuing this command:

  ```
  GRANT CONNECT ON DATABASE TO BRASS
  ```
- This right can also be granted to "`PUBLIC`", in which case every OS user can connect to the database.

  Actually, when a database is created, `CONNECT` is granted automatically to `PUBLIC`. One must first revoke it to grant it selectively.

Database Privileges (3)

`BINDADD`:

- When an application program with embedded SQL is compiled, a "package" is created in the database with the SQL statements (and access plans for them) of the application program.
- This is done with the command "`BIND`".
- `BINDADD` is the right to create new packages.

Database Privileges (4)

`IMPLICIT_SCHEMA`:

- Suppose a user writes

  ```
  CREATE TABLE X.Y (...)
  ```

  where a schema `X` does not exist yet.
- If the user has the `IMPLICIT_SCHEMA` privilege, a new schema `X` is created with owner `SYSIBM` and `PUBLIC` being allowed to create objects in this schema.

  Without the `IMPLICIT_SCHEMA` privilege, this gives an error.
- There is an explicit `CREATE SCHEMA` statement (needs `SYSADM` or `DBADM` authority).

Implicit Grants

- When a database is created, some privileges are automatically/implicitly granted to PUBLIC: CONNECT, IMPLICIT_SCHEMA, and SELECT on the system catalog.
- One can use REVOKE to remove them from public, and then give them with GRANT to specific users.
- The user who created a database gets the following privileges automatically: DBADM, CONNECT, CREATETAB, BINDADD, IMPLICIT_SCHEMA, CREATE_NOT_FENCED.

  When one revokes DBADM, he still has the other privileges.

Overview

GRANT Command (1)

- In SQL, access rights on database objects (tables, views, etc.) can be given to other users by means of the GRANT command.
- The GRANT command was already contained in the SQL-86 standard. It has the form

  GRANT ⟨Rights⟩ ON ⟨Object⟩ TO ⟨Users⟩
- E.g. give read and insert rights ("privileges") on the table COURSES to the users BRASS and SPRING.

```
GRANT SELECT, INSERT ON COURSES TO BRASS, SPRING
```


GRANT Command (2)

- This will give the users BRASS and SPRING read access to the table COURSES and the possibility to append data (add new rows).
- They will not be able to delete or modify rows in the table (unless they had these rights before).
- It is possible to later GRANT them the UPDATE and DELETE rights, too.

  Then SELECT and INSERT do not have to be repeated.
- The DBMS probably stores in a system table the user-command-object triples.

GRANT Command (3)

DB2-Specific Syntax:

- One can explicitly distinguish between granting a right to a user or a group:

  `GRANT SELECT ON X TO USER BRASS, GROUP STAFF`

  In the operating system, there might be a user and a group with the same name.

- One can explicitly state the type of the object:

  `GRANT SELECT ON TABLE X TO BRASS`

  The type "`TABLE`" includes views and nicknames. But e.g. for schemas, the keyword `SCHEMA` is required.

GRANT Command (4)

Possible Rights ("Object Privileges"):

- `SELECT`: Read access (use of the table in queries).
- `INSERT`: Appending new data (insertion of rows).
- `UPDATE`: Changing column values of existing rows.
- `UPDATE`($A_1, \ldots, A_n$): Only data in the columns A_i may be changed.
- `DELETE`: Deleting data (rows from the table).
- ... (continued on next two slides)

GRANT Command (5)

DB2 Object Privileges, continued:

- `REFERENCES`: Creating integrity constraints which reference this table.

  Referencing someone else's table in a foreign key constraint and not giving him/her `DELETE` rights on the new table can in effect limit his/her `DELETE` rights on his/her own table. Also `REFERENCES` allows checking whether a key value is there or not.

- `REFERENCES`($A_1, \ldots, A_n$): Only the A_i may be referenced.
- `ALTER`: Right to change the table definition.
- `INDEX`: Right to create an index on this table.

GRANT Command (6)

Rights for Procedures/Packages:

- `EXECUTE`: Right to execute the procedure etc.
- `BIND`: Reoptimize SQL statements in a package.

Rights for Schema Objects:

- `ALTERIN`: Alter any object in the schema.
- `CREATEIN`: Create objects in the schema.
- `DROPIN`: Drop any object in the schema.

GRANT Command (7)

`GRANT ALL PRIVILEGES`:

- Instead of listing single rights it is possible to say

  `GRANT ALL PRIVILEGES ON ⟨Object⟩ TO ⟨Users⟩`

- This means all rights which the user executing the `GRANT` has on the object.

`TO PUBLIC`:

- Instead of listing all users in the system, the following is possible (this includes future users):

  `GRANT SELECT ON COURSES TO PUBLIC`

GRANT Command (8)

`WITH GRANT OPTION`:

- A user can be given a right with the possibility to pass on the right to other users.
- To do this, add the clause "`WITH GRANT OPTION`" to the `GRANT`-command:

```
GRANT SELECT ON COURSES TO BRASS
WITH GRANT OPTION
```

- This allows the user `BRASS` also to pass the grant option to other users (who then can also pass the right to other users, etc.).

GRANT Command (9)

Owner of an Object:

- The owner of a database object (table etc.), i.e. the user who created the object, has all rights on it including the grant option for these rights.
- By default, only the owner has rights on an object. Other users can get rights only by explicit `GRANT`s.

  Users with system administrator rights might be able to access the object without being granted rights explicitly.

- The owner of an object can also drop the object again (which is not included in the other rights).

GRANT Command (10)

`CONTROL`-Right in DB2:

- This gives all rights on the object with grant option. It also gives the right to drop the object.

  There is no `CONTROL`-right in Oracle and SQL Server. But it is similar to being the owner of the object.

- By default, the object creator has the `CONTROL` right.

  For views, the creator gets the `CONTROL` right only if he/she holds it also for the underlying base tables (and used views).

- But the `CONTROL`-right is always given without `GRANT OPTION`. It can be granted only by an administrator.

  Granting `CONTROL` needs the `SYSADM` or `DBADM` authority.

Revoking Access Rights (1)

- In DB2, only a user with CONTROL-right for a table can revoke (take back, remove) access rights:

  REVOKE ⟨Rights⟩ ON ⟨Object⟩ FROM ⟨Users⟩

- The syntax of the components ⟨Rights⟩, ⟨Object⟩, ⟨Users⟩ is identical to the GRANT statement. E.g.:

  REVOKE INSERT ON COURSES FROM BRASS

- If BRASS had SELECT and INSERT rights on COURSES before this command, he will have only the SELECT right afterwards.

Revoking Access Rights (2)

- After the REVOKE the rights are gone (unless the user has the right also via a group or PUBLIC).
- This does not conform to the SQL standard. There a user might get the same right via different paths (granted from different users), and keeps the right if a path from the owner still exits.
- If the right was given WITH GRANT OPTION, the user might have given the right to other users. These keep the right in DB2.

  In the Standard, they would lose it if all paths are broken.

Revoking Access Rights (3)

- It is also strange in DB2 that users who get a right WITH GRANT OPTION, can give the right to other users, but cannot revoke it later.

  This makes the implementation simpler: The system has to remember only triples (who has what right on which table), not quadrupels (and who has given the right).

- Note that it is not possible to grant a right to PUBLIC and revoke it from a specific user.
- When a table is deleted, all GRANTs for it are deleted, too. If it is later re-created, only the owner can access it.

Groups in DB2 (1)

- DB2 has no roles, but it understands the concept of groups of the underlying operating system.

  E.g. UNIX and Windows NT have groups of users.

- Privileges can be granted not only to users, but also to groups.

  If one revokes the right from a user, he still might have it via a group.

- So there are three ways a user might get a privilege:
  - ◇ It can be granted to this user personally,
  - ◇ to a group in which he/she is member,
  - ◇ or to PUBLIC.

Groups in DB2 (2)

- In DB2, table privileges granted to groups apply only to statements that are dynamically prepared.

  I.e. compiled and immediately executed. If, in contrast, an execution plan is kept in the database that was valid when it was created, DB2 must ensure that it is still valid when it is executed later. If a right is revoked inside the database, DB2 can mark access plans as invalid that depend on this right. However, if a user is removed from a group on the operating system level, DB2 is not informed about this.

- Privileges granted to groups are not used when
  - ◇ binding a package (static embedded SQL),
  - ◇ checking rights on the base table(s) while processing a `CREATE VIEW` statement (or MQT def.).

Overview

LBAC: Concepts (1)

- Label-Based Access Control (LBAC) is new in DB2 Version 9 (“Viper”).
- Security labels can be attached to rows or columns.
- Security labels can be granted to users.
- A user can only see those rows for which his security label is high enough.

  This works like an implicit condition attached to the `WHERE`-clause.

- A user gets an error if he tries to access columns for which his security label is not high enough.

LBAC: Concepts (2)

- A security label consists of one or more security label components.

  The different components correspond to different criteria applied for deciding whether a user can access a piece of data (e.g., rank in the company, membership in a project). One can also use one component for restricting row access and one for restricting column access.

- A security label component can have one of three types:
  - ◇ Set: Different values without hierarchy.
  - ◇ Arrays: Different values with linear hierarchy.
  - ◇ Trees: Different values with tree herarchy.

LBAC: Concepts (3)

- Labels can contain several elements for a component of type Set or Tree.

  They can contain at most one element for a component of type Array. Because of the linear hierarchy, only the strongest component must be stored.

- Label components can be empty.
  - ◇ An empty component in a data label does not restrict the access in any way.
  - ◇ An empty component in a user label permits access only to data in which this component is also empty.

LBAC: Concepts (4)

- Suppose that data is protected by label D and the user holds label U. For each component C the following must hold:
  - ◇ If C of type Set: $D_C \subseteq U_C$.
  - ◇ If C is of type Tree: For each element $X \in D_C$, there is an element $Y \in U_C$ such that Y is ancestor of or equal to X.
  - ◇ If C is of type Array: U_C must come before D_C in the list/array (or $U_C = D_C$). For write access, $U_C = D_C$ must hold (no write-up or write-down).

LBAC: Concepts (5)

- A `security policy` defines a set of label components and some options.
- Each table can have only one security policy.
- There is a data type `DB2SECURITYLABEL` for security labels.

  Values of this type can be generated by calling the function `SECLABEL` with a string representation of the label, e.g. `SECLABEL('SECRET')`. If the label has several components, these must be listed in the same order as in the definition of the security policy, separated by colons. If a component has a set of values, this must be included in `(...)`, and the elements separated by commas.
  Defined (named) labels can be written as `SECLABEL_BY_NAME('P', 'L')`, where `P` is the policy and `L` is the label name.

LBAC: Concepts (6)

- Security labels can be granted for `READ` access, or for `ALL` access.

  I.e. the information about the allowed access type is not included in the security label. The security label only selects rows and columns.

- The standard object privileges are still in force.

  E.g., if a use is not granted the `SELECT` privilege on a table, he/she cannot read it, even if the security label would permit that.

- A user can hold labels for different security policies. For ia given policy, he/she can hold at most one label for read access and one label for write access.

LBAC: Concepts (7)

- When a table is accessed via a view, the security label of the current user applies.

  This is different from the standard object privileges, where (normally) the access rights of the user who defined the view are applied for the access to the base table.

- LBAC rules are not applied when checking the correctness of foreign keys.

  They are applied when a delete cascades.

- LBAC rules are not applied for backups.

LBAC: Concepts (8)

- Only the security administrator of the database (`SECADM`) can define security policies and labels.

  Only `SYSADM` can grant `SECADM`. The database administrator `DBADM` does not automatically have `SECADM`.

- It is possible to define exemptions for rules for the comparison of security labels.

LBAC: Example (1)

- Consider a simple employee table:

  `EMP(EMPNO, ENAME, JOB, SAL, DEPTNO)`

- There are three departments (`DEPTNO` can have values 10, 20, 30), and a central administration.
- Suppose that employees in a department may see the data of other employees in their department only, except the salary.
- The human resources manager of a department may read and write all data of the employees of his/her department (including the salary).

LBAC: Example (2)

- Central administration may see data of all employees, but there are also persons who may not see the salary (i.e. the two restrictions are orthogonal).

LBAC: Example (3)

- For access to rows, a security label component is defined that has a tree hierarchy:

```
CREATE SECURITY LABEL COMPONENT DEPT_RESTRICTION
TREE { 'TOP_MANAGEMENT' ROOT,
       'DEPT10' UNDER 'TOP_MANAGEMENT',
       'DEPT20' UNDER 'TOP_MANAGEMENT',
       'DEPT30' UNDER 'TOP_MANAGEMENT'
}
```

- For the salary, a single value suffices (that can be included or not included in a security label).

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


LBAC: Example (4)

- Now one has to define a security policy:

```
CREATE SECURITY SECURITY POLICY EMP_POLICY
    COMPONENTS DEPT_RESTRICTION, SAL_RESTRICTION
    WITH DB2LBACRULES
    RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL
```

`WITH DB2LBACRULES` is currently the only option. It is a set of rules for the comparison of security label components. One can define exemptions to these rules.

"`RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL`" means that one gets an error if one tries to insert or update a security label for which one has no write permission. The alternative, which is also the default, is "`OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL`" which means that one's own security label is silently substituted in this case.

LBAC: Example (5)

- Now one can define and grant security labels:

```
CREATE SECURITY LABEL EMP_POLICY.DEPT10_HR
    COMPONENT DEPT_RESTRICTION 'DEPT10',
    COMPONENT SAL_RESTRICTION 'CONFIDENTIAL'
```

- Note that the name of a security label is qualified with the name of the security policy.
- E.g. if `JONES` is the human resources manager of Department 10, one can grant this security label:

```
GRANT EMP_POLICY.DEPT10_HR TO JONES
      FOR ALL ACCESS
```


LBAC: Example (6)

- One needs a label for the salary column:

```
CREATE SECURITY LABEL EMP_POLICY.SAL_LABEL
    COMPONENT SAL_RESTRICTION 'CONFIDENTIAL'
```

- The table definition looks as follows:

```
CREATE TABLE EMP (
  COLUMN EMPNO NUMERIC(4) NOT NULL PRIMARY KEY,
  ....
  SAL NUMERIC(6) NOT NULL SECURED WITH SAL_LABEL,
  EMP_TAG DB2SECURITYLABEL NOT NULL)
  SECURITY POLICY EMP_POLICY
```

LBAC: Example (7)

- One can insert data e.g. with

```
INSERT INTO EMP VALUES
  (1000, ...,
    SECLABEL_BY_NAME(EMP_POLICY, DEPT10_LABEL))
```

- One can view security labels e.g. with

```
SELECT ...,
    VARCHAR(
        SECLABEL_TO_CHAR('EMP_POLICY',EMP_TAG),
        30)
FROM EMP
```


LBAC: Evaluation (1)

- It seems that the same effect could be reached with views (one view for each security label).
- However, the view definitions might be more complicated.
- Moreover, the security labels might correspond more to concepts of business people.
- Note that content based restrictions need constraints: One cannot use the department number itself as label.

LBAC: Evaluation (2)

- The syntax for the common case that labels have only one component is unnecessarily complicated (one still must define labels and components separately).