

---

## Security

**E**ight and one-half percent (8.5%) of the DB2 9 for Linux, UNIX, and Windows Database Administration exam (Exam 731) is designed to test your knowledge about the mechanisms DB2 uses to protect data and database objects against unauthorized access and modification. The questions that make up this portion of the exam are intended to evaluate the following:

- Your ability to identify the methods that can be used to restrict access to data stored in a DB2 database
- Your ability to identify the authorization levels used by DB2
- Your ability to identify the privileges used by DB2
- Your ability to identify how specific authorizations and/or privileges are given to a user or group
- Your ability to identify how specific authorizations and/or privileges are taken away from a user or group
- Your knowledge of the mechanisms and steps needed to implement label-based access control (LBAC)

This chapter is designed to introduce you to the various authorizations and privileges that are available with DB2 9 and to the tools that are used to give (grant) one or more of these authorizations and/or privileges to various users and groups. This chapter will also show you how to revoke one or more authorizations or privileges a user or group currently holds and how to implement LBAC to control access columns and rows in a table.

## **Controlling Database Access**

Identity theft—a crime in which someone wrongfully obtains another person's personal data (such as a Social Security number, bank account number, and credit card number) and uses it in some way that involves fraud or deception for economic gain—is the fastest-growing crime in our nation today. Criminals are stealing information by overhearing conversations made on cell phones, by reading faxes and emails, by hacking into computers, by waging telephone and email scams, by stealing wallets and purses, by stealing discarded documents from trash bins, by stealing mail, and by taking advantage of careless online shopping and banking habits. But more frightening is the fact that studies show that up to 70 percent of all identity theft cases are inside jobs—perpetrated by a coworker or an employee of a business you patronize. In these cases, all that is needed is access to your personal data, which can often be found in a company database.

Every database management system must be able to protect data against unauthorized access and modification. DB2 uses a combination of external security services and internal access control mechanisms to perform this vital task. In most cases, three different levels of security are employed: The first level controls access to the instance under which a database was created, the second controls access to the database itself, and the third controls access to the data and data objects that reside within the database.

## **Authentication**

The first security portal most users must pass through on their way to gaining access to a DB2 instance or database is a process known as authentication. The purpose of authentication is to verify that users really are who they say they are. Normally, authentication is performed by an external security facility that is not part of DB2. This security facility may be part of the operating system (as is the case with AIX, Solaris, Linux, HP-UX, Windows 2000/NT, and many others), may be a separate add-on product (for example, Distributed Computing Environment [DCE] Security Services), or may not exist at all (which is the case with Windows 95, Windows 98, and Windows Millennium Edition). If a security facility does exist, it must be presented with two specific items before a user can be authenticated: a unique user ID and a corresponding password. The user ID identifies the user to the security facility, and the password, which is information

known only by the user and the security facility, is used to verify that the user is indeed who he or she claims to be.



.....

Because passwords are a very important tool for authenticating users, you should always require passwords at the operating system level if you want the operating system to perform the authentication for your database. Keep in mind that on most UNIX operating systems, undefined passwords are treated as NULL, and any user who has not been assigned a password will be treated as having a NULL password. From the operating system's perspective, if no password is provided when a user attempts to log on, this will evaluate to being a valid match.

.....

### **Where Does Authentication Take Place?**

Because DB2 can reside in environments composed of multiple clients, gateways, and servers, each of which may be running on a different operating system, deciding where authentication is to take place can be a daunting task. To simplify things, DB2 uses a parameter in each DB2 Database Manager configuration file (the *authentication* parameter) to determine how and where users are authenticated. Such a file is associated with every instance, and the value assigned to this parameter, often referred to as the *authentication type*, is set initially when an instance is created. (On the server side, the authentication type is specified during the instance creation process; on the client side, the authentication type is specified when a remote database is cataloged.) Only one authentication type exists for each instance, and it controls access to that instance, as well as to all databases that fall under that instance's control.

With DB2 9, the following authentication types are available:

**SERVER.** Authentication occurs at the server workstation, using the security facility provided by the server's operating system. (The user ID and password provided by the user wishing to attach to an instance or connect to a database are compared to the user ID and password combinations stored at the server to determine whether the user is permitted to access the instance or database.) By default, this is the authentication type used when an instance is first created.

**SERVER\_ENCRYPT.** Authentication occurs at the server workstation, using the security facility that is provided by the server's operating system. However, the password provided by the user wishing to attach to an instance or connect to a database stored on the server may be encrypted at the client workstation before it is sent to the server workstation for validation.

**CLIENT.** Authentication occurs at the client workstation or database partition where a client application is invoked, using the security facility that is provided by the client's operating system, assuming one is available. If no security facility is available, authentication is handled in a slightly different manner. The user ID and password provided by the user wishing to attach to an instance or connect to a database are compared to the user ID and password combinations stored at the client or node to determine whether the user is permitted to access the instance or the database.

**KERBEROS.** Authentication occurs at the server workstation, using a security facility that supports the Kerberos security protocol. This protocol performs authentication as a third-party service by using conventional cryptography to create a shared secret key. The key becomes the credentials used to verify the identity of the user whenever local or network services are requested; this eliminates the need to pass a user ID and password across the network as ASCII text. (If both the client and the server support the Kerberos security protocol, the user ID and password provided by the user wishing to attach to an instance or connect to a database are encrypted at the client workstation and sent to the server for validation.) It should be noted that the KERBEROS authentication type is supported only on clients and servers that are using the Windows 2000, Windows XP, or Windows .NET operating system. In addition, both client and server workstations must either belong to the same Windows domain or belong to trusted domains.

**KRB\_SERVER\_ENCRYPT.** Authentication occurs at the server workstation, using either the KERBEROS or the SERVER\_ENCRYPT authentication method. If the client's authentication type is set to KERBEROS, authentication is performed at the server using the Kerberos security system. On the other hand, if the client's authentication type is set to anything other than KERBEROS, or if the Kerberos authentication service is unavailable, the

server acts as if the `SERVER_ENCRYPT` authentication type was specified, and the rules of this authentication method apply.

**DATA\_ENCRYPT.** Authentication occurs at the server workstation, using the `SERVER_ENCRYPT` authentication method. In addition, all user data is encrypted before it is passed from client to server and from server to client.

**DATA\_ENCRYPT\_CMP.** Authentication occurs at the server workstation, using the `SERVER_ENCRYPT` authentication method; all user data is encrypted before it is passed from client to server and from server to client. In addition, this authentication type provides compatibility for down-level products that do not support the `DATA_ENCRYPT` authentication type. Such products connect using the `SERVER_ENCRYPT` authentication type, and user data is not encrypted.

**GSSPLUGIN.** Authentication occurs at the server workstation, using a Generic Security Service Application Program Interface (GSS-API) plug-in. If the client's authentication type is not specified, the server returns a list of server-supported plug-ins (found in the *srvcon\_gssplugin\_list* database manager configuration parameter) to the client. The client then selects the first plug-in found in the client plug-in directory from the list. If the client does not support any plug-in in the list, the client is authenticated using the KERBEROS authentication method.

**GSS\_SERVER\_ENCRYPT.** Authentication occurs at the server workstation, using either the GSSPLUGIN or the `SERVER_ENCRYPT` authentication method. That is, if client authentication occurs through a GSS-API plug-in, the client is authenticated using the first client-supported plug-in found in the list of server-supported plug-ins. If the client does not support any of the plug-ins found in the server-supported plug-in list, the client is authenticated using the KERBEROS authentication method. If the client does not support the Kerberos security protocol, the client is authenticated using the `SERVER_ENCRYPT` authentication method.

It is important to note that if the authentication type used by the client workstation encrypts user ID and password information before sending it to a server for authentication (i.e., `SERVER_ENCRYPT`, `KRB_SERVER_ENCRYPT`, etc.), the server must be configured to use a compatible authentication method. Otherwise, it will not be able to process the encrypted data received, and an error will occur.

It is also important to note that if the authentication type is not specified for a client workstation, the SERVER\_ENCRYPT authentication method is used by default. If such a client tries to communicate with a server that does not support the SERVER\_ENCRYPT authentication method, the client will attempt to use the authentication type that is being used by the server—provided the server has been configured to use only one authentication type. If the server supports multiple authentication types, an error will be generated.

### **Security Plug-ins**

In DB2 9, authentication is done using security plug-ins. A security plug-in is a dynamically loadable library that provides authentication security services; DB2 9 supports two mechanisms for plug-in authentication:

- User ID/password authentication

This involves authentication using a user ID and password. The following authentication types are implemented using user ID/password authentication plug-ins:

- CLIENT
- SERVER
- SERVER\_ENCRYPT
- DATA\_ENCRYPT
- DATA\_ENCRYPT\_CMP
- GSS-API authentication

GSS-API was formally known as Generic Security Service Application Program Interface, Version 2 (IETF RFC2743) and Generic Security Service API Version 2: C-Bindings (IETF RFC2744). The following authentication types are implemented using GSS-API authentication plug-ins:

- KERBEROS
- GSSPLUGIN

- KRB\_SERVER\_ENCRYPT
- GSS\_SERVER\_ENCRYPT

KRB\_SERVER\_ENCRYPT and GSS\_SERVER\_ENCRYPT support both GSS-API authentication and user ID/password authentication; however, GSS-API authentication is the authentication type preferred.

Each plug-in can be used independently or in conjunction with one or more of the other plug-ins available. For example, you might specify only a server authentication plug-in to use and allow DB2 to use the defaults for client and group authentication. Alternatively, you might specify only a group or client authentication plug-in. The only situation where both a client and server plug-in are required is for GSS-API authentication. (In some cases—for example, if you are using Microsoft Active Directory to validate a user—you may need to create your own custom security plug-in and make it available for DB2 to use.)

The default behavior for DB2 9 is to use a user ID/password plug-in that implements an operating-system-level mechanism for authentication.

### ***Trusted Clients Versus Untrusted Clients***

If both the server and the client are configured to use the CLIENT authentication type, authentication occurs at the client workstation (if the database is a nonpartitioned database) or at the database partition from which the client application is invoked (if the database is a partitioned database), using the security facility provided by the client workstation's operating system. But what happens if the client workstation is using an operating system that does not contain a tightly integrated security facility, and no separate add-on security facility has been made available? Does such a configuration compromise security? The answer is no. However, in such environments, the DB2 Database Manager for the instance at the server must be able to determine which clients will be responsible for validating users and which clients will be forced to let the server handle user authentication. To make this distinction, clients that use an operating system that contains a tightly integrated security facility (for example, OS/390, VM, VSE, MVS, AS/400, Windows NT, Windows 2000, and all supported versions of UNIX) are classified as trusted clients, whereas clients that use an operating system that does not

provide an integrated security facility (for example, Windows 95, Windows 98, and Windows Millennium Edition) are treated as untrusted clients.

The *trust\_allclnts* parameter of a DB2 Database Manager configuration file helps the DB2 Database Manager for an instance on a server anticipate whether its clients are to be treated as trusted or untrusted. If this configuration parameter is set to YES (which is the default), the DB2 Database Manager assumes that any client that accesses the instance is a trusted client and that some form of authentication will take place at the client. However, if this configuration parameter is set to NO, the DB2 Database Manager assumes that one or more untrusted clients will try to access the server; therefore, all users must be authenticated at the server. (If this configuration parameter is set to DRDAONLY, only MVS, OS/390, VM, VSE, and OS/400 clients will be treated as trusted clients.) It is important to note that, regardless of how the *trust\_allclnts* parameter is set, whenever an untrusted client attempts to access an instance or a database, user authentication always takes place at the server.

In some situations, it may be desirable to authenticate users at the server, even when untrusted clients will not be used. In such situations, the *trust\_clntauth* configuration parameter of a DB2 Database Manager configuration file can be used to control where trusted clients are to be validated. When the default value for this parameter (which is CLIENT) is accepted, authentication for trusted clients will take place at the client workstation. If, however, the value for this parameter is changed to SERVER, authentication for all trusted clients will take place at the server.

## **Authorities and Privileges**

Once a user has been authenticated, and an attachment to an instance or a connection to a database has been established, the DB2 Database Manager evaluates any authorities and privileges that have been assigned to the user to determine what operations the user is allowed to perform. Privileges convey the rights to perform certain actions against specific database resources (such as tables and views). Authorities convey a set of privileges or the right to perform high-level administrative and maintenance/utility operations on an instance or a database. Authorities and privileges can be assigned directly to a user, or they can be obtained indirectly from the authorities and privileges that have been assigned to a group of which the user is a member. Together, authorities and privileges act to control access



to the DB2 Database Manager for an instance, to one or more databases running under that instance's control, and to a particular database's objects. Users can work only with those objects for which they have been given the appropriate authorization—that is, the required authority or privilege. Figure 8–1 provides a hierarchical view of the authorities and privileges that are recognized by DB2 9.

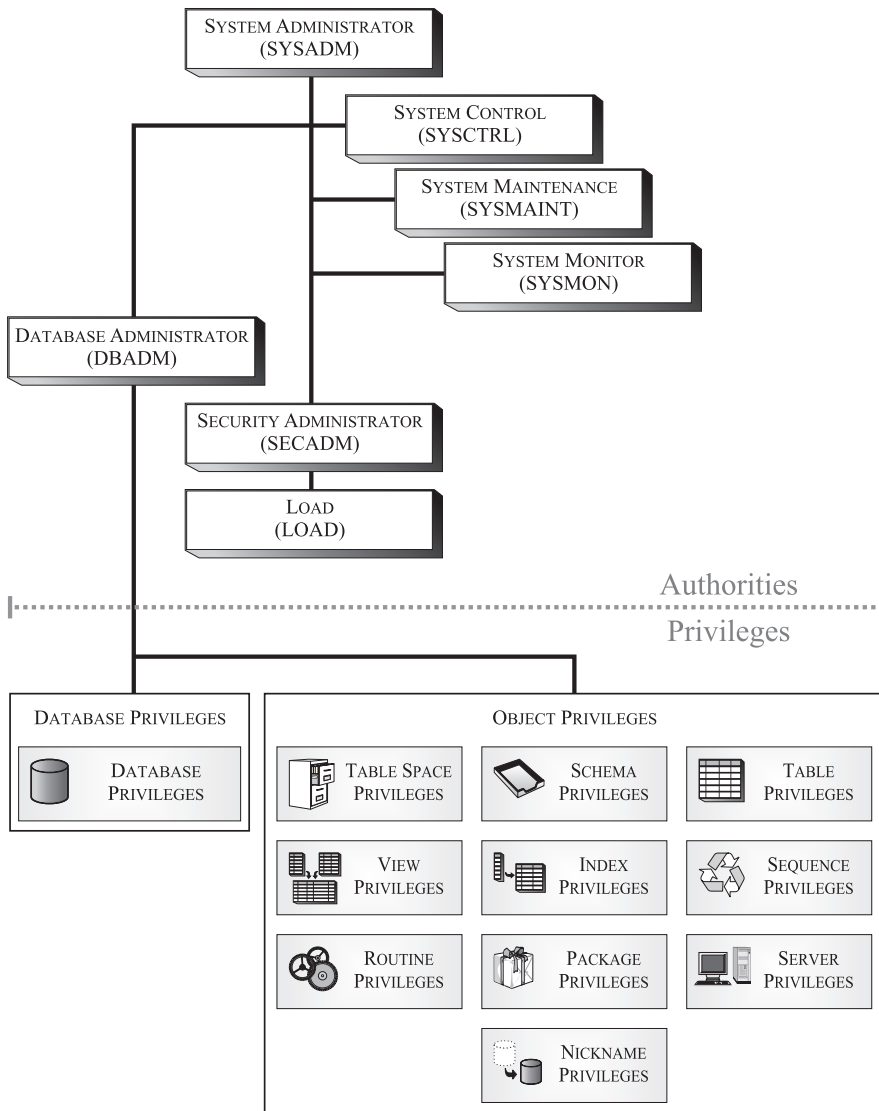


Figure 8–1: Hierarchy of the authorities and privileges available with DB2 9.

## **Authorities**

DB2 9 uses seven different levels of authority to control how users perform administrative and maintenance operations against an instance or a database:

- System Administrator (SYSADM) authority
- System Control (SYSCTRL) authority
- System Maintenance (SYSMAINT) authority
- System Monitor (SYSMON) authority
- Database Administrator (DBADM) authority
- Security Administrator (SECADM) authority
- Load (LOAD) authority

Four of these levels apply to the DB2 Database Manager instance (and to all databases that are under that instance's control), whereas three apply only to specific databases within a particular instance. The instance-level authorities can be assigned only to groups; the names of the groups that are assigned these authorities are stored in the DB2 Database Manager configuration file that is associated with the instance. Conversely, the database-level authorities can be assigned to individual users and, in some cases, groups; groups and users that have been assigned database-level authorities are recorded in the system catalog tables of the database to which the authority applies.

### **System Administrator authority**

System Administrator (SYSADM) authority is the highest level of administrative authority available. Users who have been given this authority are allowed to run any DB2 utility, execute any DB2 command, and perform any SQL/XQuery operation that does not attempt to access data protected by label-based access control (LBAC). Users with this authority also have the ability to control all database objects within an instance, including databases, database partition groups, buffer pools, table spaces, schemas, tables, views, indexes, aliases, servers, data types, functions, procedures, triggers, packages, and event monitors.

Additionally, users who have been given this authority are allowed to perform the following tasks:

- Upgrade (migrate) an existing database from a previous version of DB2 to DB2 Version 9.
- Modify the parameter values of the DB2 Database Manager configuration file associated with an instance—including specifying which groups have System Administrator, System Control, System Maintenance, and System Monitor authority. (The DB2 Database Manager configuration file is used to control the amount of system resources allocated to a single instance.)
- Give (grant) Database Administrator authority and Security Administrator authority to individual users and/or groups.
- Revoke Database Administrator authority and/or Security Administrator authority from individual users and/or groups.

System Administrator authority can be assigned only to a group; this assignment is made by storing the appropriate group name in the *sysadm\_group* parameter of the DB2 Database Manager configuration file associated with an instance. (This is done by executing an UPDATE DATABASE MANAGER CONFIGURATION command with the SYSADM\_GROUP parameter specified, along with the name of the group that is to receive System Administrator authority.) Individual membership in the group itself is controlled through the security facility provided by the operating system used on the workstation where the instance has been defined. Users who possess System Administrator authority are responsible both for controlling the DB2 Database Manager associated with an instance and for ensuring the safety and integrity of the data contained in databases that fall under the instance's control.

Users who hold System Administrator authority are implicitly given the rights granted by System Control, System Maintenance, System Monitor, and Database Administrator authority. However, they are not implicitly given the rights granted by Security Administrator authority.

## System Control authority

System Control (SYSCTRL) authority is the highest level of system or instance control authority available. Users who have been given this authority are allowed to perform maintenance and utility operations both on a DB2 Database Manager instance and on any databases that fall under that instance's control. However, because System Control authority is designed to allow special users to maintain an instance that contains sensitive data that they most likely do not have the right to view or modify, users who are granted this authority do not implicitly receive authority to access the data stored in the databases that are controlled by the instance. On the other hand, because a connection to a database is required in order to perform some of the utility operations available, users who are granted System Control authority for a particular instance also receive the privileges needed to connect to each database under that instance's control.

Users with System Control authority (or higher) are allowed to perform the following tasks:

- Update a database, node, or distributed connection services (DCS) directory (by cataloging/uncataloging databases, nodes, or DCS databases).
- Modify the parameter values in one or more database configuration files. (A database configuration file is used to control the amount of system resources that are allocated to a single database during normal operation.)
- Force users off the system.
- Create or destroy (drop) a database.
- Create, alter, or drop a table space.
- Make a backup image of a database or a table space.
- Restore an existing database using a backup image.
- Restore a table space using a backup image.
- Create a new database from a database backup image.
- Perform a roll-forward recovery operation on a database.
- Start or stop a DB2 Database Manager instance.
- Run a trace on a database operation.

- Take database system monitor snapshots of a DB2 Database Manager instance or any database under the instance's control.
- Query the state of a table space.
- Update recovery history log files.
- Quiesce (restrict access to) a table space.
- Reorganize a table.
- Collect catalog statistics using the RUNSTATS utility.

Like System Administrator authority, System Control authority can be assigned only to a group. This assignment is made by storing the appropriate group name in the *sysctrl\_group* parameter of the DB2 Database Manager configuration file that is associated with a particular instance. (This is done by executing an UPDATE DATABASE MANAGER CONFIGURATION command with the SYSCTRL\_GROUP parameter specified, along with the name of the group that is to receive System Control authority.) Again, individual membership in the group itself is controlled through the security facility that is used on the workstation where the instance has been defined.

### **System Maintenance authority**

System Maintenance (SYSMAINT) authority is the second highest level of system or instance control authority available. Users who have been given this authority are allowed to perform maintenance and utility operations both on a DB2 Database Manager instance and on any databases that fall under that instance's control. System Maintenance authority is designed to allow special users to maintain a database that contains sensitive data that they most likely do not have the right to view or modify. Therefore, users who are granted this authority do not implicitly receive authority to access the data stored in the databases on which they are allowed to perform maintenance. However, because a connection to a database must exist before some utility operations can be performed, users who are granted System Maintenance authority for a particular instance automatically receive the privileges needed to connect to each database that falls under that instance's control.

Users with System Maintenance authority (or higher) are allowed to perform the following tasks:

- Modify the parameter values of one or more DB2 database configuration files
- Make a backup image of a database or a table space
- Restore an existing database using a backup image
- Restore a table space using a backup image
- Perform a roll-forward recovery operation on a database
- Start or stop a DB2 Database Manager instance
- Run a trace on a database operation
- Take database system monitor snapshots of a DB2 Database Manager instance or any database under the instance's control
- Query the state of a table space
- Update recovery log history files
- Quiesce (restrict access to) a table space
- Reorganize a table
- Collect catalog statistics using the RUNSTATS utility

Like System Administrator and System Control authority, System Maintenance authority can be assigned only to a group. This assignment is made by storing the appropriate group name in the *sysmaint\_group* parameter of the DB2 Database Manager configuration file that is associated with a particular instance. (This is done by executing an UPDATE DATABASE MANAGER CONFIGURATION command with the SYSMAINT\_GROUP parameter specified, along with the name of the group that is to receive System Maintenance authority.) Again, individual membership in the group itself is controlled through the security facility that is used on the workstation where the instance has been defined.

## System Monitor authority

System Monitor (SYSMON) authority is the third highest level of system or instance control authority available with DB2. Users who have been given this authority are allowed to take system monitor snapshots for a DB2 Database Manager instance and/or for one or more databases that fall under that instance's control. System Monitor authority is designed to allow special users to monitor the performance of a database that contains sensitive data that they most likely do not have the right to view or modify. Therefore, users who are granted this authority do not implicitly receive authority to access the data stored in the databases on which they are allowed to collect snapshot monitor information. However, because a connection to a database must exist before the snapshot monitor SQL table functions can be used, users who are granted System Monitor authority for a particular instance automatically receive the privileges needed to connect to each database under that instance's control.

Users with System Monitor authority (or higher) are allowed to perform the following tasks:

- Obtain the current settings of the snapshot monitor switches
- Modify the settings of one or more snapshot monitor switches
- Reset all counters used by the snapshot monitor
- Obtain a list of active databases
- Obtain a list of active applications, including DCS applications
- Collect snapshot monitor data
- Use the snapshot monitor SQL table functions

Like System Administrator, System Control, and System Maintenance authority, System Monitor authority can be assigned only to a group. This assignment is made by storing the appropriate group name in the *sysmon\_group* parameter of the DB2 Database Manager configuration file that is associated with a particular instance. (This is done by executing an UPDATE DATABASE MANAGER CONFIGURATION command with the SYSMON\_GROUP parameter specified, along with the name of the group that is to receive System Monitor authority.) Again,

individual membership in the group itself is controlled through the security facility that is used on the workstation where the instance has been defined.

## **Database Administrator authority**

Database Administrator (DBADM) authority is the second highest level of administrative authority available (just below System Administrator authority). Users who have been given this authority are allowed to run most DB2 utilities, issue database-specific DB2 commands, perform most SQL/XQuery operations, and access data stored in any table in a database—provided that data is not protected by LBAC. (To access data protected by LBAC, a user must have the appropriate LBAC credentials.) However, they can perform these functions only on the database for which Database Administrator authority is held.

Additionally, users with Database Administrator authority (or higher) are allowed to perform the following tasks:

- Read database log files
- Create, activate, and drop event monitors
- Query the state of a table space
- Update recovery history log files
- Quiesce (restrict access to) a table space
- Reorganize a table
- Collect catalog statistics using the RUNSTATS utility

Unlike System Administrator, System Control, System Maintenance, and System Monitor authority, Database Administrator authority can be assigned to both individual users and groups. This assignment is made by executing the appropriate form of the GRANT SQL statement (which we will look at shortly). When a user is given Database Administrator authority for a particular database, they automatically receive all database privileges available for that database as well.





Any time a user with SYSADM or SYSCTRL authority creates a new database, that user automatically receives DBADM authority on that database. Furthermore, if a user with SYSADM or SYSCTRL authority creates a database and is later removed from the SYSADM or SYSCTRL group (i.e., the user's SYSADM or SYSCTRL authority is revoked), the user retains DBADM authority for that database until it is explicitly removed (revoked).

### Security Administrator authority

Security Administrator (SECADM) authority is a special database level of authority that is designed to allow special users to configure various label-based access control (LBAC) elements to restrict access to one or more tables that contain data to which they most likely do not have access themselves. Users who are granted this authority do not implicitly receive authority to access the data stored in the databases for which they manage data access. In fact, users with Security Administrator authority are allowed to perform only the following tasks:

- Create and drop security policies
- Create and drop security labels
- Grant and revoke security labels to and from individual users (using the GRANT SECURITY LABEL and REVOKE SECURITY LABEL SQL statements)
- Grant and revoke LBAC rule exemptions
- Grant and revoke SETSESSIONUSER privileges (using the GRANT SETSESSIONUSER SQL statement)
- Transfer ownership of any object not owned by the Security Administrator (by executing the TRANSFER OWNERSHIP SQL statement)

No other authority, including System Administrator authority, provides a user with these abilities.

Security Administrator authority can be assigned only to individual users; it cannot be assigned to groups (including the group PUBLIC). This assignment is made by executing the appropriate form of the GRANT SQL statement, and only users with System Administrator authority are allowed to grant this authority.

### **Load authority**

Load (LOAD) authority is a special database level of administrative authority that has a much smaller scope than DBADM authority. Users who have been given this authority, along with INSERT and in some cases DELETE privileges, on a particular table are allowed to bulk-load data into that table, using either the AutoLoader utility (db2atld command) or the LOAD command/API. Load authority is designed to allow special users to perform bulk-load operations against a database with which they most likely cannot do anything else. This authority provides a way for Database Administrators to allow more users to perform special database operations, such as Extraction-Transform-Load (ETL) operations, without having to sacrifice control.

In addition to being able to load data into a database table, users with Load authority (or higher) are allowed to perform the following tasks:

- Query the state of a table space using the LIST TABLESPACES command.
- Quiesce (restrict access to) a table space.
- Perform bulk-load operations using the LOAD utility. (If exception tables are used as part of a load operation, the user must have INSERT privilege on the exception tables used as well as INSERT privilege on the table being loaded.)
- Collect catalog statistics using the RUNSTATS utility.

Like Database Administrator authority, Load authority can be assigned to both individual users and groups. This assignment is made by executing the appropriate form of the GRANT SQL statement.

## Privileges

As mentioned earlier, privileges are used to convey the rights to perform certain actions on specific database resources to both individual users and groups. With DB2 9, two distinct types of privileges exist: database privileges and object privileges.

### Database privileges

Database privileges apply to a database as a whole, and in many cases, they act as a second security checkpoint that must be cleared before access to data is provided. Figure 8–2 shows the different types of database privileges available.

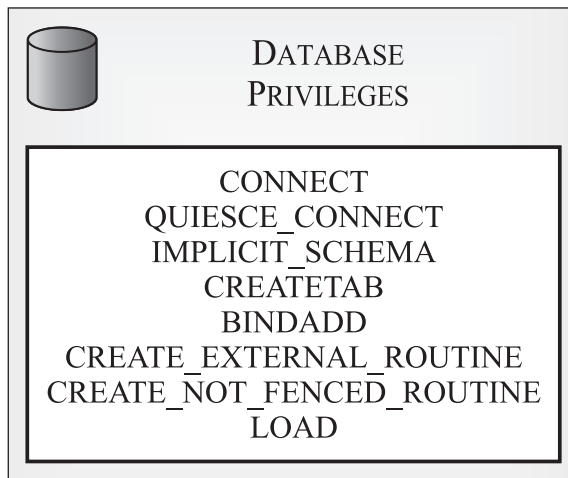


Figure 8–2: Database privileges available with DB2 9.

As you can see in Figure 8–2, eight different database privileges exist. They are:

**CONNECT.** Allows a user to establish a connection to the database.

**QUIESCE\_CONNECT.** Allows a user to establish a connection to the database while it is in a quiesced state (i.e., while access to it is restricted).

**IMPLICIT\_SCHEMA.** Allows a user to create a new schema in the database implicitly by creating an object and assigning that object a schema name that is different from any of the schema names that already exist in the database.

**CREATETAB.** Allows a user to create new tables in the database.

**BINDADD.** Allows a user to create packages in the database (by precompiling embedded SQL application source code files against the database or by binding application bind files to the database).

**CREATE\_EXTERNAL\_ROUTINE.** Allows a user to create user-defined functions (UDFs) and/or procedures and store them in the database so that they can be used by other users and applications.

**CREATE\_NOT\_FENCED\_ROUTINE.** Allows a user to create unfenced UDFs and/or procedures and store them in the database. (Unfenced UDFs and stored procedures are UDFs/procedures that are considered “safe” enough to be run in the DB2 Database Manager operating environment’s process or address space. Unless a UDF/procedure is registered as unfenced, the DB2 Database Manager insulates the UDF/procedure’s internal resources in such a way that they cannot be run in the DB2 Database Manager’s address space.)

**LOAD.** Allows a user to bulk-load data into one or more existing tables in the database.

At a minimum, a user must have **CONNECT** privilege on a database before he or she can work with any object contained in that database.

## **Object privileges**

Unlike database privileges, which apply to a database as a whole, object privileges apply only to specific objects within a database. These objects include table spaces, schemas, tables, views, indexes, sequences, routines, packages, servers, and nicknames. Because the nature of each database object available varies, the individual privileges that exist for each object can vary as well. The following sections describe the different sets of object privileges that are available with DB2 9.

**Table space privileges.** Table space privileges control what users can and cannot do with a particular table space. (Table spaces are used to control where data in a database physically resides.) Figure 8–3 shows the only table space privilege available.

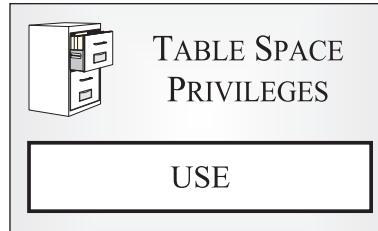


Figure 8-3: Table space privilege available with DB2 9.

As you can see in Figure 8-3, only one table space privilege exists. That privilege is the USE privilege, which, when granted, allows a user to create tables and indexes in the table space. The owner of a table space (usually the individual who created the table space) automatically receives USE privilege for that table space.



.....

The USE privilege cannot be used to provide a user with the ability to create tables in the SYSCATSPACE table space or in any temporary table space that might exist.

.....

**Schema privileges.** Schema privileges control what users can and cannot do with a particular schema. (A schema is an object that is used to logically classify and group other objects in the database; most objects are named using a naming convention that consists of a schema name, followed by a period, followed by the object name.) Figure 8-4 shows the different types of schema privileges available.

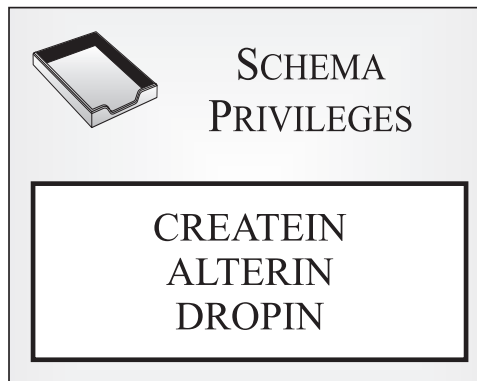


Figure 8-4: Schema privileges available with DB2 9.

As you can see in Figure 8–4, three different schema privileges exist. They are:

**CREATEIN.** Allows a user to create objects within the schema.

**ALTERIN.** Allows a user to change the comment associated with any object in the schema or to alter any object that resides within the schema.

**DROPIN.** Allows a user to remove (drop) any object within the schema.

Objects that can be manipulated within a schema include tables, views, indexes, packages, user-defined data types, user-defined functions, triggers, stored procedures, and aliases. The owner of a schema (usually the individual who created the schema) automatically receives all privileges available for that schema, along with the right to grant any combination of those privileges to other users and groups.

**Table privileges.** Table privileges control what users can and cannot do with a particular table in a database. (A table is a logical structure used to present data as a collection of unordered rows with a fixed number of columns.) Figure 8–5 shows the different types of table privileges available.

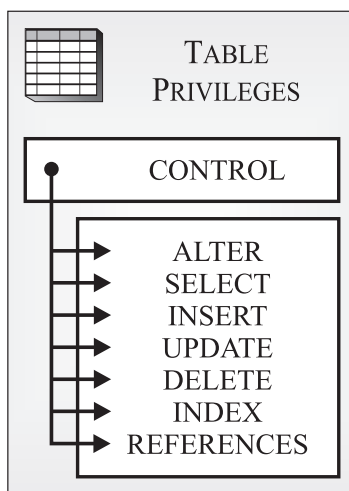


Figure 8–5: Table privileges available with DB2 9.

As you can see in Figure 8–5, eight different table privileges exist. They are:

**CONTROL.** Provides a user with every table privilege available, allows the user to remove (drop) the table from the database, and gives the user the ability to grant and revoke one or more table privileges (except the CONTROL privilege) to and from other users and groups.

**ALTER.** Allows a user to execute the ALTER TABLE SQL statement against the table. In other words, allows a user to add columns to the table, add or change comments associated with the table or any of its columns, create or drop a primary key for the table, create or drop a unique constraint for the table, create or drop a check constraint for the table, create or drop a referential constraint for the table, and create or drop triggers for the table (provided the user holds the appropriate privileges for every object referenced by the trigger).

**SELECT.** Allows a user to execute a SELECT SQL statement against the table. In other words, this privilege allows a user to retrieve data from a table, create a view that references the table, and run the Export utility against the table.

**INSERT.** Allows a user to execute the INSERT SQL statement against the table. In other words, this privilege allows a user to add data to the table and run the Import utility against the table.

**UPDATE.** Allows a user to execute the UPDATE SQL statement against the table. In other words, this privilege allows a user to modify data in the table. (This privilege can be granted for the entire table or limited to one or more columns within the table.)

**DELETE.** Allows a user to execute the DELETE SQL statement against the table. In other words, allows a user to remove rows of data from the table.

**INDEX.** Allows a user to create an index for the table.

**REFERENCES.** Allows a user to create and drop foreign key constraints that reference the table in a parent relationship. (This privilege can be granted for the entire table or limited to one or more columns within the

table, in which case only those columns can participate as a parent key in a referential constraint.)

The owner of a table (usually the individual who created the table) automatically receives all privileges available for that table (including CONTROL privilege), along with the right to grant any combination of those privileges (except CONTROL privilege) to other users and groups. If the CONTROL privilege is later revoked from the table owner, all other privileges that were automatically granted to the owner for that particular table are not automatically revoked. Instead, they must be explicitly revoked in one or more separate operations.

**View privileges.** View privileges control what users can and cannot do with a particular view. (A view is a virtual table residing in memory that provides an alternative way of working with data that resides in one or more base tables. For this reason, views can be used to prevent access to select columns in a table.) Figure 8–6 shows the different types of view privileges available.

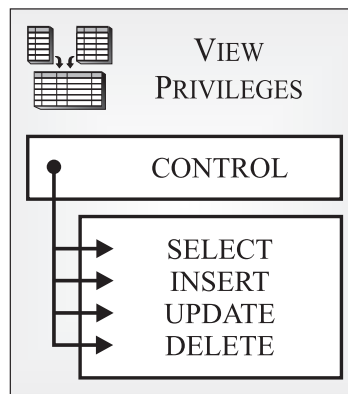


Figure 8–6: View privileges available with DB2 9.

As you can see in Figure 8–6, five different view privileges exist. They are:

**CONTROL.** Provides a user with every view privilege available, allows the user to remove (drop) the view from the database, and gives the user the ability to grant and revoke one or more view privileges (except the CONTROL privilege) to and from other users and groups.



**SELECT.** Allows a user to retrieve data from the view, create a second view that references the view, and run the Export utility against the view.

**INSERT.** Allows a user to execute the INSERT SQL statement against the view. In other words, allows a user to add data to the view.

**UPDATE.** Allows a user to execute the UPDATE SQL statement against the view. In other words, this privilege allows a user to modify data in the view. (This privilege can be granted for the entire view or limited to one or more columns within the view.)

**DELETE.** Allows a user to execute the DELETE SQL statement against the view. In other words, this privilege allows a user to remove rows of data from the view.

In order to create a view, a user must hold appropriate privileges (at a minimum, SELECT privilege) on each base table the view references. The owner of a view (usually the individual who created the view) automatically receives all privileges available—with the exception of the CONTROL privilege—for that view, along with the right to grant any combination of those privileges (except CONTROL privilege) to other users and groups. A view owner will receive CONTROL privilege for a view only if he or she also holds CONTROL privilege for every base table the view references.



.....

If a user who holds SELECT privilege on one or more tables creates a view based on one or more of those tables and his or her SELECT privileges are later revoked, the view will become inoperative, and any privileges that have been granted for that view will be revoked automatically.

.....

**Index privileges.** Index privileges control what users can and cannot do with a particular index. (An index is an ordered set of pointers that refer to one or more key columns in a base table; indexes are used to improve query performance.)

Figure 8–7 shows the only index privilege available.

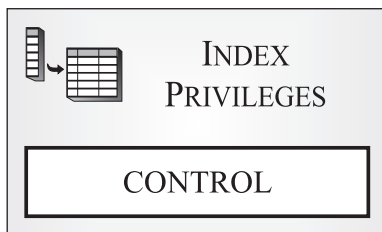


Figure 8–7: Index privilege available with DB2 9.

As you can see in Figure 8–7, only one index privilege exists. That privilege is the **CONTROL** privilege, which, when granted, allows a user to remove (drop) the index from the database. Unlike the **CONTROL** privilege for other objects, the **CONTROL** privilege for an index does not give a user the ability to grant and revoke index privileges to and from other users and groups. That’s because the **CONTROL** privilege is the only index privilege available, and only users who hold System Administrator (SYSADM) or Database Administrator (DBADM) authority are allowed to grant and revoke **CONTROL** privileges for an object.

The owner of an index (usually the individual who created the index) automatically receives **CONTROL** privilege for that index.

**Sequence privileges.** Sequence privileges control what users can and cannot do with a particular sequence. (A sequence is an object that can be used to generate values automatically. Sequences are ideal for generating unique key values, and they can be used to avoid the possible concurrency and performance problems that can occur when unique counters residing outside the database are used for data generation.) Figure 8–8 shows the different types of sequence privileges available.

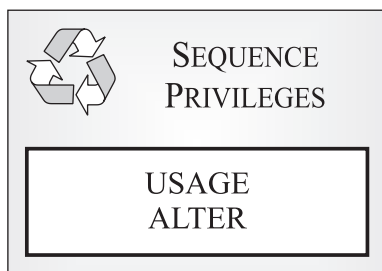


Figure 8–8: Sequence privileges available with DB2 9.

As you can see in Figure 8–8, two different sequence privileges exist. They are:

**USAGE.** Allows a user to use the PREVIOUS VALUE and NEXT VALUE expressions that are associated with the sequence. (The PREVIOUS VALUE expression returns the most recently generated value for the specified sequence; the NEXT VALUE expression returns the next value for the specified sequence.)

**ALTER.** Allows a user to perform administrative tasks such as restarting the sequence, changing the increment value for the sequence, and adding or changing the comment associated with the sequence.

The owner of a sequence (usually the individual who created the sequence) automatically receives all privileges available for that sequence, along with the right to grant any combination of those privileges to other users and groups.

**Routine privileges.** Routine privileges control what users can and cannot do with a particular routine. (A routine can be a user-defined function, a stored procedure, or a method that can be invoked by several different users.) Figure 8–9 shows the only routine privilege available.

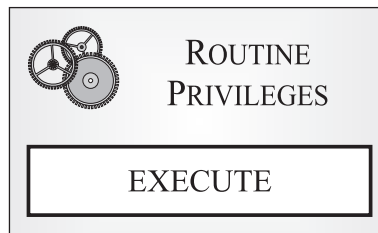


Figure 8–9: Routine privilege available with DB2 9.

As you can see in Figure 8–9, only one routine privilege exists. That privilege is the EXECUTE privilege, which, when granted, allows a user to invoke the routine, create a function that is sourced from the routine (provided the routine is a function), and reference the routine in any Data Definition Language SQL statement (for example, CREATE VIEW and CREATE TRIGGER).



.....

Before a user can invoke a routine (user-defined function, stored procedure, or method), he or she must hold both EXECUTE privilege on the routine and any privileges required by that routine. Thus, in order to execute a stored procedure that queries a table, a user must hold both EXECUTE privilege on the stored procedure and SELECT privilege on the table against which the query is run.

.....

The owner of a routine (usually the individual who created the routine) automatically receives EXECUTE privilege for that routine.

**Package privileges.** Package privileges control what users can and cannot do with a particular package. (A package is an object that contains the information needed by the DB2 Database Manager to process SQL statements in the most efficient way possible on behalf of an embedded SQL application.) Figure 8–10 shows the different types of package privileges available.

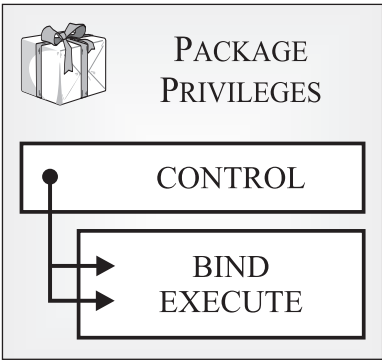


Figure 8–10: Package privileges available with DB2 9.

As you can see in Figure 8–10, three different package privileges exist. They are:

**CONTROL.** Provides a user with every package privilege available, allows the user to remove (drop) the package from the database, and gives the user the ability to grant and revoke one or more table privileges (except the CONTROL privilege) to and from other users and groups.

**BIND.** Allows a user to rebind or add new package versions to a package that has already been bound to a database. (In addition to the BIND package privilege, a user must hold the privileges needed to execute the SQL statements that make up the package before the package can be successfully rebound.)

**EXECUTE.** Allows a user to execute the package. (A user who has EXECUTE privilege for a particular package can execute that package, even if the user does not have the privileges that are needed to execute the SQL statements stored in the package. That is because any privileges needed to execute the SQL statements are implicitly granted to the package user. It is important to note that for privileges to be implicitly granted, the creator of the package must hold privileges as an individual user or as a member of the group PUBLIC—not as a member of another named group.)

The owner of a package (usually the individual who created the package) automatically receives all privileges available for that package (including CONTROL privilege), along with the right to grant any combination of those privileges (except CONTROL privilege) to other users and groups. If the CONTROL privilege is later revoked from the package owner, all other privileges that were automatically granted to the owner for that particular package are not automatically revoked. Instead, they must be explicitly revoked in one or more separate operations.



.....

Users who have EXECUTE privilege for a package that contains nicknames do not need additional authorities or privileges for the nicknames in the package; however, they must be able to pass any authentication checks performed at the data source(s) in which objects referenced by the nicknames are stored, and they must hold the appropriate authorizations and privileges needed to access all objects referenced.

.....

**Server privileges.** Server privileges control what users can and cannot do with a particular federated database server. (A DB2 federated system is a distributed computing system that consists of a DB2 server, known as a *federated server*, and one or more data sources to which the federated server sends queries. Each data source consists of an instance of some supported relational database management system—such as Oracle—plus the database or databases that the instance supports.) Figure 8–11 shows the only server privilege available.

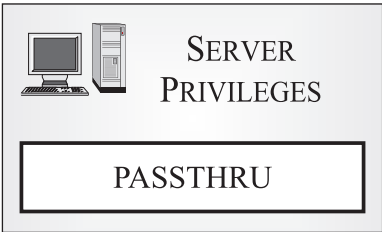


Figure 8–11: Server privilege available with DB2 9.

As you can see in Figure 8–11, only one server privilege exists. That privilege is the PASSTHRU privilege, which, when granted, allows a user to issue Data Definition Language (DDL) and Data Manipulation Language (DML) SQL statements (as pass-through operations) directly to a data source via a federated server.

**Nickname privileges.** Nickname privileges control what users can and cannot do with a particular nickname. (When a client application submits a distributed request to a federated database server, the server forwards the request to the appropriate data source for processing. However, such a request does not identify the data source itself; instead, it references tables and views within the data source by using nicknames that map to specific table and view names in the data source. Nicknames are not alternate names for tables and views in the same way that aliases are; instead, they are pointers by which a federated server references external objects.) Figure 8–12 shows the different types of nickname privileges available.

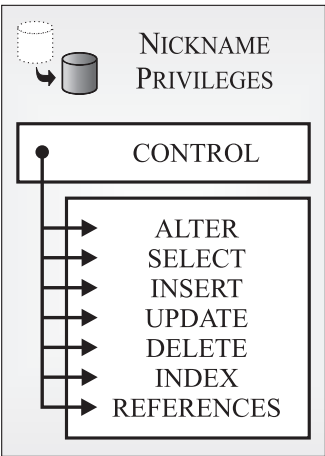


Figure 8–12: Nickname privileges available with DB2 9.

As you can see in Figure 8–12, eight different nickname privileges exist. They are:

**CONTROL.** Provides a user with every nickname privilege available, allows the user to remove (drop) the nickname from the database, and gives the user the ability to grant and revoke one or more nickname privileges (except the CONTROL privilege) to and from other users and groups.

**ALTER.** Allows a user to execute the ALTER NICKNAME SQL statement against the table. In other words, this privilege allows a user to change column names in the nickname, add or change the DB2 data type to which a particular nickname column's data type maps, and specify column options for a specific nickname column.

**SELECT.** Allows a user to execute a SELECT SQL statement against the nickname. In other words, this privilege allows a user to retrieve data from the table or view within a federated data source to which the nickname refers.

**INSERT.** Allows a user to execute the INSERT SQL statement against the nickname. In other words, this privilege allows a user to add data to the table or view within a federated data source to which the nickname refers.

**UPDATE.** Allows a user to execute the UPDATE SQL statement against the nickname. In other words, this privilege allows a user to modify data in the table or view within a federated data source to which the nickname refers. (This privilege can be granted for the entire table or limited to one or more columns within the table to which the nickname refers.)

**DELETE.** Allows a user to execute the DELETE SQL statement against the nickname. In other words, allows a user to remove rows of data from the table or view within a federated data source to which the nickname refers.

**INDEX.** Allows a user to create an index specification for the nickname.

**REFERENCES.** Allows a user to create and drop foreign key constraints that reference the nickname in a parent relationship.

The owner of a nickname (usually the individual who created the table) automatically receives all privileges available for that nickname (including CONTROL privilege), along with the right to grant any combination of those privileges (except CONTROL privilege) to other users and groups. If the CONTROL privilege is later revoked from the nickname owner, all other privileges that were automatically granted to the owner for that particular table are not automatically revoked. Instead, they must be explicitly revoked in one or more separate operations.

## **Granting Authorities and Privileges**

There are three different ways that users (and in some cases, groups) can obtain database-level authorities and database/object privileges. They are:

**Implicitly.** When a user creates a database, that user implicitly receives Database Administrator authority for that database, along with most database privileges available. Likewise, when a user creates a database object, that user implicitly receives all privileges available for that object, along with the ability to grant any combination of those privileges (with the exception of the CONTROL privilege) to other users and groups. Privileges can also be implicitly given whenever a higher-level privilege is explicitly granted to a user (for example, if a user is explicitly given CONTROL privilege for a table space, the user will implicitly receive the USE privilege for that table space as well). It's important to remember that such implicitly assigned privileges are not automatically revoked when the higher-level privilege that caused them to be granted is revoked.

**Indirectly.** Indirectly assigned privileges are usually associated with packages; when a user executes a package that requires additional privileges that the user does not have (for example, a package that deletes a row of data from a table requires the DELETE privilege on that table), the user is indirectly given those privileges for the express purpose of executing the package. Indirectly granted privileges are temporary and do not exist outside the scope in which they are granted.

**Explicitly.** Database-level authorities, database privileges, and object privileges can be explicitly given to or taken from an individual user or a group of users by anyone who has the authority to do so. To grant privileges



explicitly on most database objects, a user must have System Administrator (SYSADM) or Database Administrator (DBADM) authority, or CONTROL privilege on that object. Alternately, a user can explicitly grant any privilege that user was assigned with the WITH GRANT OPTION specified. To grant CONTROL privilege for any object, a user must have System Administrator (SYSADM) or Database Administrator (DBADM) authority; to grant System Administrator (SYSADM) or Database Administrator (DBADM) authority, a user must have System Administrator (SYSADM) authority.

### ***Granting and Revoking Authorities and Privileges from the Control Center***

One way to explicitly grant and revoke database-level authorities, as well as many of the object privileges available, is by using the various authorities and privileges management dialogs that are provided with the Control Center. These dialogs are activated by highlighting the appropriate database or object name shown in the Control Center panes and selecting either Authorities or Privileges from the corresponding database or object menu. Figure 8–13 shows the menu items that must be selected in the Control Center in order to activate the Table Privileges dialog for a particular table. Figure 8–14 shows how the Table Privileges dialog might look immediately after a table is first created. (A single check mark under a privilege means that the individual or group shown has been granted that privilege; a double check mark means the individual or group has also been granted ability to grant that privilege to other users and groups.)

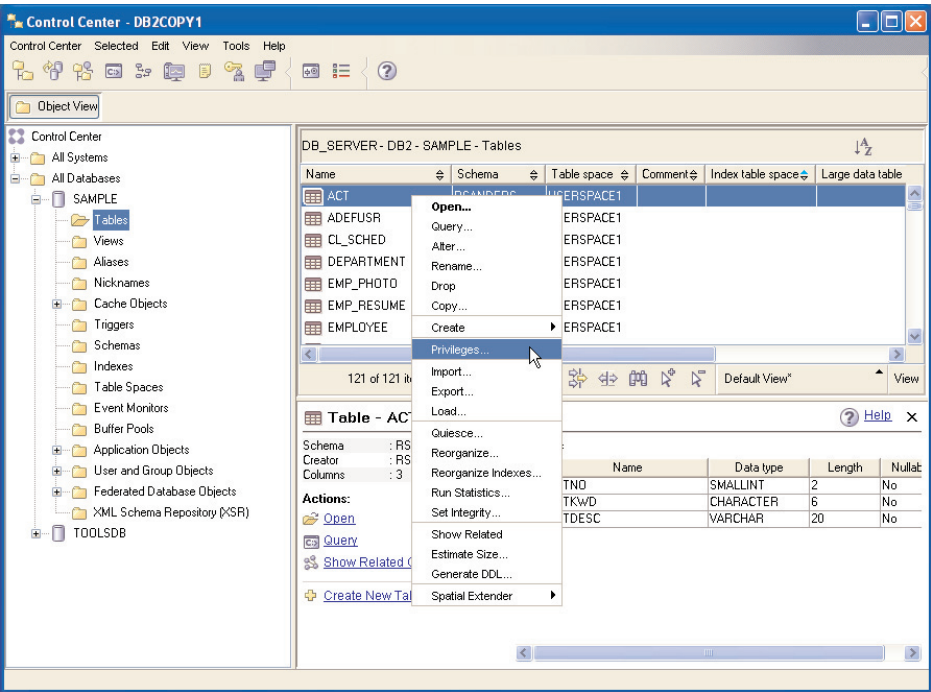


Figure 8-13: Invoking the Table Privileges dialog from the Control Center.

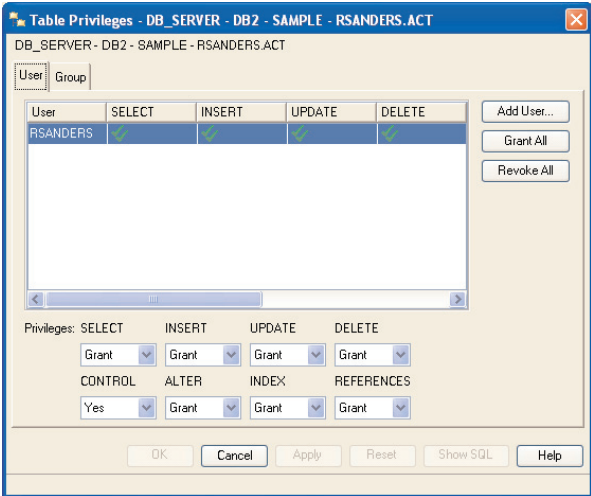


Figure 8-14: The Table privileges dialog.

To assign privileges to an individual user from the Table Privileges dialog (or a similar authorities/privileges dialog), you simply identify a particular user by highlighting the user's entry in the recognized users list—if the desired user is not in the list, the user can be added by selecting the “Add User” push button—and assigning the appropriate privileges (or authorities) using the “Privileges” (or “Authorities”) drop-down list(s) or the “Grant All” or “Revoke All” push buttons. To assign privileges to a group of users, you select the “Group” tab to display a list of recognized groups and repeat the process (using the “Add Group” push button instead of the “Add User” push button to add a desired group to the list if the group is not already there).

### **Granting Authorities and Privileges with the GRANT SQL Statement**

Not all privileges can be explicitly given to users or groups with the privileges management dialogs available. In situations where no privileges dialog exists (and in situations where you elect not to use the privileges dialogs available), database-level authorities and database/object privileges can be explicitly given to users and/or groups by executing the appropriate form of the GRANT SQL statement. The syntax for the GRANT SQL statement varies according to the authority or privilege being granted. The following subsections show the syntax used to grant each database-level authority and database/object privilege available.

#### **Database-level authorities and privileges**

```
GRANT [Privilege, ...] ON DATABASE
TO [Recipient, ...]
```

where:

*Privilege*

Identifies one or more database privileges that are to be given to one or more users and/or groups. The following values are valid for this parameter: DBADM, SECADM, CONNECT, CONNECT\_QUIESCE, IMPLICIT\_SCHEMA, CREATETAB, BINDADD, CREATE\_EXTERNAL\_ROUTINE, CREATE\_NOT\_FENCED\_ROUTINE, and LOAD.

<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the database privileges specified. The value specified for the <i>Recipient</i> parameter can be any combination of the following:
<USER> [ <i>UserName</i> ]	Identifies a specific user to which the privilege(s) specified are to be given.
<GROUP> [ <i>GroupName</i> ]	Identifies a specific group to whom the privilege(s) specified are to be given.
PUBLIC	Indicates that the privilege(s) specified are to be given to the group PUBLIC. (All users are members of the group PUBLIC.)



.....

Checking is not performed to ensure that the names of users and/or groups specified in the *Recipient* parameter are valid. Therefore, it is possible to grant privileges to users and groups that do not yet exist.

.....

**Table space privileges**

```
GRANT USE OF TABLESPACE [TablespaceName]  
TO [Recipient, ...]  
<WITH GRANT OPTION>
```

where:

<i>TablespaceName</i>	Identifies by name the table space that the USE privilege is to be associated with.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the USE privilege. Again, the value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

If the WITH GRANT OPTION clause is specified, each *Recipient* is given the ability to grant the privilege received to others.

### Schema privileges

```
GRANT [Privilege, ...] ON SCHEMA [SchemaName]
TO [Recipient, ...]
<WITH GRANT OPTION>
```

where:

<i>Privilege</i>	Identifies one or more schema privileges that are to be given to one or more users and/or groups. The following values are valid for this parameter: CREATIN, ALTERIN, and DROPIN.
<i>SchemaName</i>	Identifies by name the schema with which all schema privileges specified are to be associated.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the schema privileges specified. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

### Table privileges

```
GRANT [ALL <PRIVILEGES> |
      Privilege <( ColumnName, ... )> , ...]
ON TABLE [TableName]
TO [Recipient, ...]
<WITH GRANT OPTION>
```

where:

<i>Privilege</i>	Identifies one or more table privileges that are to be given to one or more users and/or groups. The following values are valid for this parameter: CONTROL, ALTER, SELECT, INSERT, UPDATE, DELETE, INDEX, and REFERENCES.
------------------	--

<i>ColumnName</i>	Identifies by name one or more specific columns with which UPDATE or REFERENCES privileges are to be associated. This option is used only when the <i>Privilege</i> parameter contains the value UPDATE or REFERENCES.
<i>TableName</i>	Identifies by name the table with which all table privileges specified are to be associated.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the table privileges specified. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

It is important to note that only users who hold System Administrator (SYSADM) or Database Administrator (DBADM) authority are allowed to grant CONTROL privilege for a table. For this reason, when the ALL PRIVILEGES clause is specified, all table privileges *except* CONTROL privilege are granted to each *Recipient*; CONTROL privilege must be granted separately.

### **View privileges**

```
GRANT [ALL <PRIVILEGES> |  
      Privilege <( ColumnName, ... )> , ...]  
ON [ ViewName ]  
TO [ Recipient, ... ]  
<WITH GRANT OPTION>
```

where:

<i>Privilege</i>	Identifies one or more view privileges that are to be given to one or more users and/or groups. The following values are valid for this parameter: CONTROL, SELECT, INSERT, UPDATE, and DELETE.
<i>ColumnName</i>	Identifies by name one or more specific columns with which UPDATE privileges are to be associated. This option is used only when the <i>Privilege</i> parameter contains the value UPDATE.

<i>ViewName</i>	Identifies by name the view with which all view privileges specified are to be associated.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the view privileges specified. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

Again, only users who hold System Administrator (SYSADM) or Database Administrator (DBADM) authority are allowed to grant CONTROL privilege for a table. Therefore, when the ALL PRIVILEGES clause is specified, all view privileges *except* CONTROL privilege are granted to each *Recipient*; CONTROL privilege must be granted separately.

### Index privileges

```
GRANT CONTROL ON INDEX [IndexName]
TO [Recipient, ...]
```

where:

<i>IndexName</i>	Identifies by name the index with which the CONTROL privilege is to be associated.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the CONTROL privilege. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

### Sequence privileges

```
GRANT [Privilege, ...] ON SEQUENCE [SequenceName]
TO [Recipient, ...]
<WITH GRANT OPTION>
```

where:

<i>Privilege</i>	Identifies one or more sequence privileges that are to be given to one or more users and/or groups. The following values are valid for this parameter: USAGE and ALTER.
<i>SequenceName</i>	Identifies by name the sequence with which all sequence privileges specified are to be associated.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the sequence privileges specified. Again, the value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

## Routine privileges

```
GRANT EXECUTE ON [RoutineName |  
    FUNCTION <SchemaName.> * |  
    METHOD * FOR [TypeName] |  
    METHOD * FOR <SchemaName.> * |  
    PROCEDURE <SchemaName.> *]  
TO [Recipient, ...]  
<WITH GRANT OPTION>
```

where:

<i>RoutineName</i>	Identifies by name the routine (user-defined function, method, or stored procedure) with which the EXECUTE privilege is to be associated.
<i>TypeName</i>	Identifies by name the type in which the specified method is found.
<i>SchemaName</i>	Identifies by name the schema in which all functions, methods, or procedures—including those that may be created in the future—are to have the EXECUTE privilege granted.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the EXECUTE privilege. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.



## Package privileges

```
GRANT [Privilege, ...] ON PACKAGE <SchemaName.>[PackageID] TO [Recipient,
...]  
<WITH GRANT OPTION>
```

where:

<i>Privilege</i>	Identifies one or more package privileges that are to be given to one or more users and/or groups. The following values are valid for this parameter: CONTROL, BIND, and EXECUTE.
<i>SchemaName</i>	Identifies by name the schema in which the specified package is found.
<i>PackageName</i>	Identifies by name the package with which all package privileges specified are to be associated.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the package privileges specified. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC. (DB2 for Linux, UNIX, and Windows does not allow users to grant package privileges to themselves.)

## Server privileges

```
GRANT PASSTHRU ON SERVER [ServerName]  
TO [Recipient, ...]
```

where:

<i>ServerName</i>	Identifies by name the server with which the PASSTHRU privilege is to be associated.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the PASSTHRU privilege. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

## Nickname privileges

```
GRANT [ALL <PRIVILEGES> |  
      Privilege <( ColumnName, ... )> , ...]  
ON [Nickname]  
TO [Recipient, ...]  
<WITH GRANT OPTION>
```

where:

*Privilege*            Identifies one or more nickname privileges that are to be given to one or more users and/or groups. The following values are valid for this parameter: CONTROL, ALTER, SELECT, INSERT, UPDATE, DELETE, INDEX, and REFERENCES.

*ColumnName*        Identifies by name one or more specific columns with which UPDATE or REFERENCES privileges are to be associated. This option is used only when the *Privilege* parameter contains the value UPDATE or REFERENCES.

*Nickname*           Identifies by name the nickname with which all privileges specified are to be associated.

*Recipient*           Identifies the name of the user(s) and/or group(s) that are to receive the nickname privileges specified. The value specified for the *Recipient* parameter can be any combination of the following: <USER> [*UserName*], <GROUP> [*GroupName*], and PUBLIC.

Only users who hold System Administrator (SYSADM) or Database Administrator (DBADM) authority are allowed to grant CONTROL privilege for a nickname. Therefore, when the ALL PRIVILEGES clause is specified, all nickname privileges *except* CONTROL privilege are granted to each *Recipient*; CONTROL privilege must be granted separately.

## GRANT SQL statement examples

Now that we've seen the basic syntax for the various forms of the GRANT SQL statement, let's take a look at some examples.

**Example 1.** A server has both a user and a group named TESTER. Give the group TESTER the ability to bind applications to the database SAMPLE:

```
CONNECT TO sample;  
GRANT BINDADD ON DATABASE TO GROUP tester;
```

**Example 2.** Give all table privileges available for the table PAYROLL.EMPLOYEE (except CONTROL privilege) to the group PUBLIC:

```
GRANT ALL PRIVILEGES ON TABLE payroll.employee TO PUBLIC
```

**Example 3.** Give user USER1 and user USER2 the privileges needed to perform DML operations on the table DEPARTMENT using the view DEPTVIEW:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON deptview  
TO USER user1, USER user2
```

**Example 4.** Give user JOHN\_DOE the privileges needed to query the table INVENTORY, along with the ability to give these privileges to other users whenever appropriate:

```
GRANT SELECT ON TABLE inventory  
TO john_doe  
WITH GRANT OPTION
```

**Example 5.** Give user USER1 the ability to run an embedded SQL application that requires a package named GET\_INVENTORY:

```
GRANT EXECUTE ON PACKAGE get_inventory TO USER user1
```

**Example 6.** Give user USER1 the ability to use a stored procedure named PAYROLL.CALC\_SALARY in a query:

```
GRANT EXECUTE ON PROCEDURE payroll.calc_salary TO user1
```

**Example 7.** Give user USER1 and group GROUP1 the ability to define a referential constraint between the tables EMPLOYEE and DEPARTMENT using column EMPID in table EMPLOYEE as the parent key:

```
GRANT REFERENCES(empid) ON TABLE employee TO USER user1,  
GROUP group1
```

**Example 8.** Give the group PUBLIC the ability to modify information stored in the ADDRESS and HOME\_PHONE columns of the table EMP\_INFO:

```
GRANT UPDATE(address, home_phone) ON TABLE emp_info
TO PUBLIC
```

### **Revoking Authorities and Privileges with the REVOKE SQL Statement**

Just as there is an SQL statement that can be used to grant database-level authorities and database/object privileges, there is an SQL statement that can be used to revoke database-level authorities and database/object privileges. This statement is the REVOKE SQL statement, and as with the GRANT statement, the syntax for the REVOKE statement varies according to the authority or privilege being revoked. The following sections show the syntax used to revoke each database-level authority and database/object privilege available.

#### **Database-level authorities and privileges**

```
REVOKE [Privilege, ...] ON DATABASE
FROM [Forfeiter, ...] <BY ALL>
```

where:

*Privilege*

Identifies one or more database privileges that are to be taken from one or more users and/or groups. The following values are valid for this parameter: DBADM, SECADM, CONNECT, CONNECT\_QUIESCE, IMPLICIT\_SCHEMA, CREATETAB, BINDADD, CREATE\_EXTERNAL\_ROUTINE, CREATE\_NOT\_FENCED\_ROUTINE, and LOAD.

*Forfeiter*

Identifies the name of the user(s) and/or group(s) that are to lose the database privileges specified. The value specified for the *Forfeiter* parameter can be any combination of the following:

<USER> [ <i>UserName</i> ]	Identifies a specific user from whom the privilege(s) specified are to be taken.
<GROUP> [ <i>GroupName</i> ]	Identifies a specific group from which the privilege(s) specified are to be taken.
PUBLIC	Indicates that the privilege(s) specified are to be taken from the group PUBLIC. (All users are members of the group PUBLIC.)

The BY ALL clause is optional and is provided as a courtesy for administrators who are familiar with the syntax of the DB2 for OS/390 REVOKE SQL statement. Whether it is included or not, all privileges specified will be revoked from all users and/or groups specified.

It is important to note that when Database Administrator (DBADM) authority is revoked, privileges held on objects in the database by the *Forfeiter* specified are not automatically revoked. The same is true for all other database authorities that were implicitly and automatically granted when DBADM authority was granted.

### Table space privileges

```
REVOKE USE OF TABLESPACE [TablespaceName]
FROM [Forfeiter, ...] <BY ALL>
```

where:

<i>TablespaceName</i>	Identifies by name the table space with which the USE privilege is associated.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the USE privilege. Again, the value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

**Schema privileges**

```
REVOKE [Privilege, ...] ON SCHEMA [SchemaName]
FROM [Forfeiter, ...] <BY ALL>
```

where:

<i>Privilege</i>	Identifies one or more schema privileges that are to be taken from one or more users and/or groups. The following values are valid for this parameter: CREATIN, ALTERIN, and DROPIN.
<i>SchemaName</i>	Identifies by name the schema with which all schema privileges specified are to be associated.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the schema privileges specified. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

**Table privileges**

```
REVOKE [ALL <PRIVILEGES> |
      Privilege, ...]
ON TABLE [TableName]
FROM [Forfeiter, ...] <BY ALL>
```

where:

<i>Privilege</i>	Identifies one or more table privileges that are to be taken from one or more users and/or groups. The following values are valid for this parameter: CONTROL, ALTER, SELECT, INSERT, UPDATE, DELETE, INDEX, and REFERENCES.
<i>TableName</i>	Identifies by name the table with which all table privileges specified are to be associated.

*Forfeiter* Identifies the name of the user(s) and/or group(s) that are to lose the table privileges specified. The value specified for the *Forfeiter* parameter can be any combination of the following: <USER> [*UserName*], <GROUP> [*GroupName*], and PUBLIC.

It is important to note that only users who hold System Administrator (SYSADM) or Database Administrator (DBADM) authority are allowed to revoke CONTROL privilege for a table. For this reason, when the ALL PRIVILEGES clause is specified, all table privileges *except* CONTROL privilege are revoked from each *Forfeiter*; CONTROL privilege must be revoked separately.

## View privileges

```
REVOKE [ALL <PRIVILEGES> |
      Privilege, ...]
ON [ViewName]
FROM [Forfeiter, ...] <BY ALL>
```

where:

*Privilege* Identifies one or more view privileges that are to be taken from one or more users and/or groups. The following values are valid for this parameter: CONTROL, SELECT, INSERT, UPDATE, and DELETE.

*ViewName* Identifies by name the view with which all view privileges specified are to be associated.

*Forfeiter* Identifies the name of the user(s) and/or group(s) that are to lose the view privileges specified. The value specified for the *Forfeiter* parameter can be any combination of the following: <USER> [*UserName*], <GROUP> [*GroupName*], and PUBLIC.

Again, only users who hold System Administrator (SYSADM) or Database Administrator (DBADM) authority are allowed to revoke CONTROL privilege for a table. For this reason, when the ALL PRIVILEGES clause is specified, all table

privileges *except* CONTROL privilege are revoked from each *Forfeiter*; CONTROL privilege must be revoked separately.

### Index privileges

```
REVOKE CONTROL ON INDEX [IndexName]  
FROM [Forfeiter, ...] <BY ALL>
```

where:

<i>IndexName</i>	Identifies by name the index with which the CONTROL privilege is associated.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the CONTROL privilege. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

### Sequence privileges

```
REVOKE [Privilege, ...] ON SEQUENCE [SequenceName]  
FROM [Forfeiter, ...] <BY ALL>
```

where:

<i>Privilege</i>	Identifies one or more sequence privileges that are to be taken from one or more users and/or groups. The following values are valid for this parameter: USAGE and ALTER.
<i>SequenceName</i>	Identifies by name the sequence with which all sequence privileges specified are to be associated with.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the sequence privileges specified. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.



## Routine privileges

```

REVOKE EXECUTE ON [RoutineName |
    FUNCTION <SchemaName.> * |
    METHOD * FOR [TypeName] |
    METHOD * FOR <SchemaName.> * |
    PROCEDURE <SchemaName.> *]
FROM [Forfeiter, ...] <BY ALL>
RESTRICT

```

where:

<i>RoutineName</i>	Identifies by name the routine (user-defined function, method, or stored procedure) with which the EXECUTE privilege is associated.
<i>TypeName</i>	Identifies by name the type in which the specified method is found.
<i>SchemaName</i>	Identifies by name the schema from which all functions, methods, or procedures—including those that may be created in the future—are to have the EXECUTE privilege revoked.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the EXECUTE privilege. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

The RESTRICT clause guarantees EXECUTE privilege will not be revoked if the routine specified is used in a view, trigger, constraint, index, SQL function, SQL method, or transform group or is referenced as the source of a sourced function. Additionally, EXECUTE privilege will not be revoked if the loss of the privilege would prohibit the routine definer from executing the routine (i.e., if the user who created the routine is identified as a *Forfeiter*).

## Package privileges

```

REVOKE [Privilege, ...] ON PACKAGE <SchemaName.>[PackageID]
FROM [Forfeiter, ...] <BY ALL>

```

where:

<i>Privilege</i>	Identifies one or more package privileges that are to be taken from one or more users and/or groups. The following values are valid for this parameter: CONTROL, BIND, and EXECUTE.
<i>SchemaName</i>	Identifies by name the schema in which the specified package is found.
<i>PackageName</i>	Identifies by name the specific package with which all package privileges specified are to be associated.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the package privileges specified. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

### Server privileges

```
REVOKE PASSTHRU ON SERVER [ServerName]  
FROM [Forfeiter, ...] <BY ALL>
```

where:

<i>ServerName</i>	Identifies by name the server with which the PASSTHRU privilege is associated.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the PASSTHRU privilege. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

### Nickname privileges

```
REVOKE [ALL <PRIVILEGES> |  
      Privilege, ...]  
ON [Nickname]  
FROM [Forfeiter, ...] <BY ALL>
```

where:

<i>Privilege</i>	Identifies one or more nickname privileges that are to be taken from one or more users and/or groups. The following values are valid for this parameter: CONTROL, ALTER, SELECT, INSERT, UPDATE, DELETE, INDEX, and REFERENCES.
<i>Nickname</i>	Identifies by name the nickname with which all privileges specified are to be associated.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the nickname privileges specified. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

Only users who hold System Administrator (SYSADM) or Database Administrator (DBADM) authority are allowed to revoke CONTROL privilege for a nickname. For this reason, when the ALL PRIVILEGES clause is specified, all nickname privileges *except* CONTROL privilege are revoked from each *Forfeiter*; CONTROL privilege must be revoked separately.

## REVOKE SQL statement examples

Now that we've seen the basic syntax for the various forms of the REVOKE SQL statement, let's take a look at some examples.

**Example 1.** A server has both a user and a group named Q045. Remove the ability to connect to the database named SAMPLE from the group Q045:

```
CONNECT TO sample;
REVOKE CONNECT ON DATABASE FROM GROUP q045;
```

**Example 2.** Revoke all table privileges available for the table DEPARTMENT (except CONTROL privilege) from the user USER1 and the group PUBLIC:

```
REVOKE ALL PRIVILEGES ON TABLE department FROM user1, PUBLIC
```



.....

If all table privileges are revoked from the group PUBLIC, all views that reference the table will become inaccessible to the group PUBLIC. That's because SELECT privilege must be held on a table in order to access a view that references the table.

.....

**Example 3.** Take away user USER1's ability to use a user-defined function named CALC\_BONUS:

```
REVOKE EXECUTE ON FUNCTION calc_bonus FROM user1
```

**Example 4.** Take away user USER1's ability to modify information stored in the ADDRESS and HOME\_PHONE columns of the table EMP\_INFO:

```
REVOKE UPDATE(address, home_phone) ON TABLE emp_info FROM user1 BY ALL
```

**Example 5.** Take away user USER1's ability to read data stored in a table named INVENTORY:

```
REVOKE SELECT ON TABLE inventory FROM user1
```

**Example 6.** Prevent users in the group PUBLIC from adding or changing data stored in a table named EMPLOYEE:

```
REVOKE INSERT, UPDATE ON TABLE employee FROM PUBLIC
```

**Requirements for Granting and Revoking Authorities and Privileges**

Not only do authorization levels and privileges control what a user can and cannot do; they also control what authorities and privileges a user is allowed to grant and revoke. A list of the authorities and privileges that a user who has been given a specific authority level or privilege is allowed to grant and revoke can be seen in Table 8.1.

<i>Table 8.1 Requirements for Granting/Revoking Authorities and Privileges</i>		
<b>If a User Holds...</b>	<b>The User Can Grant...</b>	<b>The User Can Revoke...</b>
System Administrator (SYSADM) authority	System Control (SYSCTRL) authority System Maintenance (SYSMAINT) authority System Monitor (SYSMON) authority Database Administrator (DBADM) authority Security Administrator (SECADM) authority Load (LOAD) authority Any database privilege, including CONTROL privilege Any object privilege, including CONTROL privilege	System Control (SYSCTRL) authority System Maintenance (SYSMAINT) authority System Monitor (SYSMON) authority Database Administrator (DBADM) authority Security Administrator (SECADM) authority Load (LOAD) authority Any database privilege, including CONTROL privilege Any object privilege, including CONTROL privilege
System Control (SYSCTRL) authority	The USE table space privilege	The USE table space privilege
System Maintenance (SYSMAINT) authority	No authorities or privileges	No authorities or privileges
System Monitor (SYSMON) authority	No authorities or privileges	No authorities or privileges
Database Administrator (DBADM) authority	Any database privilege, including CONTROL privilege Any object privilege, including CONTROL privilege	Any database privilege, including CONTROL privilege Any object privilege, including CONTROL privilege
Security Administrator (SECADM) authority	No authorities or privileges	No authorities or privileges
Load (LOAD) authority	No authorities or privileges	No authorities or privileges
CONTROL privilege on an object (but no other authority)	All privileges available (with the exception of the CONTROL privilege) for the object on which the user holds CONTROL privilege	All privileges available (with the exception of the CONTROL privilege) for the object on which the user holds CONTROL privilege
A privilege on an object that was assigned with the WITH GRANT OPTION option specified	The same object privilege that was assigned with the WITH GRANT OPTION option specified.	No authorities or privileges

## **Authorities and Privileges Needed to Perform Common Tasks**

So far, we have identified the authorities and privileges that are available, and we have examined how these authorities and privileges are granted and revoked. But to use authorities and privileges effectively, you must be able to determine which authorities and privileges are appropriate for an individual user and which are not.

Often, a blanket set of authorities and privileges is assigned to an individual, based on his or her job title and job responsibilities. Then, as the individual begins to work with the database, the set of authorities and privileges he or she has is modified as appropriate. Some of the more common job titles used, along with the tasks that usually accompany them and the authorities/privileges needed to perform those tasks, can be seen in Table 8.2.

<i>Table 8.2 Common Job Titles, Tasks, and Authorities/Privileges Needed</i>		
<b>Job Title</b>	<b>Tasks</b>	<b>Authorities/Privileges Needed</b>
Department Administrator	Oversees the departmental system; designs and creates databases.	System Control (SYSCTRL) authority or System Administrator (SYSADM) authority (if the department has its own instance)
Security Administrator	Grants authorities and privileges to other users and revokes them, if necessary.	System Administrator (SYSADM) authority or Database Administrator (DBADM) authority (Security Administrator [SECADM] authority if label-based access control is used)
Database Administrator	Designs, develops, operates, safeguards, and maintains one or more databases.	Database Administrator (DBADM) authority over one or more databases and System Maintenance (SYSMAINT) authority, or in some cases System Control (SYSCTRL) authority, over the instance(s) that control the databases
System Operator	Monitors the database and performs routine backup operations. Also performs recovery operations if needed.	System Maintenance (SYSMAINT) authority or System Monitor (SYSMON) authority
Application Developer/Programmer	Develops and tests database/DB2 Database Manager application programs; may also create test tables and populate them with data.	CONNECT and CREATE_TAB privilege for one or more databases, BINDADD and BIND privilege on one or more packages, one or more schema privileges for one or more schemas, and one or more table privileges for one or more tables; CREATE_EXTERNAL_ROUTINE privilege for one or more databases may also be required
User Analyst	Defines the data requirements for an application program by examining the database structure using the system catalog views.	CONNECT privilege for one or more databases and SELECT privilege on the system catalog views

*Table 8.2 Common Job Titles, Tasks, and Authorities/Privileges Needed (continued)*

Job Title	Tasks	Authorities/Privileges Needed
End User	Executes one or more application programs.	CONNECT privilege for one or more databases and EXECUTE privilege on the package associated with each application used; if an application program contains dynamic SQL statements, SELECT, INSERT, UPDATE, and DELETE privileges for one or more tables may be needed as well
Information Center Consultant	Defines the data requirements for a query user; provides the data needed by creating tables and views and by granting access to one or more database objects.	Database Administrator (DBADM) authority for one or more databases
Query User	Issues SQL statements (usually from the Command Line Processor) to retrieve, add, update, or delete data (may also save results of queries in tables)	CONNECT privilege on one or more databases; SELECT, INSERT, UPDATE, and DELETE privilege on each table used; and CREATEIN privilege on the schema in which tables and views are to be created
Adapted from Table 78 on pages 608–609 of the <i>IBM DB2 Version 9 for Linux, UNIX, and Windows Administration Guide—Implementation</i> manual.		

## Securing Data with Label-Based Access Control (LBAC)

Earlier, we saw that authentication is performed at the operating system level to verify that users are who they say they are, and authorities and privileges control access to a database and the objects and data that reside within it. Views, which allow different users to see different presentations of the same data, can be used in conjunction with privileges to limit access to specific columns. But what if your security requirements dictate that you create and manage several hundred views? Or, more importantly, what if you want to restrict access to individual rows in a table? If you're using DB2 9, the solution for these situations is label-based access control.

So just what is label-based access control (LBAC)? LBAC is a new security feature that uses one or more security labels to control who has read access and who has write access to individual rows and/or columns in a table. The United States and many other governments use LBAC models in which hierarchical classification labels such as CONFIDENTIAL, SECRET, and TOP SECRET are assigned to data based on its sensitivity. Access to data labeled at a certain level (for example, SECRET) is restricted to those users who have been granted that level of access or higher. With

LBAC, you can construct security labels to represent any criteria your company uses to determine who can read or modify particular data values. And LBAC is flexible enough to handle the most simple to the most complex criteria.

One problem with the traditional security methods DB2 uses is that security administrators and DBAs have access to sensitive data stored in the databases they oversee. To solve this problem, LBAC-security administration tasks are isolated from all other tasks—only users with Security Administrator (SECADM) authority are allowed to configure LBAC elements.

### ***Implementing Row-Level LBAC***

Before you implement a row-level LBAC solution, you need to have a thorough understanding of the security requirement needs. Suppose you have a database that contains company sales data, and you want to control how senior executives, regional managers, and sales representatives access data stored in a table named SALES. Security requirements might dictate that access to this data should comply with these rules:

- Senior executives are allowed to view, but not update, all records in the table.
- Regional managers are allowed to view and update only records that were entered by sales representatives who report to them.
- Sales representatives are allowed to view and update only records of the sales they made.

Once the security requirements are known, you must then define the appropriate security policies and labels, create an LBAC-protected table (or alter an existing table to add LBAC protection), and grant the proper security labels to the appropriate users.

### **Defining a security label component**

Security label components represent criteria that may be used to decide whether a user should have access to specific data. Three types of security label components can exist:



- A *set* is a collection of elements (character string values) where the order in which each element appears is not important.
- An *array* is an ordered set that can represent a simple hierarchy. In an array, the order in which the elements appear is important—the first element ranks higher than the second, the second ranks higher than the third, and so on.
- A *tree* represents a more complex hierarchy that can have multiple nodes and branches.

To create security label components, you execute one of the following CREATE SECURITY LABEL COMPONENT SQL statements:

```
CREATE SECURITY LABEL COMPONENT [ComponentName]  
SET {StringConstant, ...}
```

or

```
CREATE SECURITY LABEL COMPONENT [ComponentName]  
ARRAY [StringConstant, ...]
```

or

```
CREATE SECURITY LABEL COMPONENT [ComponentName]  
TREE (StringConstant ROOT < StringConstant UNDER StringConstant >)]
```

where:

*ComponentName* Identifies the name that is to be assigned to the security label component being created.

*StringConstant* Identifies one or more string constant values that make up the valid array, set, or tree of values to be used by the security label component being created.

Thus, to create a security label component named SEC\_COMP that contains a set of values whose order is insignificant, you would execute a CREATE SECURITY LABEL COMPONENT statement that looks something like this:

```
CREATE SECURITY LABEL COMPONENT sec_comp  
SET {'CONFIDENTIAL', 'SECRET', 'TOP_SECRET'}
```

To create a security label component that contains an array of values listed from highest to lowest order, you would execute a CREATE SECURITY LABEL COMPONENT statement that looks something like this:

```
CREATE SECURITY LABEL COMPONENT sec_comp  
ARRAY ['MASTER_CRAFTSMAN', 'JOURNEYMAN', 'APPRENTICE']
```

And to create a security label component that contains a tree of values that describe a company's organizational chart, you would execute a CREATE SECURITY LABEL COMPONENT statement that looks something like this:

```
CREATE SECURITY LABEL COMPONENT sec_comp  
TREE ('EXEC_STAFF' ROOT,  
      'N_MGR' UNDER 'EXEC_STAFF',  
      'E_MGR' UNDER 'EXEC_STAFF',  
      'S_MGR' UNDER 'EXEC_STAFF',  
      'W_MGR' UNDER 'EXEC_STAFF',  
      'C_MGR' UNDER 'EXEC_STAFF',  
      'SALES_REP1' UNDER 'N_MGR',  
      'SALES_REP2' UNDER 'W_MGR')
```

## Defining a security policy

Security policies determine exactly how a table is to be protected by LBAC. Specifically, a security policy identifies the following:

- What security label components will be used in the security labels that will be part of the policy
- What rules will be used when security label components are compared (at this time, only one set of rules is supported: DB2LBACRULES)
- Which optional behaviors will be used when accessing data protected by the policy

Every LBAC-protected table must have one (and only one) security policy associated with it. Rows and columns in that table can be protected only with security labels that are part of that security policy; all protected data access must

adhere to the rules of that policy. You can have multiple security policies within a single database, but you can't have more than one security policy protecting any given table.

To create a security policy, execute the CREATE SECURITY POLICY SQL statement as follows:

```
CREATE SECURITY POLICY [PolicyName]
COMPONENTS [ComponentName ,...]
WITH DB2LBACRULES
<[OVERRIDE | RESTRICT] NOT AUTHORIZED
    WRITE SECURITY LABEL>
```

where:

*PolicyName*            Identifies the name that is to be assigned to the security policy being created.

*ComponentName*    Identifies, by name, one or more security label components that are to be part of the security policy being created.

The [OVERRIDE | RESTRICT] NOT AUTHORIZED WRITE SECURITY LABEL option specifies the action to be taken when a user who is not authorized to write the security label explicitly specified with INSERT and UPDATE statements attempts to write data to the protected table. By default, the value of a user's security label, rather than an explicitly specified security label, is used for write access during insert and update operations (OVERRIDE NOT AUTHORIZED WRITE SECURITY LABEL). If the RESTRICT NOT AUTHORIZED WRITE SECURITY LABEL option is used, insert and update operations will fail if the user isn't authorized to write the explicitly specified security label to the protected table.

Therefore, to create a security policy named SEC\_POLICY that is based on the SEC\_COMP security label component created earlier, you would execute a CREATE SECURITY POLICY statement that look something like this:

```
CREATE SECURITY POLICY sec_policy
COMPONENTS sec_comp
WITH DB2LBACRULES
```

## Defining security labels

Security labels describe a set of security criteria and are used to protect data against unauthorized access or modification. Security labels are granted to users who are allowed to access or modify protected data; when users attempt to access or modify protected data, their security label is compared to the security label protecting the data to determine whether the access or modification is allowed. Every security label is part of exactly one security policy, and a security label must exist for each security label component found in the security policy.

Security labels are created by executing the CREATE SECURITY LABEL SQL statement. The syntax for this statement is:

```
CREATE SECURITY LABEL [LabelName]  
[COMPONENT [ComponentName] [StringConstant] ,...]
```

where:

- |                       |   |
|-----------------------|---|
| <i>LabelName</i>      | Identifies the name that is to be assigned to the security label being created. The name specified must be qualified with a security policy name and must not match an existing security label for the security policy specified. |
| <i>ComponentName</i>  | Identifies, by name, a security label component that is part of the security policy specified as the qualifier for the <i>LabelName</i> parameter.  |
| <i>StringConstant</i> | Identifies one or more string constant values that are valid elements of the security label component specified in the <i>ComponentName</i> parameter.  |

Thus, to create a set of security labels for the security policy named SEC\_POLICY that was created earlier, you would execute a set of CREATE SECURITY LABEL statements that looks something like this:

```

CREATE SECURITY LABEL sec_policy.exec_staff
COMPONENT sec_comp 'EXEC_STAFF'

CREATE SECURITY LABEL sec_policy.n_mgr
COMPONENT sec_comp 'N_MGR'

CREATE SECURITY LABEL sec_policy.e_mgr
COMPONENT sec_comp 'E_MGR'

CREATE SECURITY LABEL sec_policy.s_mgr
COMPONENT sec_comp 'S_MGR'

CREATE SECURITY LABEL sec_policy.w_mgr
COMPONENT sec_comp 'W_MGR'

CREATE SECURITY LABEL sec_policy.c_mgr
COMPONENT sec_comp 'C_MGR'

CREATE SECURITY LABEL sec_policy.sales_rep1
COMPONENT sec_comp 'SALES_REP1'

CREATE SECURITY LABEL sec_policy.sales_rep2
COMPONENT sec_comp 'SALES_REP2'

```

### **Creating a LBAC-protected table**

Once you have defined the security policy and labels needed to enforce your security requirements, you're ready to create a table and configure it for LBAC protection. To configure a new table for row-level LBAC protection, you include a column with the data type `DB2SECURITYLABEL` in the table's definition and associate a security policy with the table using the `SECURITY POLICY` clause of the `CREATE TABLE` SQL statement.

So to create a table named `SALES` and configure it for row-level LBAC protection using the security policy named `SEC_POLICY` created earlier, you would execute a `CREATE TABLE` statement that looks something like this:

```

CREATE TABLE corp.sales (
    sales_rec_id    INTEGER NOT NULL,
    sales_date      DATE WITH DEFAULT,
    sales_rep       INTEGER,
    region          VARCHAR(15),
    manager         INTEGER,
    sales_amt       DECIMAL(12,2),
    margin          DECIMAL(12,2),
    sec_label       DB2SECURITYLABEL)
SECURITY POLICY sec_policy

```

To configure an existing table named SALES for row-level LBAC protection using a security policy named SEC\_POLICY, you would execute an ALTER TABLE statement that looks like this instead:

```
ALTER TABLE corp.sales
  ADD COLUMN sec_label DB2SECURITYLABEL
  ADD SECURITY POLICY sec_policy
```

However, before you can execute such an ALTER TABLE statement, you must be granted a security label for write access that is part of the security policy that will be used to protect the table (which, in this case is SEC\_POLICY). Otherwise, you won't be able to create the DB2SECURITYLABEL column.

### Granting security labels to users

Once the security policy and labels needed to enforce your security requirements have been defined, and a table has been enabled for LBAC protection, you must grant the proper security labels to the appropriate users and indicate whether they are to have read access, write access, or full access to data that is protected by that label. Security labels are granted to users by executing a special form of the GRANT SQL statement. The syntax for this form of the GRANT statement is:

```
GRANT SECURITY LABEL [LabelName]
  TO USER [UserName]
  [FOR ALL ACCESS | FOR READ ACCESS | FOR WRITE ACCESS]
```

where:

*LabelName*            Identifies the name of an existing security label. The name specified must be qualified with the security policy name that was used when the security label was created.

*UserName*            Identifies the name of the user to which the security label is to be granted.

Thus, to give a user named USER1 the ability to read data protected by the security label SEC\_POLICY.EXEC\_STAFF, you would execute a GRANT statement that looks like this:

```
GRANT SECURITY LABEL sec_policy.exec_staff
  TO USER user1 FOR READ ACCESS
```

### Putting row-level LBAC into action

To enforce the security requirements listed earlier, we must first give users the ability to perform DML operations against the SALES table by executing the following GRANT statements, as a user with SYSADM or DBADM authority:

```
GRANT ALL PRIVILEGES ON TABLE corp.sales TO exec_staff;
GRANT ALL PRIVILEGES ON TABLE corp.sales TO n_manager;
GRANT ALL PRIVILEGES ON TABLE corp.sales TO e_manager;
GRANT ALL PRIVILEGES ON TABLE corp.sales TO s_manager;
GRANT ALL PRIVILEGES ON TABLE corp.sales TO w_manager;
GRANT ALL PRIVILEGES ON TABLE corp.sales TO c_manager;
GRANT ALL PRIVILEGES ON TABLE corp.sales TO sales_rep1;
GRANT ALL PRIVILEGES ON TABLE corp.sales TO sales_rep2;
```

Next, we must grant the proper security labels to the appropriate users and indicate whether they are to have read access, write access, or full access to data that is protected by that label. This is done by executing the following GRANT statements, this time as a user with SECADM authority:

```
GRANT SECURITY LABEL sec_policy.exec_staff
TO USER exec_staff FOR READ ACCESS;

GRANT SECURITY LABEL sec_policy.n_mgr
TO USER n_manager FOR ALL ACCESS;

GRANT SECURITY LABEL sec_policy.e_mgr
TO USER e_manager FOR ALL ACCESS;

GRANT SECURITY LABEL sec_policy.s_mgr
TO USER s_manager FOR ALL ACCESS;

GRANT SECURITY LABEL sec_policy.w_mgr
TO USER w_manager FOR ALL ACCESS;

GRANT SECURITY LABEL sec_policy.c_mgr
TO USER c_manager FOR ALL ACCESS;

GRANT SECURITY LABEL sec_policy.sales_rep1
TO USER sales_rep1 FOR ALL ACCESS;

GRANT SECURITY LABEL sec_policy.sales_rep2
TO USER sales_rep2 FOR ALL ACCESS;
```

Now, suppose user SALES\_REP1 adds three rows to the SALES table by executing the following SQL statements:

```
INSERT INTO corp.sales VALUES (1, DEFAULT, 1, 'NORTH', 5,  
    1000.50, 500.00,  
    SECLABEL_BY_NAME('SEC_POLICY', 'SALES_REP1'));  
  
INSERT INTO corp.sales VALUES (2, DEFAULT, 1, 'NORTH', 5,  
    2000.00, 400.00,  
    SECLABEL_BY_NAME('SEC_POLICY', 'SALES_REP1'));  
  
INSERT INTO corp.sales VALUES (3, DEFAULT, 1, 'NORTH', 5,  
    4500.90, 850.00,  
    SECLABEL_BY_NAME('SEC_POLICY', 'SALES_REP1'));
```

Because SALES\_REP1 has been given read/write access to the table using the SEC\_POLICY.SALES\_REP1 security label, the statements execute successfully. Next, user SALES\_REP2 adds two additional rows to the SALES table by executing the following SQL statements:

```
INSERT INTO corp.sales VALUES (4, DEFAULT, 1, 'WEST', 20,  
    1000.50, 500.00,  
    SECLABEL_BY_NAME('SEC_POLICY', 'SALES_REP2'));  
  
INSERT INTO corp.sales VALUES (5, DEFAULT, 1, 'WEST', 20,  
    3200.00, 600.00,  
    SECLABEL_BY_NAME('SEC_POLICY', 'SALES_REP2'));
```

SALES\_REP2 has also been given read/write access to the table using the SEC\_POLICY.SALES\_REP2 security label, so the rows are successfully inserted.

Now, when user EXEC\_STAFF queries the SALES table, all five records entered will appear (because the security label SEC\_POLICY.EXEC\_STAFF is the highest level in the security policy's security label component tree). However, if user EXEC\_STAFF attempts to insert additional records or update an existing record, an error will be generated because user EXEC\_STAFF is allowed only to read the data (only read access was granted).

When user N\_MANAGER queries the table, only records entered by the user SALES\_REP1 will be displayed; the user W\_MANAGER will see only records entered by the user SALES\_REP2; and the users E\_MANAGER, S\_MANAGER, and C\_MANAGER will not see any records at all. (SALES\_REP1 reports to N\_MANAGER, SALES\_REP2 reports to W\_MANAGER; no other managers have a sales representative reporting to them.)



And finally, when SALES\_REP1 or SALES\_REP2 queries the SALES table, they will see only the records they personally entered. Likewise, they can update only the records they entered.

### ***Implementing Column-Level LBAC***

To illustrate how column-level LBAC is employed, let's assume you want to control how Human Resources (HR) staff members, managers, and employees are going to access data stored in a table named EMPLOYEES. For this scenario, the security requirements are as follows:

- Name, gender, department, and phone number information can be viewed by all employees.
- Hire date, salary, and bonus information (in addition to name, gender, department, and phone number information) can be seen only by managers and HR staff members.
- Employee ID and Social Security Number information can be seen only by HR staff members. Additionally, HR staff members are the only users who can create and modify employee records.

Once again, after the security requirements have been identified, the next steps are to define the appropriate security component, policies, and labels; create the table that will house the data; alter the table to add LBAC protection; and grant the proper security labels to the appropriate users.

### **Defining security label components, security policies, and security labels**

Because an array of values, listed from highest to lowest order, would be the best way to implement the security requirements just outlined, you could create the security component needed by executing a CREATE SECURITY LABEL COMPONENT statement (as a user with SECADM authority) that looks something like this:

```
CREATE SECURITY LABEL COMPONENT sec_comp
ARRAY ['CONFIDENTIAL', 'CLASSIFIED', 'UNCLASSIFIED']
```

After the appropriate security label component has been created, you can create a security policy named SEC\_POLICY that is based on the SEC\_COMP security label component by executing a CREATE SECURITY POLICY statement (as a user with SECADM authority) that looks like this:

```
CREATE SECURITY POLICY sec_policy
COMPONENTS sec_comp
WITH DB2LBACRULES
```

Earlier, we saw that security labels are granted to users who are allowed to access or modify LBAC-protected data; when users attempt to access or modify protected data, their security label is compared to the security label protecting the data to determine whether the access or modification is allowed. But before security labels can be granted, they must first be defined. To create a set of security labels for the security policy named SEC\_POLICY that was just created, you would execute the following set of CREATE SECURITY LABEL statements (as a user with SECADM authority):

```
CREATE SECURITY LABEL sec_policy.confidential
COMPONENT sec_comp 'CONFIDENTIAL'

CREATE SECURITY LABEL sec_policy.classified
COMPONENT sec_comp 'CLASSIFIED'

CREATE SECURITY LABEL sec_policy.unclassified
COMPONENT sec_comp 'UNCLASSIFIED'
```

Keep in mind that every security label is part of exactly one security policy, and a security label must exist for each security label component found in that security policy.

### **Creating a LBAC-protected table and granting privileges and security labels to users**

Earlier, we saw that in order to configure a new table for row-level LBAC protection, you must associate a security policy with the table being created with the SECURITY POLICY clause of the CREATE TABLE SQL statement. The same is true if column-level LBAC protection is desired. Therefore, to create a table named EMPLOYEES and associate it with a security policy named SEC\_POLICY, you would need to execute a CREATE TABLE statement that looks something like this:

```
CREATE TABLE hr.employees (
    emp_id    INTEGER NOT NULL,
    f_name    VARCHAR(20),
    l_name    VARCHAR(20),
    gender    CHAR(1),
    hire_date DATE WITH DEFAULT,
    dept_id   CHAR(5),
    phone     CHAR(14),
    ssn       CHAR(12),
    salary    DECIMAL(12,2),
    bonus     DECIMAL(12,2))
SECURITY POLICY sec_policy
```

Then, in order to enforce the security requirements identified earlier, you must give users the ability to perform the appropriate DML operations against the EMPLOYEES table. This is done by executing the following GRANT SQL statements (as a user with SYSADM or DBADM authority):

```
GRANT ALL PRIVILEGES ON TABLE hr.employees TO hr_staff;
GRANT SELECT ON TABLE hr.employees TO manager1;
GRANT SELECT ON TABLE hr.employees TO employee1;
```

Finally, you must grant the proper security label to the appropriate users and indicate whether they are to have read access, write access, or full access to data that is protected by that label. This is done by executing a set of GRANT statements (as a user with SECADM authority) that looks something like this:

```
GRANT SECURITY LABEL sec_policy.confidential
TO USER hr_staff FOR ALL ACCESS;

GRANT SECURITY LABEL sec_policy.classified
TO USER manager1 FOR READ ACCESS;

GRANT SECURITY LABEL sec_policy.unclassified
TO USER employee1 FOR READ ACCESS;
```

## **Creating LBAC-protected columns**

Once you've defined the security policy and labels needed to enforce your security requirements and have granted the appropriate privileges and security labels to users, you are ready to modify the table associated with the security policy and configure its columns for column-level LBAC protection. This is done by executing an ALTER TABLE statement that looks something like this:

```
ALTER TABLE hr.employees
  ALTER COLUMN emp_id SECURED WITH confidential
  ALTER COLUMN f_name SECURED WITH unclassified
  ALTER COLUMN l_name SECURED WITH unclassified
  ALTER COLUMN gender SECURED WITH unclassified
  ALTER COLUMN hire_date SECURED WITH classified
  ALTER COLUMN dept_id SECURED WITH unclassified
  ALTER COLUMN phone SECURED WITH unclassified
  ALTER COLUMN ssn SECURED WITH confidential
  ALTER COLUMN salary SECURED WITH classified
  ALTER COLUMN bonus SECURED WITH classified;
```

Here is where things get a little tricky. If you try to execute the ALTER TABLE statement shown as a user with SYSADM or SECADM authority, the operation will fail, and you will be presented with an error message that looks something like this:

```
SQL20419N For table "EMPLOYEES", authorization ID " " does not have LBAC
credentials that allow using the security label "CONFIDENTIAL" to protect
column "EMP_ID".  SQLSTATE=42522
```

That's because the only user who can secure a column with the "CONFIDENTIAL" security label is a user who has been granted *write access* to data that is protected by that label. In our scenario, this is the user HR\_STAFF. So what happens when user HR\_STAFF attempts to execute the preceding ALTER TABLE statement? Now a slightly different error message is produced:

```
SQL20419N For table "EMPLOYEES", authorization ID "HR_STAFF" does not
have LBAC credentials that allow using the security label "UNCLASSIFIED"
to protect column "F_NAME".  SQLSTATE=42522
```

Why? Because, by default, the LBAC rules set associated with the security policy assigned to the EMPLOYEES table allows the user HR\_STAFF to write data only to columns or rows that are protected by the same security label that he/she has been granted.

## **DB2LBACRULES rules**

An LBAC rule set is a predefined set of rules that is used when comparing security labels. Currently, only one LBAC rule set is supported (DB2LBACRULES), and as we have just seen, this rule set prevents both write-up and write-down behavior. (Write-up and write-down apply only to ARRAY security label components and only

to write access.) Write-up is when the security label protecting data to which you are attempting to write is higher than the security label you have been granted; write-down is when the security label protecting data is lower.

Which rules are actually used when two security labels are compared is dependent on the type of component used (SET, ARRAY, or TREE) and the type of access being attempted (read or write). Table 8.3 lists the rules found in the DB2LBACRULES rules set, identifies which component each rule is used for, and describes how the rule determines if access is to be blocked.

Table 8.3 Summary of the DB2LBACRULES Rules				
Rule Name	Component	Component	Access	Access is blocked when this condition is met
DB2LBACREADARRAY		ARRAY	Read	The user's security label is lower than the protecting security label.
DB2LBACREADSET		SET	Read	There are one or more protecting security labels that the user does not hold.
DB2LBACREADTREE		TREE	Read	None of the user's security labels are equal to or an ancestor of one of the protecting security labels.
DB2LBACWRITEARRAY		ARRAY	Write	The user's security label is higher than the protecting security label or lower than the protecting security label.
DB2LBACWRITESET		SET	Write	There are one or more protecting security labels that the user does not hold.
DB2LBACWRITETREE		TREE	Write	None of the user's security labels are equal to or an ancestor of one of the protecting security labels.
Adapted from Table 78 on pages 608–609 of the <i>IBM DB2 Version 9 for Linux, UNIX, and Windows Administration Guide—Implementation</i> manual.				

## Granting exemptions

So how can the remaining columns in the EMPLOYEES table be secured with the appropriate security labels? The Security Administrator must first grant user HR\_STAFF an exemption to one or more security policy rules. When a user holds an exemption on a particular security policy rule, that rule is not enforced when the user attempts to access data that is protected by that security policy.

Security policy exemptions are granted by executing the GRANT EXEMPTION ON RULE SQL statement (as a user with SECADM authority). The syntax for this statement is:

```
CREATE EXEMPTION ON RULE [Rule] ,...  
FOR [PolicyName]  
TO USER [UserName]
```

where:

*Rule* Identifies one or more DB2LBACRULES security policy rules for which exemptions are to be given. The following values are valid for this parameter: DB2LBACREADARRAY, DB2LBACREADSET, DB2LBACREADTREE, DB2LBACWRITEARRAY WRITEDOWN, DB2LBACWRITEARRAY WRITEUP, DB2LBACWRITESSET, DB2LBACWRITETREE, and ALL. (If an exemption is held for every security policy rule, the user will have complete access to all data protected by that security policy.)

*PolicyName* Identifies the security policy for which the exemption is to be granted.

*UserName* Identifies the name of the user to which the exemptions specified are to be granted.

Thus, to grant an exemption to the DB2LBACWRITEARRAY rule in the security policy named SEC\_POLICY created earlier to a user named HR\_STAFF, you would execute a GRANT EXEMPTION statement that looks something like this:

```
GRANT EXEMPTION ON RULE DB2LBACWRITEARRAY  
WRITEDOWN FOR sec_policy  
TO USER hr_staff
```

Once this exemption is granted along with the appropriate security label, user HR\_STAFF will then be able to execute the ALTER TABLE statement shown earlier without generating an error. (Alternately, the following CREATE TABLE statement could be used to create the EMPLOYEES table and protect each column with the appropriate security label, provided user HR\_STAFF has the privileges needed to create the table.)

```
CREATE TABLE hr.employees (
    emp_id      INTEGER NOT NULL SECURED WITH confidential,
    f_name      VARCHAR(20) SECURED WITH unclassified,
    l_name      VARCHAR(20) SECURED WITH unclassified,
    gender      CHAR(1) SECURED WITH unclassified,
    hire_date   DATE WITH DEFAULT SECURED WITH classified,
    dept_id     CHAR(5) SECURED WITH unclassified,
    phone       CHAR(14) SECURED WITH unclassified,
    ssn         CHAR(12) SECURED WITH confidential,
    salary      DECIMAL(12,2) SECURED WITH classified,
    bonus       DECIMAL(12,2) SECURED WITH classified)
SECURITY POLICY sec_policy
```

### Putting column-level LBAC into action

Now that we have established a column-level LBAC environment, let's see what happens when different users try to access data stored in protected columns of the EMPLOYEES table. Suppose the user HR\_STAFF adds three rows to the EMPLOYEES table by executing the following SQL statements.

```
INSERT INTO hr.employees VALUES(1, 'John', 'Doe', 'M',
    DEFAULT, 'A01', '919-555-1212', '111-22-3333',
    42000.50, 8500.00);

INSERT INTO hr.employees VALUES(2, 'Jane', 'Doe', 'F',
    DEFAULT, 'B02', '919-555-3434', '222-33-4444',
    38000.75, 5000.00);

INSERT INTO hr.employees VALUES(3, 'Paul', 'Smith', 'M',
    DEFAULT, 'C03', '919-555-5656', '333-44-5555',
    39250.00, 3500.00);
```

User HR\_STAFF1 has been given read/write access to all columns in the table (with the SEC\_POLICY.CLASSIFIED security label and the DB2LBACWRITEARRAY WRITEDOWN exemption), so the statements execute successfully. If user HR\_STAFF attempts to query the table, he or she will be able to see every column and every row because he or she has been granted the highest security level in the array.

Now, when user MANAGER1 attempts to read every column in the table, an error will be generated stating that he or she does not have “READ” access to the column “SSN.” However, MANAGER1 will be able to execute the following query because he or she has been granted read access to each column specified:

```
SELECT f_name, l_name, hire_date, salary, bonus
FROM hr.employees
```

Now, if user EMPLOYEE1 attempts to execute the same query, an error will be generated stating that he or she does not have “READ” access to the column “BONUS.” But an attempt by EMPLOYEE1 to execute the following query will be successful:

```
SELECT f_name, l_name, gender, dept_id, phone
FROM hr.employees
```

Additionally, if user MANAGER1 or user EMPLOYEE1 attempt to insert additional records or update existing information, they will get an error stating they do not have permission to perform the operation against the table.

### **Combining Row-Level and Column-Level LBAC**

There may be times when you would like to limit an individual user’s access to a specific combination of rows and columns. When this is the case, you must include a column with the data type DB2SECURITYLABEL in the table’s definition, add the SECURED WITH [*SecurityLabel*] option to each column in the table’s definition, and associate a security policy with the table using the SECURITY POLICY clause of the CREATE TABLE SQL statement or the ADD SECURITY POLICY clause of the ALTER TABLE statement. Typically, you will also create two security label components—one for rows and one for columns—and use both components to construct the security policy and labels needed.

For example, assume that you created two security label components by executing the following commands:

```
CREATE SECURITY LABEL COMPONENT scom_level
ARRAY ['CONFIDENTIAL', 'CLASSIFIED', 'UNCLASSIFIED'];

CREATE SECURITY LABEL COMPONENT scom_country
TREE ('NA' ROOT, 'CANADA' UNDER 'NA', 'USA' UNDER 'NA');
```

You would then create a security policy by executing a CREATE SECURITY POLICY command that looks something like this:



```
CREATE SECURITY POLICY sec_policy  
COMPONENTS scom_level, scom_country  
WITH DB2LBACRULES
```

Then you could create corresponding security labels by executing commands that look something like this:

```
CREATE SECURITY LABEL sec_policy.confidential  
COMPONENT scom_level 'CONFIDENTIAL';  
  
CREATE SECURITY LABEL sec_policy.uc_canada  
COMPONENT scom_level 'UNCLASSIFIED'  
COMPONENT scom_country 'CANADA';  
  
CREATE SECURITY LABEL sec_policy.uc_us  
COMPONENT scom_level 'UNCLASSIFIED'  
COMPONENT scom_country 'USA';
```

Finally, after associating the appropriate security labels with individual columns, you would grant the proper security label to each user and conduct a few tests to ensure data access is controlled as expected.

## Practice Questions

### Question 1

---

Which of the following is NOT a security mechanism that is used to control access DB2 data?

- ☐ A. Authorization
- ☐ B. Privileges
- ☐ C. Validation
- ☐ D. Authentication

### Question 2

---

Which of the following identifies which users have SYSMAINT authority?

- ☐ A. The DB2 registry
- ☐ B. The DB2 Database Manager configuration
- ☐ C. The database configuration
- ☐ D. The system catalog

### Question 3

---

Which of the following database privileges are NOT automatically granted to the group PUBLIC when a database is created?

- ☐ A. CONNECT
- ☐ B. BINDADD
- ☐ C. IMPLICIT\_SCHEMA
- ☐ D. CREATE\_EXTERNAL\_ROUTINE

---

**Question 4**

User USER1 needs to remove a view named ORDERS\_V, which is based on a table named ORDERS, from the SALES database. Assuming user USER1 does not hold any privileges, which of the following privileges must be granted before user USER1 will be allowed to drop the view?

- ☐ A. DROP privilege on the ORDERS\_V view
- ☐ B. CONTROL privilege on the ORDERS table
- ☐ C. DROP privilege on the ORDERS\_V view
- ☐ D. CONTROL privilege on the ORDERS\_V view

---

**Question 5**

Which of the following identifies how authentication is performed for an instance?

- ☐ A. The operating system used by the instance
- ☐ B. The communications configuration used by the instance
- ☐ C. The DB2 registry
- ☐ D. The DB2 Database Manager configuration

---

**Question 6**

After the following SQL statement is executed:

```
GRANT ALL PRIVILEGES ON TABLE employee TO USER user1
```

Assuming user USER1 has no other authorities or privileges, which of the following actions is USER1 allowed to perform?

- ☐ A. Drop an index on the EMPLOYEE table.
- ☐ B. Grant all privileges on the EMPLOYEE table to other users.
- ☐ C. Alter the table definition.
- ☐ D. Drop the EMPLOYEE table.

**Question 7**

---

A user named USER1 is granted DBADM authority. Assuming no other authorities/privileges have been granted and all privileges have been revoked from the group PUBLIC, if the following SQL statement is executed:

```
REVOKE DBADM ON DATABASE FROM user1
```

What authorities/privileges will user USER1 have?

- ☐ A. None
- ☐ B. CONNECT
- ☐ C. SYSCTRL
- ☐ D. EXECUTE

**Question 8**

---

User USER1 wants to call an SQL stored procedure that dynamically retrieves data from a table. Which two privileges must user USER1 have in order to invoke the stored procedure?

- ☐ A. EXECUTE privilege on the stored procedure.
- ☐ B. CALL privilege on the stored procedure.
- ☐ C. SELECT privilege on the table the stored procedure retrieves data from.
- ☐ D. EXECUTE privilege on the package for the stored procedure.
- ☐ E. SELECT privilege on the stored procedure.

**Question 9**

---

Which of the following privileges allow a user to remove a foreign key that has been defined for a table?

- ☐ A. ALTER privilege on the table.
- ☐ B. DELETE privilege on the table.
- ☐ C. DROP privilege on the table.
- ☐ D. UPDATE privilege on the table.

**Question 10**

Which of the following privileges allows a user to generate a package for an embedded SQL application and store it in a database?

- ☐ A. BIND
- ☐ B. BINDADD
- ☐ C. CREATE\_EXTERNAL\_ROUTINE
- ☐ D. CREATE\_NOT\_FENCED\_ROUTINE

**Question 11**

Which of the following statements is NOT true about DB2 security?

- ☐ A. A custom security plug-in must be created if Microsoft Active Directory will be used to validate users.
- ☐ B. Only users with Security Administrator authority are allowed to grant and revoke SETSESSIONUSER privileges.
- ☐ C. Users and groups must exist before they can be granted privileges.
- ☐ D. If a user holding SELECT privilege on a table creates a view based on that table and their SELECT privilege is later revoked, the view will become inoperative.

**Question 12**

User USER1 has the privileges needed to invoke a stored procedure named GEN\_RESUME. User USER2 needs to be able to call the procedure—user USER1 and all members of the group PUBLIC should no longer be allowed to call the procedure. Which of the following statement(s) can be used to accomplish this?

- ☐ A. GRANT EXECUTE ON ROUTINE gen\_resume TO user2 EXCLUDE user1, PUBLIC
- ☐ B. GRANT EXECUTE ON PROCEDURE gen\_resume TO user2;  
REVOKE EXECUTE ON PROCEDURE gen\_resume FROM user1, PUBLIC;
- ☐ C. GRANT CALL ON ROUTINE gen\_resume TO user2 EXCLUDE user1, PUBLIC
- ☐ D. GRANT CALL ON PROCEDURE gen\_resume TO user2;  
REVOKE CALL ON PROCEDURE gen\_resume FROM user1, PUBLIC;

### Question 13

---

Which of the following is NOT used to limit access to individual rows in a table that is protected by Label-Based Access Control (LBAC)?

- ☐ A. One or more security profiles
- ☐ B. A security policy
- ☐ C. One or more security labels
- ☐ D. A DB2SECURITYLABEL column

### Question 14

---

Which of the following statements is NOT true about Label-Based Access Control (LBAC)?

- ☐ A. LBAC can be used to restrict access to individual rows and columns.
- ☐ B. Users that have been granted different LBAC security labels will get different results when they execute the same query.
- ☐ C. Only users with SYSADM or SECADM authority are allowed to create security policies and security labels.
- ☐ D. Security label components represent criteria that may be used to decide whether a user should have access to specific data.

### Question 15

---

Which of the following SQL statements allows a user named USER1 to write to LBAC-protected columns that have been secured with a LBAC label that indicates a lower level of security than that held by USER1 ?

- ☐ A. GRANT EXECPTION ON RULE DB2LBACWRITEARRAY WRITEDOWN FOR sec\_policy TO USER user1
- ☐ B. GRANT EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEDOWN FOR sec\_policy TO USER user1
- ☐ C. GRANT EXECPTION ON RULE DB2LBACWRITEARRAY WRITEUP FOR sec\_policy TO USER user1
- ☐ D. GRANT EXEMPTION ON RULE DB2LBACWRITEARRAY WRITEUP FOR sec\_policy TO USER user1

## Answers

### Question 1

---

The correct answer is **C**. The first security portal most users must pass through on their way to gaining access to a DB2 instance or database is a process known as authentication. The purpose of authentication is to verify that users really are who they say they are. Once a user has been authenticated and an attachment to an instance or a connection to a database has been established, the DB2 Database Manager evaluates any authorities and privileges that have been assigned to the user to determine what operations the user is allowed to perform. *Privileges* convey the rights to perform certain actions against specific database resources (such as tables and views). *Authorities* convey a set of privileges or the right to perform high-level administrative and maintenance/utility operations on an instance or a database.

### Question 2

---

The correct answer is **B**. Like System Administrator (SYSADM), System Control (SYSCTRL), and System Monitor (SYSMON) authority, System Maintenance (SYSMAINT) authority can only be assigned to a group. This assignment is made by storing the appropriate group name in the *sysmaint\_group* parameter of the DB2 Database Manager configuration file that is associated with a particular instance.

### Question 3

---

The correct answer is **D**. To connect to and work with a particular database, a user must have the authorities and privileges needed to use that database. Therefore, whenever a new database is created, unless otherwise specified, the following authorities and privileges are automatically granted:

- Database Administrator (DBADM) authority, along with CONNECT, CREATETAB, BINDADD, CREATE\_NOT\_FENCED, IMPLICIT\_SCHEMA, and LOAD privileges, are granted to the user who created the database.
- USE privilege on the table space USERSPACE1 is granted to the group PUBLIC.
- CONNECT, CREATETAB, BINDADD, and IMPLICIT\_SCHEMA privileges are granted to the group PUBLIC.
- SELECT privilege on each system catalog table is granted to the group PUBLIC.

- EXECUTE privilege on all procedures found in the SYSIBM schema is granted to the group PUBLIC.
- EXECUTE WITH GRANT privilege on all functions found in the SYSFUN schema is granted to the group PUBLIC.
- BIND and EXECUTE privileges for each successfully bound utility are granted to the group PUBLIC.

(For more information, refer to Chapter 3 – “Data Placement”)

---

**Question 4**

The correct answer is **D**. The CONTROL view privilege provides a user with every view privilege available, allows the user to remove (drop) the view from the database, and gives the user the ability to grant and revoke one or more view privileges (except the CONTROL privilege) to/from other users and groups.

---

**Question 5**

The correct answer is **D**. Because DB2 can reside in environments comprised of multiple clients, gateways, and servers, each of which may be running on a different operating system, deciding where authentication is to take place is determined by the value assigned to the *authentication* parameter in each DB2 Database Manager configuration file. The value assigned to this parameter, often referred to as the authentication type, is set initially when an instance is created. (On the server side, the authentication type is specified during the instance creation process; on the client side, the authentication type is specified when a remote database is cataloged.) Only one authentication type exists for each instance, and it controls access to that instance, as well as to all databases that fall under that instance’s control.



**Question 6**

---

The correct answer is **C**. The GRANT ALL PRIVILEGES statement gives USER1 the following privileges for the EMPLOYEE table: ALTER, SELECT, INSERT, UPDATE, DELETE, INDEX, and REFERENCES. To drop an index, USER1 would need CONTROL privilege on the index—not the table the index is based on; USER1 cannot grant privileges to other users because the WITH GRANT OPTION clause was not specified with the GRANT ALL PRIVILEGES statement used to give USER1 table privileges; and in order to drop the EMPLOYEE table, USER1 would have to have CONTROL privilege on the table—CONTROL privilege is not granted with the GRANT ALL PRIVILEGES statement.

**Question 7**

---

The correct answer is **B**. When a user is given Database Administrator (DBADM) authority for a particular database, they automatically receive all database privileges available for that database as well (CONNECT, CONNECT\_QUIESCE, IMPLICIT\_SCHEMA, CREATETAB, BINDADD, CREATE\_EXTERNAL\_ROUTINE, CREATE\_NOT\_FENCED\_ROUTINE, and LOAD). When Database Administrator authority is revoked, all other database authorities that were implicitly and automatically granted when DBADM authority was granted are not automatically revoked. The same is true for privileges held on objects in the database.

**Question 8**

---

The correct answers are **A** and **C**. Before a user can invoke a routine (user-defined function, stored procedure, or method) they must hold both EXECUTE privilege on the routine and any privileges required by that routine. Thus, in order to execute a stored procedure that queries a table, a user must hold both EXECUTE privilege on the stored procedure and SELECT privilege on the table the query is ran against.

Package privileges control what users can and cannot do with a particular package. (A package is an object that contains the information needed by the DB2 Database Manager to process SQL statements in the most efficient way possible on behalf of an embedded SQL application.)

**Question 9**

---

The correct answer is **A**. The ALTER table privilege allows a user to execute the ALTER TABLE SQL statement against a table. In other words, this privilege allows a user to add columns to the table, add or change comments associated with the table or any of its columns, create or drop a primary key for the table, create or drop a unique constraint for the table, create or drop a check constraint for the table, create or drop a referential constraint for the table, and create triggers for the table (provided the user holds the appropriate privileges for every object referenced by the trigger).

The UPDATE privilege allows a user to execute the UPDATE SQL statement against the table. In other words, this privilege allows a user to modify data in the table. The DELETE privilege allows a user to execute the DELETE SQL statement against the table. In other words, it allows a user to remove rows of data from the table.

**Question 10**

---

The correct answer is **B**. The BINDADD database privilege allows a user to create packages in the database (by precompiling embedded SQL application source code files against the database or by binding application bind files to the database).

The BIND package privilege allows a user to rebind or add new package versions to a package that has already been bound to a database. (In addition to the BIND package privilege, a user must hold the privileges needed to execute the SQL statements that make up the package before the package can be successfully rebound.) The CREATE\_EXTERNAL\_ROUTINE database privilege allows a user to create user-defined functions (UDFs) and/or procedures and store them in the database so that they can be used by other users and applications. The CREATE\_NOT\_FENCED\_ROUTINE database privilege allows a user to create unfenced UDFs and/or procedures and store them in the database. (Unfenced UDFs and stored procedures are UDFs/procedures that are considered “safe” enough to be run in the DB2 Database Manager operating environment’s process or address space. Unless a UDF/procedure is registered as unfenced, the DB2 Database Manager insulates the UDF/procedure’s internal resources in such a way that they cannot be run in the DB2 Database Manager’s address space.)

## Question 11

The correct answer is **C**. The GRANT statement does not check to ensure that the names of users and/or groups that are to be granted authorities and privileges are valid. Therefore, it is possible to grant authorities and privileges to users and groups that do not exist.

## Question 12

The correct answer is **B**. The syntax used to grant the only stored procedure privilege available is:

```
GRANT EXECUTE ON [RoutineName] |
                    [PROCEDURE <SchemaName.> *]
TO [Recipient, ...]
<WITH GRANT OPTION>
```

The syntax used to revoke the only stored procedure privilege available is:

```
REVOKE EXECUTE ON [RoutineName |
                   [PROCEDURE <SchemaName.> *]
FROM [Forfeiter, ...] <BY ALL>
RESTRICT
```

where:

<i>RoutineName</i>	Identifies by name the routine (user-defined function, method, or stored procedure) that the EXECUTE privilege is to be associated with.
<i>Type</i>	Identifies by name the type in which the specified method is found.
<i>SchemaName</i>	Identifies by name the schema in which all functions, methods, or procedures—including those that may be created in the future—are to have the EXECUTE privilege granted on.
<i>Recipient</i>	Identifies the name of the user(s) and/or group(s) that are to receive the EXECUTE privilege. The value specified for the <i>Recipient</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.
<i>Forfeiter</i>	Identifies the name of the user(s) and/or group(s) that are to lose the package privileges specified. The value specified for the <i>Forfeiter</i> parameter can be any combination of the following: <USER> [ <i>UserName</i> ], <GROUP> [ <i>GroupName</i> ], and PUBLIC.

Thus, the proper way to grant and revoke stored procedure privileges is by executing the GRANT EXECUTE ... and REVOKE EXECUTE ... statements.

**Question 13**

---

The correct answer is **A**. To restrict access to rows in a table using Label-Based Access Control (LBAC), you must define a security label component, define a security policy, create one or more security labels, create an LBAC-protected table or alter an existing table to add LBAC protection (this is done by adding the security policy to the table and defining a column that has the DB2SECURITYLABEL data type), and grant the proper security labels to the appropriate users. There are no LBAC security profiles.

**Question 14**

---

The correct answer is **C**. Security Administrator (SECADM) authority is a special database level of authority that is designed to allow special users to configure various label-based access control (LBAC) elements to restrict access to one or more tables that contain data to which they most likely do not have access themselves. Users with Security Administrator authority are only allowed to perform the following tasks:

- Create and drop security policies.
- Create and drop security labels.
- Grant and revoke security labels to/from individual users (using the GRANT SECURITY LABEL and REVOKE SECURITY LABEL SQL statements).
- Grant and revoke LBAC rule exemptions.
- Grant and revoke SETSESSIONUSER privileges (using the GRANT SETSESSIONUSER SQL statement).
- Transfer ownership of any object not owned by the Security Administrator (by executing the TRANSFER OWNERSHIP SQL statement).

No other authority provides a user with these abilities, including System Administrator authority.

## Question 15

The correct answer is **B**. When a user holds an exemption on an LBAC security policy rule, that rule is not enforced when the user attempts to read and/or write data that is protected by that security policy.

Security policy exemptions are granted by executing the GRANT EXEMPTION ON RULE SQL statement (as a user with SECADM authority). The syntax for this statement is:

```
CREATE EXEMPTION ON RULE [Rule] ,...
FOR [PolicyName]
TO USER [UserName]
```

where:

*Rule* Identifies one or more DB2LBACRULES security policy rules that exemptions are to be given for. The following values are valid for this parameter: DB2LBACREADARRAY, DB2LBACREADSET, DB2LBACREADTREE, DB2LBACWRITEARRAY WRITEDOWN, DB2LBACWRITEARRAY WRITEUP, DB2LBACWRITESET, DB2LBACWRITETREE, and ALL. (If an exemption is held for every security policy rule, the user will have complete access to all data protected by that security policy.)

*PolicyName* Identifies the security policy for which the exemption is to be granted.

*UserName* Identifies the name of the user to which the exemptions specified are to be granted.

Thus, to grant an exemption to the DB2LBACWRITEARRAY rule in a security policy named SEC\_POLICY to a user named USER1, you would execute a GRANT EXEMPTION statement that looks something like this:

```
GRANT EXEMPTION ON RULE DB2LBACWRITEARRAY
WRITEDOWN FOR sec_policy
TO USER user1
```