

# Database Security: What Students Need to Know

**Meg Coffin Murray**  
**Kennesaw State University, Kennesaw, GA, USA**

[mcmurray@kennesaw.edu](mailto:mcmurray@kennesaw.edu)

## Executive Summary

Database security is a growing concern evidenced by an increase in the number of reported incidents of loss of or unauthorized exposure to sensitive data. As the amount of data collected, retained and shared electronically expands, so does the need to understand database security. The Defense Information Systems Agency of the US Department of Defense (2004), in its *Database Security Technical Implementation Guide*, states that database security should provide controlled, protected access to the contents of a database as well as preserve the integrity, consistency, and overall quality of the data. Students in the computing disciplines must develop an understanding of the issues and challenges related to database security and must be able to identify possible solutions.

At its core, database security strives to insure that only authenticated users perform authorized activities at authorized times. While database security incorporates a wide array of security topics, notwithstanding, physical security, network security, encryption and authentication, this paper focuses on the concepts and mechanisms particular to securing data. Within that context, database security encompasses three constructs: confidentiality or protection of data from unauthorized disclosure, integrity or prevention from unauthorized data access, and availability or the identification of and recovery from hardware and software errors or malicious activity resulting in the denial of data availability.

In the computing discipline curricula, database security is often included as a topic in an introductory database or introductory computer security course. This paper presents a set of sub-topics that might be included in a database security component of such a course. Mapping to the three constructs of data security, these topics include access control, application access, vulnerability, inference, and auditing mechanisms. Access control is the process by which rights and privileges are assigned to users and database objects. Application access addresses the need to assign appropriate access rights to external applications requiring a database connection. Vulnerability refers to weaknesses that allow malicious users to exploit resources. Inference refers to the use of legitimate data to infer unknown information without having rights to directly retrieve that information. Database auditing tracks database access and user activity providing a way to identify breaches that have occurred so that corrective action might be taken.

---

Material published as part of this publication, either on-line or in print, is copyrighted by the Informing Science Institute. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact [Publisher@InformingScience.org](mailto:Publisher@InformingScience.org) to request redistribution permission.

As the knowledge base related to database security continues to grow, so do the challenges of effectively conveying the material. This paper addresses those challenges by incorporating a set of interactive software modules into each sub-topic. These modules are part of an animated database courseware project designed to support the teaching of database concepts. The courseware covers

the domains of Database Design, Structured Query Language, Database Transactions, and Database Security. The Security Module, presented in this paper, allows students to explore such areas as access control, SQL injections, database inference, database auditing, and security matrices. The courseware was developed as part of a National Science Foundation grant and has been made freely available at <http://adbc.kennesaw.edu>

**Keywords:** database security, data integrity, database courseware, database vulnerability, access control.

## Introduction

Database technologies are a core component of many computing systems. They allow data to be retained and shared electronically and the amount of data contained in these systems continues to grow at an exponential rate. So does the need to insure the integrity of the data and secure the data from unintended access. The Privacy Rights Clearing House (2010) reports that more than 345 million customer records have been lost or stolen since 2005 when they began tracking data breach incidents, and the Ponemon Institute reports the average cost of a data breach has risen to \$202 per customer record (Ponemon, 2009). In August 2009, criminal indictments were handed down in the United States to three perpetrators accused of carrying out the single largest data security breach recorded to date. These hackers allegedly stole over 130 million credit and debit card numbers by exploiting a well known database vulnerability, a SQL injection (Phifer, 2010). The Verizon Business Risk Team, who have been reporting data breach statistics since 2004, examined 90 breaches during the 2008 calendar year. They reported that more than 285 million records had been compromised, a number exceeding the combined total from all prior years of study (Baker et al., 2009). Their findings provide insight into who commits these acts and how they occur. Consistently, they have found that most data breaches originate from external sources, with 75% of the incidents coming from outside the organization as compared to 20% coming from inside. They also report that 91% of the compromised records were linked to organized criminal groups. Further, they cite that the majority of breaches result from hacking and malware often facilitated by errors committed by the victim, i.e., the database owner. Unauthorized access and SQL injection were found to be the two most common forms of hacking, an interesting finding given that both of these exploits are well known and often preventable. Given the increasing number of breaches to database systems, there is a corresponding need to increase awareness of how to properly protect and monitor database systems.

At its core, database security strives to insure that only authenticated users perform authorized activities at authorized times. It includes the system, processes, and procedures that protect a database from unintended activity. The Defense Information Systems Agency of the US Department of Defense (2004), in its *Database Security Technical Implementation Guide*, states that database security should provide “controlled, protected access to the contents of your database and, in the process, preserve the integrity, consistency, and overall quality of your data” (p. 9). The goal is simple, the path to achieving the goal, a bit more complex. Traditionally database security focused on user authentication and managing user privileges to database objects (Guimaraes, 2006). This has proven to be inadequate given the growing number of successful database hacking incidents and the increase in the number of organizations reporting loss of sensitive data. A more comprehensive view of database security is needed, and it is becoming imperative for students in the computing disciplines to develop an understanding of the issues and challenges related to database security and to identify possible solutions.

Database security is often included as a topic in an introductory database course or introductory computer security course. However as the knowledge base related to database security continues to grow, so do the challenges of effectively conveying the material. Further, many topics related to database security are complex and require students to engage in active learning to fully com-

prehend the fundamental nature of database security issues. This paper presents a set of sub-topics for inclusion in a database security component of a course. These sub-topics are illustrated using a set of interactive software modules.

As part of a National Science Foundation Course, Curriculum and Laboratory Improvement Grant (#0717707), a set of interactive software modules, referred to as Animated Database Courseware (ADbC) has been developed to support the teaching of database concepts. The courseware has been made freely available and may be accessed at <http://adbc.kennesaw.edu>. ADbC consists of over 100 animations and tutorials categorized into four main modules (Database Design, Structured Query Language [SQL], Transactions and Security) and several sub-modules. Interactive instructional materials such as animations can often be incorporated into the instructional process to enhance and enrich the standard presentation of important concepts. Animations have been found to increase student motivation, and visualizations have been found to help students develop understanding of abstract concepts which are otherwise considered to be ‘invisible’ (Steinke, Huk, & Floto, 2003). Further, software animations can be effective at reinforcing topics introduced in the classroom as they provide a venue for practice and feedback. Specifically, the Security module and corresponding sub-modules will be covered in this paper. These sub-modules cover six areas: access control, row level security, application security as portrayed in a security matrix, SQL injections, database inference, and database auditing.

## Database Security Topics

The following presents an organizational structure for presenting database security concepts in a course in which database security is one of many topics. As such the focus is limited and material introductory. While database security incorporates a wide array of security topics, notwithstanding, physical security, network security, encryption and authentication, this paper focuses on the concepts and mechanisms particular to securing data. Database security is built upon a framework encompassing three constructs: confidentiality, integrity and availability (Bertino & Sandhu, 2005). Confidentiality or secrecy refers to the protection of data against unauthorized disclosure, integrity refers to the prevention of unauthorized and improper data modification, and availability refers to the prevention and recovery from hardware and software errors as well as from malicious data access resulting in the denial of data availability (Bertino, Byun & Kamra, 2007). Mapping to these three constructs, a database security component in any course needs to cover access control, application access, vulnerability, inference, and auditing mechanisms.

### Access Control

The primary method used to protect data is limiting access to the data. This can be done through authentication, authorization, and access control. These three mechanisms are distinctly different but usually used in combination with a focus on access control for granularity in assigning rights to specific objects and users. For instance, most database systems use some form of authentication, such as username and password, to restrict access to the system. Further, most users are authorized or assigned defined privileges to specific resources. Access control further refines the process by assigning rights and privileges to specific data objects and data sets. Within a database, these objects usually include tables, views, rows, and columns. For instance, StudentA may be given login rights to the University database with authorization privileges of a student user which include read-only privileges for the Course\_Listing data table. Through this granular level of access control, students may be given the ability to browse course offerings but not to peruse grades assigned to their classmates. Many students, today, inherently understand the need for granularity in granting access when framed in terms of granting ‘friends’ access to their Facebook site. Limiting access to database objects can be demonstrated through the Grant/Revoke access control mechanism.

## Access control – Grant/revoke

Access control is a core concept in security. Access control limits actions on objects to specific users. In database security, objects pertain to data objects such as tables and columns as well as SQL objects such as views and stored procedures. Data actions include read (select), insert, update, and delete or execute for stored procedures. For instance a faculty member, Dr. Smith, may be given read privileges to the Student table.

Generally, access control is defined in three ways: Mandatory Access Control (MAC), Discretionary Access Control (DAC), and Role Based Access Control (RBAC). MAC and DAC provide privileges to specified users or groups to which users are assigned. MAC rules are system applied and considered static and more secure. An example MAC rule would be giving Dr. Smith read access to the Student table. DAC rules are user supplied, considered dynamic and content focused. An example DAC rule would be giving Dr. Smith read access to the Student table but only for students enrolled in a specific course such as 'Introduction to Security.' Dr. Smith would not be able to select student data for students enrolled in other courses. MAC and DAC provide powerful tools but Role Based Access Control proves to be especially effective for database systems. Roles are analogous to job functions. With roles, the focus is on identifying operations and the objects to which those operations need access. Users assigned to a role automatically receive its associated privileges. For instance Dr. Smith may be assigned to the role of Faculty. Faculty members are given rights to read the Students table, obtain course enrollment data, and update grades for students assigned to their courses. By being assigned to the Faculty role, Dr. Smith is implicitly given these privileges.

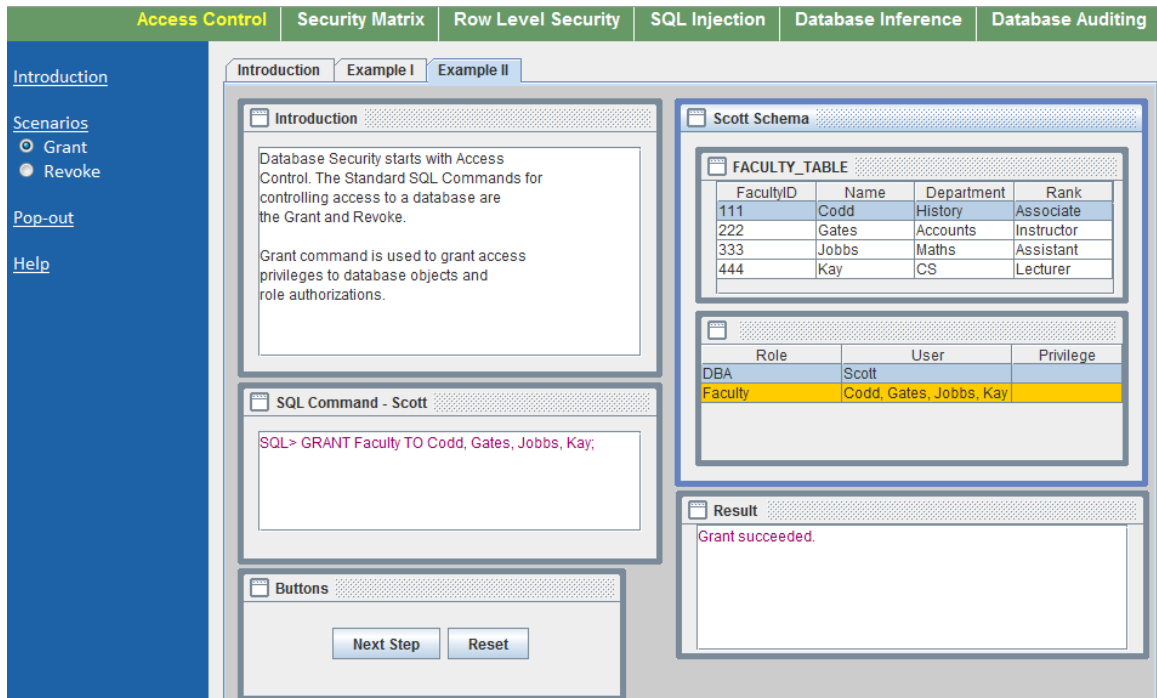
Identifying users and assessing their processing and data access needs is a major undertaking in establishing good database security protocols. Identifying and defining roles and correctly granting access rights to actions and objects and then appropriately assigning users to those roles is the crux of the process. Once a role has been created, the format for implementing RBAC follows the pattern:

```
GRANT privilege_name  
ON object_name  
TO role_name;
```

Privilege\_name identifies the rights to be granted. These include such rights as selecting data, modifying data, or manipulating the database structure. ON identifies the database objects and TO identifies the roles to which those privileges are applied. For instance, if Dr. Smith was assigned the role of Faculty and Faculty were given read rights to the Student table, the RBAC rule would be:

```
GRANT Select  
ON Student_Table  
TO Faculty;
```

The Access Control sub-module on the ADbC site introduces the concept of access control and provides two examples for granting and revoking privileges. The introduction explains the process and models its implementation through corresponding SQL statements. Example one uses a student scenario and example two uses a faculty scenario. The grant sub-module steps through the process of assigning users to roles and assigning privileges to those roles. For example, using the faculty scenario, the steps for granting role authorization to individual users include having a database administrator create the role of faculty, assigning faculty to this role, and then assigning specific rights or privileges to database objects. After being assigned to the role of Faculty, the user has all privileges assigned to that role. Figure 1 depicts the step in the process where individuals are assigned to the Faculty role.



**Access Control** | Security Matrix | Row Level Security | SQL Injection | Database Inference | Database Auditing

Introduction | Example I | Example II

**Introduction**

Database Security starts with Access Control. The Standard SQL Commands for controlling access to a database are the Grant and Revoke.

Grant command is used to grant access privileges to database objects and role authorizations.

**SQL Command - Scott**

```
SQL> GRANT Faculty TO Codd, Gates, Jobbs, Kay;
```

**Buttons**

Next Step | Reset

**Scott Schema**

**FACULTY\_TABLE**

FacultyID	Name	Department	Rank
111	Codd	History	Associate
222	Gates	Accounts	Instructor
333	Jobbs	Maths	Assistant
444	Kay	CS	Lecturer

**RoleUserPrivilege**

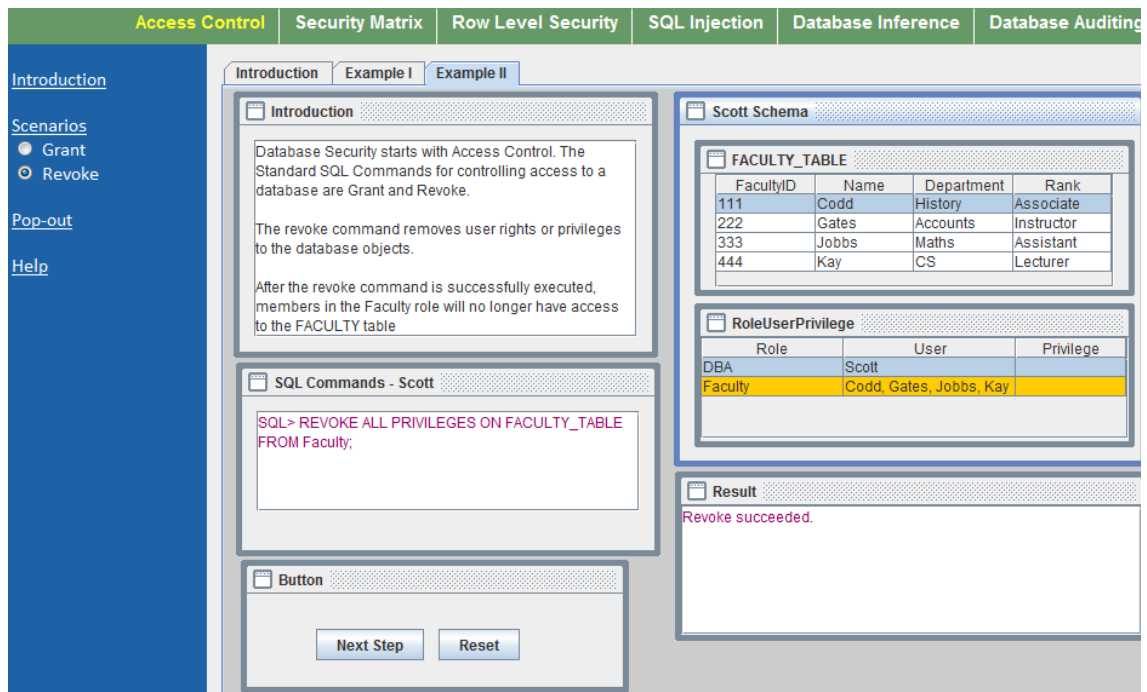
Role	User	Privilege
DBA	Scott	
Faculty	Codd, Gates, Jobbs, Kay	

**Result**

Grant succeeded.

Figure 1. ADbC Access Control Sub-module: Example Granting Role Authorization

The revoke sub-module steps through the process of revoking rights and removing users from role authorization. For example, using the faculty scenario, the steps for revoking role authorization to individual users include revoking privileges to specific database objects and removing individual users from pre-defined roles. In the case depicted in Figure 2, privileges to the Faculty



**Access Control** | Security Matrix | Row Level Security | SQL Injection | Database Inference | Database Auditing

Introduction | Example I | Example II

**Introduction**

Database Security starts with Access Control. The Standard SQL Commands for controlling access to a database are Grant and Revoke.

The revoke command removes user rights or privileges to the database objects.

After the revoke command is successfully executed, members in the Faculty role will no longer have access to the FACULTY table

**SQL Commands - Scott**

```
SQL> REVOKE ALL PRIVILEGES ON FACULTY_TABLE FROM Faculty;
```

**Button**

Next Step | Reset

**Scott Schema**

**FACULTY\_TABLE**

FacultyID	Name	Department	Rank
111	Codd	History	Associate
222	Gates	Accounts	Instructor
333	Jobbs	Maths	Assistant
444	Kay	CS	Lecturer

**RoleUserPrivilege**

Role	User	Privilege
DBA	Scott	
Faculty	Codd, Gates, Jobbs, Kay	

**Result**

Revoke succeeded.

Figure 2. ADbC Access Control Sub-module: Example Revoking Role-based Privileges

table are removed from the Faculty role. Once the privileges are revoked, members of the Faculty role will not be able to access data in the Faculty table. Figure 2 depicts the step in the process where privileges to the Faculty table are revoked from the pre-defined role of Faculty.

Syntactically, creating roles and implementing RBAC is fairly straightforward. The challenge is the management of users and their associated roles (Jaquith, 2007). Entitlement management includes not only identifying appropriate roles and their respective rights but continuous management of granted entitlements. The general security rule is to assign the most restrictive set of privileges required to complete authorized tasks. However, constructing the organizational structure for a RBAC system can quickly become complex, and the fact that users frequently change roles means that RBAC requires constant monitoring. In his book, *Security Metrics: Replacing Fear, Uncertainty, and Doubt*, Jaquith (2007) states, “Today's information security battleground is all about entitlements – who's got them, whether they were granted properly, and how to enforce them” (p.117). Being able to assess access control techniques is critical to student understanding of database security.

### Row level security

Controlling access to database tables or columns is frequently required and can be enacted by simply granting privileges to one of these objects. Restricting access to data contained in individual records (rows) requires additional steps. For instance, a student should only be able to view or modify the row or rows of data that correspond specifically to him or her. However, implementation of row level security cannot be done in the same manner as access control is applied to database objects such as tables. This is because the selection of a row is based on the evaluation of specific data values. Therefore, a common way to implement row level security is through the use of SQL Views. A View can be constructed that executes a select statement which returns specified rows of data evaluated against a specific value, such as the current user. For instance, the following SQL view would return only the row of data in which the value of the AttributeName column matched the user's id:

```
CREATE VIEW View_Name AS
SELECT *
FROM Table_name
WHERE AttributeName = USER;
```

The ADbC site provides a sub-module, entitled Row Level Security, that demonstrates this concept. A data window is presented showing table data and the SQL code for creating a View that returns row level data restricted to the name of the user. The ‘Code’ button displays all associated steps and SQL code needed for creating the table, users, and View and for assigning access rights to that View. Students can experiment with the row level security mechanism by choosing a username from the associated dropdown box. An output window displays the results of the execution of the View given the selections made by the user. As the username is modified, a different row is displayed in the output window. Figure 3 shows that when username ‘Jones’ is selected, only data related to this user is displayed.

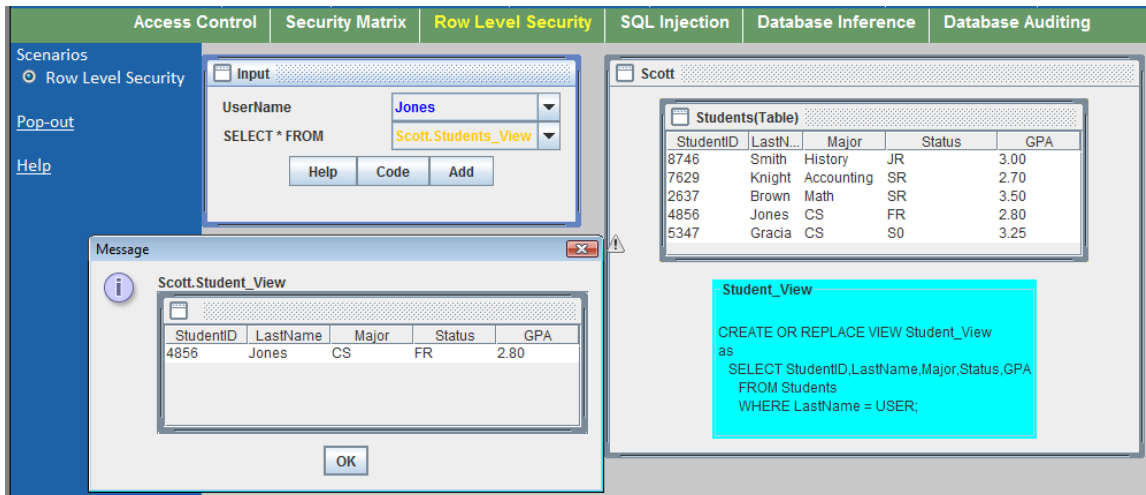


Figure 3: ADbC Row Level Security Sub-module: Example Implementation using a SQL View

Row level security, although difficult to implement, is an important database security concept. It allows for the restriction of access to data in tables in which data related to many different users is stored. It would be inefficient to store each student at a university in a separate database; it is also inappropriate to give students access to all of the data in a centralized student table. Students should be made aware of the trade-offs that have to be made to implement row level security. As an advanced topic in this area, students can be directed to study Oracle's Virtual Private Database solution to applying security policies as a way to enact row level security (Knox, 2004).

## Application Access Assessment

Most users do not access a database by directly logging into the database system. Instead they access the database through an application program. A simple tool, known as a security (or CRUD) matrix can be used to explicitly identify the required access rights needed by an application program. Specifically, the security matrix provides a visual depiction of the correlation between the operations or authorizations needed for database objects and input/output sources such as forms and reports. Operations depicted in a security matrix include Select, Create (insert), Update, and Delete. The top row of the matrix lists database table objects. Application programs are listed in the left-most column. The letters C, R, U, D are placed in intersecting cells to identify the type of access required by a particular program. Any given cell may contain any combination of these letters or none at all. An empty cell denotes that a program does not need access to the intersecting table. Conversely, a cell with all four letters, CRUD, needs full access to the table.

A Security Matrix as shown in the ADbC Security Matrix sub-module is presented in Figure 4. A customer-order scenario is depicted. Seven tables are listed across the top. Seven forms are listed down the left-hand side. Scanning the matrix left to right shows that the Order Form requires access to five tables including modification rights to three of them. Specifically the Order Form needs only read access to the Customers and Employees tables, requires read, insert, update, and delete rights to the Order\_Details and Orders table, and requires read and update rights to the Products table. Scanning top to bottom shows that three applications, Customer Labels, Customer Information, and Order Form, access the Customers table. The Customer Labels and Orders Form require read access to the Customers table while the Customer Information form requires read, insert, update, and delete rights. The Security Matrix sub-module includes an accompanying set of interactive questions that ask users to identify relationships between the tables and the application programs.



	CATEGORIES	CUSTOMERS	EMPLOYEES	ORDER_DETAILS	ORDERS	SUPPLIERS	PRODUCTS
Categories Form	CRUD						
Customer Labels		R					
Customer Info		CRUD					
Order Form		R	R	CRUD	CRUD		RU
Employee Form			CRUD				
Supplier Form						CRUD	
Product Form							CRUD

C = Create or Insert a Record (row)  
 R = Read/Query or Select a row  
 U = Update or Modify  
 D = Delete

Figure 4: ADbC Security Matrix Sub-module: Example Security Matrix

Another advantage to the security matrix is that it visually depicts rules of integrity. For instance, the matrix makes it easy to identify all application programs potentially affected by any change made to a database table. For example, a column deleted from the Products table will impact the Orders form and Products form, possibly generating an error when these applications are executed. Before such a change is made, its subsequent impact must be assessed to ascertain what applications will need updates. In summary, the security matrix is a simple, yet effective, tool for identifying needed security permissions to database objects.

### **Database Vulnerability**

Security breaches are an increasing phenomenon. As more and more databases are made accessible via the Internet and web-based applications, their exposure to security threats will rise. The objective is to reduce susceptibility to these threats. Perhaps the most publicized database application vulnerability has been the SQL injection. SQL injections provide excellent examples for discussing security as they embody one of the most important database security issues, risks inherent to non-validated user input. SQL injections can happen when SQL statements are dynamically created using user input. The threat occurs when users enter malicious code that ‘tricks’ the database into executing unintended commands. The vulnerability occurs primarily because of the features of the SQL language that allow such things as embedding comments using double hyphens (- -), concatenating SQL statements separated by semicolons, and the ability to query metadata from database data dictionaries. The solution to stopping an SQL injection is input validation.

A common example depicts what might occur when a login process is employed on a web page that validates a username and password against data retained in a relational database. The web page provides input forms for user entry of text data. The user-supplied text is used to dynamically create a SQL statement to search the database for matching records. The intention is that valid username and password combinations would be authenticated and the user permitted access to the system. Invalid username and passwords would not be authenticated. However, if a disingenuous user enters malicious text, they could, in essence, gain access to data to which they have no privilege. For instance, the following string, ' OR 1=1 -- entered into the username textbox



gains access to the system without having to know either a valid username or password. This hack works because the application generates a dynamic query that is formed by concatenating fixed strings with the values entered by the user.

For example, the model SQL code might be:

```
SELECT Count(*) FROM UsersTable
WHERE UserName = 'contents of username textbox'
AND Password = 'contents of password textbox';
```

When a user enters a valid username, such as 'Mary' and a password of 'qwerty', the SQL query becomes:

```
SELECT Count(*) FROM UsersTable
WHERE UserName='Mary'
AND Password='qwerty';
```

However, if a user enters the following as a username: 'OR 1=1 -- the SQL query becomes:

```
SELECT Count(*) FROM UsersTable
WHERE UserName=' OR 1=1 -- '
AND Password='';
```

The expression  $1 = 1$  is true for every row in the table causing the OR clause to return a value of true. The double hyphens comment out the rest of the SQL query string. This query will return a count greater than zero, assuming there is at least one row in the users table, resulting in what appears to be a successful login. In fact, it is not. Access to the system was successful without a user having to know either a username or password.

Another SQL injection is made possible when a database system allows for the processing of stacked queries. Stacked queries are the execution of more than one SQL query in a single function call from an application program. In this case, one string is passed to the database system with multiple queries, each separated by a semicolon. The following example demonstrates a stacked query. The original intent is to allow the user to select attributes of products retained in a Products table. The user injects a stacked query incorporating an additional SQL query that also deletes the Customers table.

```
SELECT * FROM PRODUCTS; DROP CUSTOMERS;
```

This string when passed as an SQL query will result in the execution of two queries. A listing of all information for all products will be returned. In addition the Customers table will be removed from the database. The table structure will be deleted and all customer data will be lost. In database systems that do not allow stacked queries, or invalidate SQL strings containing a semicolon, this query would not be executed.

The ADbC courseware sub-module for SQL injections demonstrates the insertion of malicious code during the login process. The sub-module steps through the process by first showing the entry of valid data and then demonstrating entry of malicious code, how it is injected into a dynamically created SQL statement and then executed. Figure 5 shows the step where malicious code is entered. Figure 6 shows the dynamically created SQL command and the resulting display of all the data in the user table. Additional steps present code resulting in the modification or deletion of data.

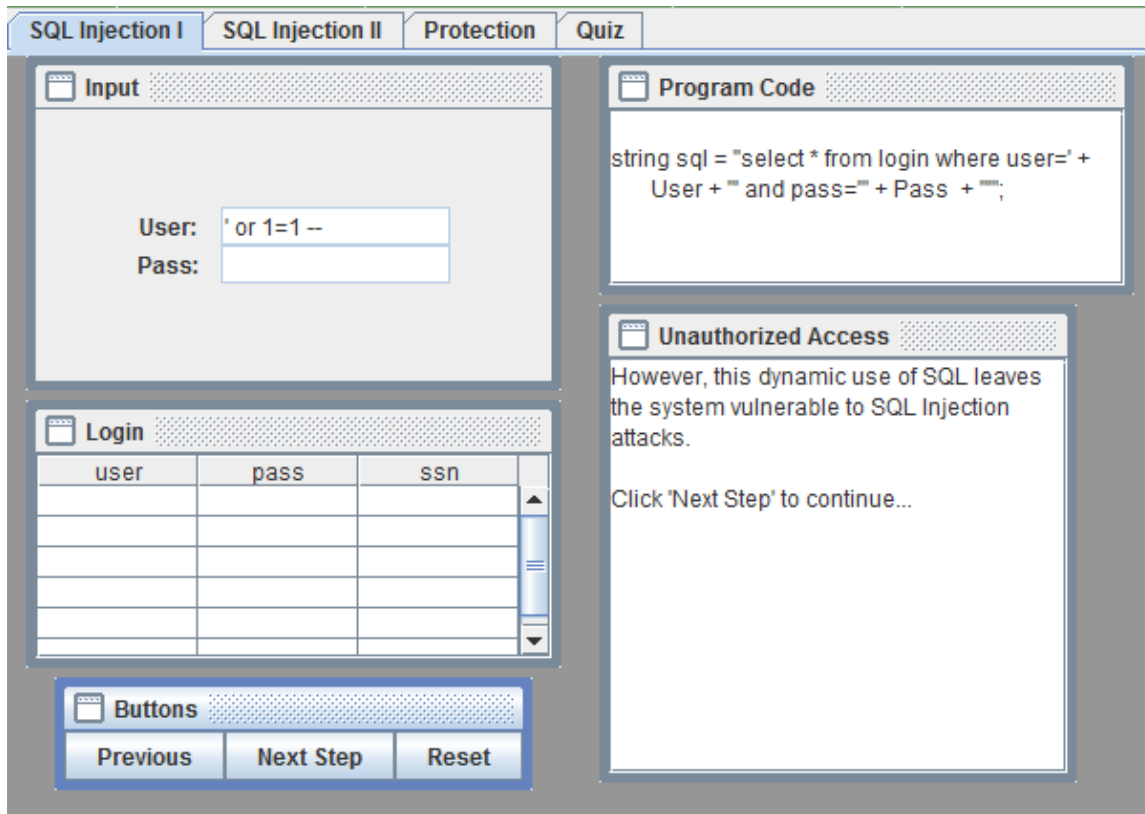


Figure 5: ADbC SQL Injection Sub-Module: Entering Malicious Code in a SQL Injection

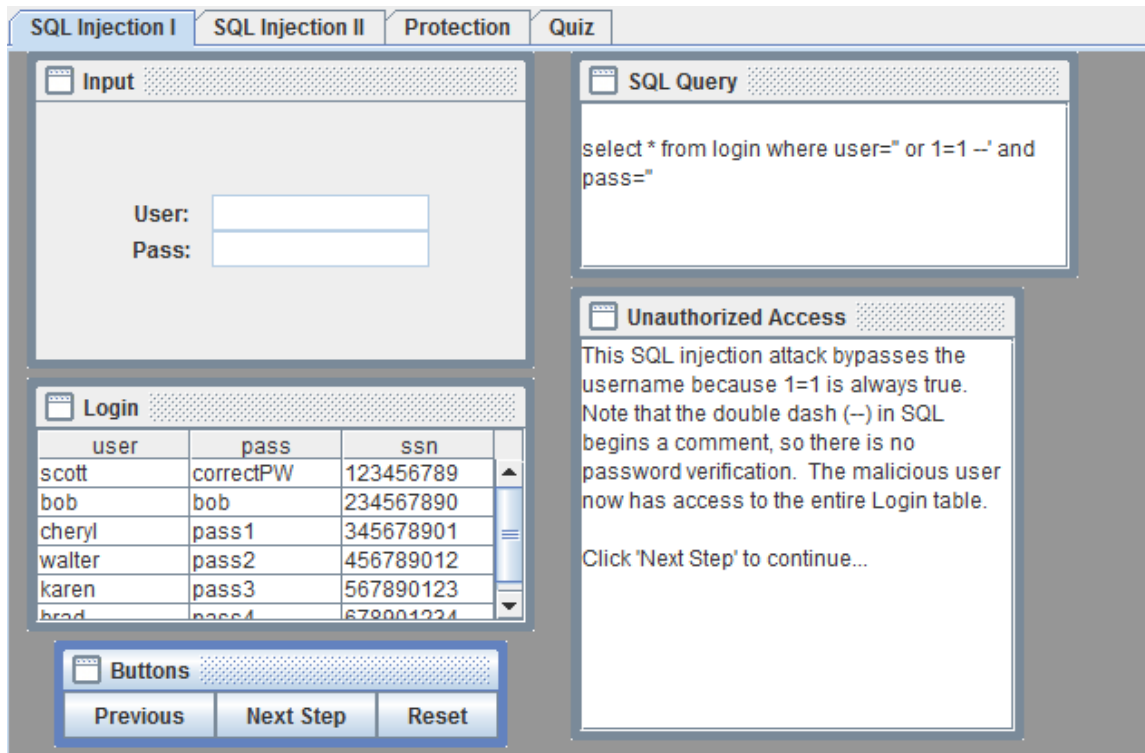


Figure 6: ADbC SQL Injection Sub-Module: Result of SQL Injection using Malicious Code

SQL injection vulnerabilities result from the dynamic creation of SQL queries in application programs that access a database system. The SQL queries are built incorporating user input and passed to the database system as a string variable. SQL injections can be prevented by validating user input. Three approaches are commonly used to address query string validation: using a black list, using a white list, or implementing parameterized queries. The black list parses the input string comparing each character to a predefined list of non-allowed characters. The disadvantage to using a black list is that many special characters can be legitimate but will be rejected using this approach. The common example is the use of the apostrophe in a last name such as O'Hare. The white list approach is similar except that each character is compared to a list of allowable characters. The approach is preferred but special considerations have to be made when validating the single quote. Parameterized queries use internally defined parameters to fill in a previously prepared SQL statement. The importance of input validation cannot be overstated. It is one of the primary defense mechanisms for preventing database vulnerabilities including SQL injections.

### **Database Inference**

A subtle vulnerability found within database technologies is inference, or the ability to derive unknown information based on retrieved information. The problem with inference is that there are no ideal solutions to the problem. The only recommended solutions include controls related to queries (suppression) or controls related to individual items in a database (concealing). In other words, sensitive data requested in a query are either not provided or answers given are close but not exact, preventing the user from obtaining enough information to make inferences. Neither of these represents ideal solutions as they are restrictive in nature. However, it is important for students to understand the risks of inference and how it might occur. Examples are the best way to demonstrate inference. The ADbC inference sub-module includes three animations that demonstrate how users might be able to put together (infer) information when data is available to those with a higher security access level or when they are only given access to aggregate data.

Inference often happens in cases where the actual intent is for users to generate or view aggregate values when they have not been given access to individual data items. However, because they are exposed to information about the data, they are sometimes able to infer individual data values. Take for example a scenario where a worker desires to find out their co-worker Goldberg's salary. In this organization, salary data is confidential. The worker has rights to generate aggregate data such as summarizing organizational salary data averaged across specific criteria (i.e., salary averaged by gender). Although the worker does not have access to individual data items, he or she does possess particular and unique details about Goldberg; specifically that Goldberg is a female and has 11 dependents. Based on this information, the worker can derive an aggregate function such as *SELECT AVG (Salary) FROM EMPLOYEES WHERE Gender = "F" and Dependents = 11*. This will return Goldberg's salary because the average is taken from an aggregated data set of one. The ADbC inference sub-module animation for this scenario is illustrated in Figure 7. The SQL-command window depicts the construction of the requested query to ascertain salary averages. Employees Table data is shown in the upper left and underneath is the result of the query.

The screenshot displays the ADbC Inference Sub-module interface with three tabs: Inference I, Inference II, and Inference III. The Inference II tab is active.

**Information Window:** The user wants to know Goldberg's salary and knows that Goldberg is a female with 11 dependents. The user then creates a query to select Female employees with 11 dependents, believing that no other female employees will have 11 dependents. Click 'Next' button to continue.

**EMPLOYEES Table:**

LASTNAME	GENDER	TITLE	CITY	STATE	SALARY	DEPEND...
Monroe	F	Consultant	Atlanta	GA	50000	2
Jobs	M	DBA	Cupertino	CA	98000	1
Goldberg	F	Manager	Palo Alto	CA	76000	11
Gay	M	Director	Sacrame...	CA	82000	3
Gates	M	Director	Seattle	WA	198000	5
Hopper	F	Manager	Boston	MA	32000	2
Wozniack	M	Director	Freemont	CA	65000	3
Codd	M	Consultant	San Jose	CA	22000	4

**RESULTS - Junior Employee:**

AVG_SALARY
76000

**SQL Commands - Junior Employee:**

```
SELECT AVG(SALARY) AS AVG_SALARY
FROM EMPLOYEES
WHERE GENDER = 'F'
AND DEPENDENTS = 11;
```

>Access Level: Admin  
>Query successful.

**Buttons:** Prev., Next, Reset, Help

Figure 7. ADbC Inference Sub-module: Using Aggregate Data to Infer Information

Inference can also occur when users are able to ascertain information from data accessible to them at their security level even though that specific information is protected at a higher security access level. It is difficult to explain this without the aid of a demonstration. The second example in the ADbC Inference sub-module provides a scenario where specific data, in this case company product prototype data, is not made accessible to junior employees. However, junior employees are given access to update the Storage table that tracks the contents in company storage areas. When perusing this table, the junior employee is not able to read any rows containing prototype products. The problem occurs if the employee tries to update a protected row. This triggers an error message. Based on the error message, the junior employee could surmise that information was being hidden and might infer that something of a secretive nature was being stored in the storage compartment referenced in the update request. Figure 8 depicts an error being generated when a junior employee issues a query against a protected row of data. The table on the top right shows all of the data contained in the Storage table. The table on the bottom shows the data accessible to junior employees. Notice that Compartment B containing ProductX is not displayed in the lower table. A possible solution to this inference problem is polyinstantiation. Polyinstantiation allows a database to retain multiple records having the same primary key; they are uniquely distinguished by a security level identifier. If polyinstantiation were enacted in the preceding scenario, the insert would be successful. However, this does not prevent the 'double booking' of the storage compartment area.

The screenshot displays the ADbC Inference Sub-module interface, which is divided into several sections:

- Information Window:** Contains text explaining the inference process. It states that an employee cannot insert a row into a table and can infer that something is stored in a compartment above their security clearance level. It also suggests that the employee could use this knowledge to potentially harm the company and instructs the user to click the 'Next' button.
- SQL Commands - Junior Employee:** Shows an SQL INSERT statement for the 'STORAGE' table. The command is:
 

```
INSERT INTO STORAGE
(ROOM_NUM, COMPARTMENT, DESC_TEXT,
ACCESS_TYPE)
VALUES('123', 'B', 'Misc Supplies', 'All')
```

 Below the command, there are two error messages:
 

```
>Access denied.
>Operation failed.
```
- STORAGE Table:** A table with columns ROOM\_NUM, COMPARTMENT, DESC\_TEXT, and ACCESS\_TYPE. The data is as follows:
 

ROOM_NUM	COMPARTMENT	DESC_TEXT	ACCESS_TYPE
123	A	Computer	All
123	B	Product X	Special
123	C	Test Equipment	All
125	A	Research	All
125	B	Janitorial Supplies	All
125	C	File Backup	All
- RESULTS - Junior Employee Table:** A table with columns ROOM\_NUM, COMPARTMENT, DESC\_TEXT, and ACCESS\_TYPE. The data is as follows:
 

ROOM_NUM	COMPARTMENT	DESC_TEXT	ACCESS_TYPE
123	A	Computer	All
123	C	Test Equipment	All
125	A	Research	All
125	B	Janitorial Supplies	All
125	C	File Backup	All
- Buttons:** A set of four buttons labeled 'Prev.', 'Next', 'Reset', and 'Help'.

Figure 8. ADbC Inference Sub-module: Security Level Access Error Leading to Inference

Developing technological solutions to detecting database inference is complex. Much of the work done in this area involves revoking access to specific database objects based on a user's past querying history (Staddon, 2003). The problem with inference detection, especially when done at query processing time, is that it results in a significant delay between the time the query is executed and the results are presented. As with other approaches to mitigating database security vulnerabilities, trade-offs must be made. The protection of highly sensitive data requires an examination of what situations could lead to exposure to unauthorized users and what monitoring policies should be implemented to insure appropriate responses are enacted.

## Auditing

Database auditing is used to track database access and user activity. Auditing can be used to identify who accessed database objects, what actions were performed, and what data was changed. Database auditing does not prevent security breaches, but it does provide a way to identify if breaches have occurred. Common categories of database auditing include monitoring database access attempts, Data Control Language (DCL) activities, Data Definition Language (DDL) activities, and Data Manipulation Language (DML) activities (Yang, 2009). Monitoring access attempts includes retaining information on successful and unsuccessful logon and logoff attempts. DCL audits record changes to user and role privileges, user additions, and user deletions. DDL audits record changes to the database schema such as changes to table structure or attribute data-types. DML audits record changes to data. In addition, database errors should be monitored (Yang, 2009). Database auditing is implemented via log files and audit tables.

The real challenge of database auditing is deciding what and how much data to retain and how long to keep it. Several options exist. A basic audit trail usually captures user access, system resources used, and changes made to the structure of a database. More complete auditing captures data reads as well as data modifications. The ADbC auditing sub-module provides step-by-step examples for creating audits of user sessions, changes to database structure, and modifications to data. Figure 9 shows an example of code required to implement and trigger an audit of a user login. Data recorded includes the username and the date and time of the user login and logoff.

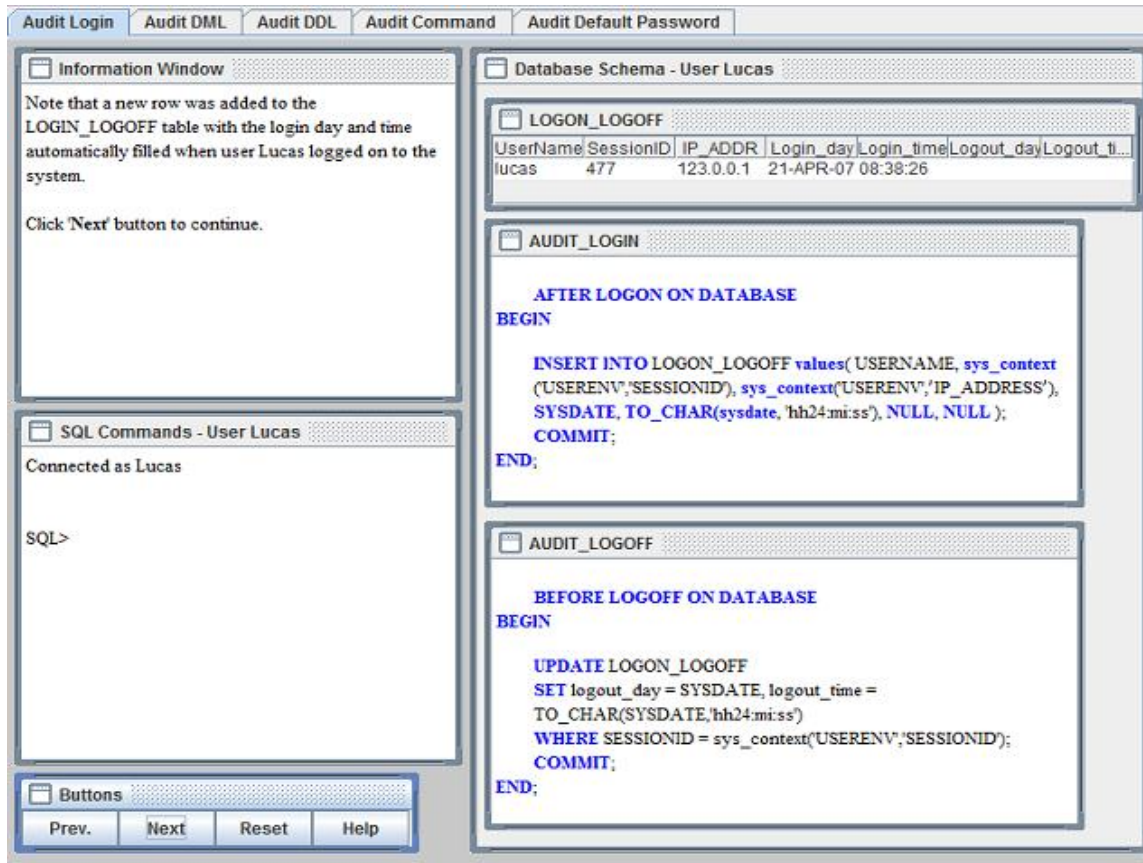


Figure 9. ADbC Database Audit Sub-module: Monitoring User Logins

An audit trail provides a more complete trace recording of not only user access but also user actions. This type of facility is included with many database management systems. The most common items that are audited include login attempts, data read and data modifications operations, unsuccessful attempts to access database tables, and attempts to insert data that violates specific constraints. Figure 10 shows an example audit trail of user access and user actions as demonstrated in the Audit Command animation in the ADbC Database Audit sub-module. The SQL Commands window displays the SQL statement used to retrieve data from the audit table.



The screenshot displays the ADbC Database Audit Sub-module interface with the following components:

- Navigation Tabs:** Audit Login, Audit DML, Audit DDL, Audit Command (selected), Audit Default Password.
- Information Window:** Last two access attempts to the Customers table are logged.
- DBA\_AUDIT\_TRAIL Table:**

USERNAME	TIMESTAMP	OWNER	OBJ_NAME	ACTION_NAME
SMITH1	05-DEC-06	SMITH1	CUSTOMERS	ALTER TABLE
SMITH1	05-DEC-06	SMITH1	CUSTOMERS	DELETE
- SQL Commands:**

```
CONNECT CS8990/C;

SELECT USERNAME, TIMESTAMP, OWNER, OBJ_NAME, ACTION_NAME
FROM DBA_AUDIT_TRAIL
WHERE ROWNUM < 3
ORDER BY TIMESTAMP DESC;
```
- Buttons:** Prev., Next, Reset, Help.

Figure 10. ADbC Database Audit Sub-module: Example Database Audit Trail

Auditing plays a central role in a comprehensive database security plan. The primary weakness of the audit process is the time delay between when data is recorded and when analysis is performed. Consequently, breaches and other unauthorized activities are identified after the fact, making it difficult to mitigate adverse effects in a timely manner. However, solutions are being introduced that allow for real-time monitoring of database activity looking for patterned events indicative of potential breaches and enacting real-time notification to database administrators when such actions occur. Whatever the case, database auditing is a necessary process, and students must be made aware of the need for continuous monitoring of database log files.

## ADbC Courseware Use and Evaluation

The ADbC courseware was developed to provide supplemental instructional support in the classroom. The software is not tailored to any specific product or textbook nor is it intended to substitute for these resources. Instead it provides another venue for student interaction and engagement with course material. The ADbC components support both in-class instruction and out-of-class practice and reinforcement. They were not designed to be the sole source for learning a new concept. In the classroom, faculty may use the courseware for demonstration when explaining a topic. The sub-modules can also be assigned as out-of-class activities to reinforce concepts presented in class. Most of the sub-modules include more than one animation allowing for one example to be demonstrated in class and another example to be assigned for out of class practice. Students may also be referred to the courseware for assistance when completing homework assignments or studying for exams.

An evaluation of student perceptions of the teaching effectiveness of the Security Module was undertaken in the spring of 2009. Sixty students were asked to complete an online questionnaire. Thirty-eight students responded. In addition, interviews were conducted with three faculty members using the courseware in their classes. Students responding to the survey were asked to assess if a particular sub-module (access control, SQL injections, security matrix, row level security, database inference, and database auditing) helped them to learn the material. The majority of students indicated that they agreed or strongly agreed that the sub-modules did enhance their learning. Sub-modules with the highest rating included SQL injections and row level security. Database inference and database auditing received the highest number of students reporting a rating of 'neutral.' When asked when they were motivated to use the software, 41% indicated they only used it when specifically requested to do so by their instructor. However, 51% reported they also used it on their own. Eight percent indicated they used the software often. The most common reason given for using the software was to complete homework assignments or study for exams.

Faculty reported that their teaching was enhanced by the courseware, citing that the courseware mapped well to the concepts taught in class, that it provided a means of reinforcement for student learning, and that it was a much better process for presenting the concepts than the traditional way of drawing pictures on a whiteboard. This evaluation was limited in scope to a relatively small number of students and faculty. However, the results are encouraging. Both students and faculty reported positive benefits to using the courseware.

## Conclusion

The need to secure computer systems is well understood and securing data must be part of an overall computer security plan. Growing amounts of sensitive data are being retained in databases and more of these databases are being made accessible via the Internet. As more data is made available electronically, it can be assumed that threats and vulnerabilities to the integrity of that data will increase as well. Database security is becoming an increasingly important topic and students need to develop core understandings in this area. The primary objectives of database security are to prevent unauthorized access to data, prevent unauthorized tampering or modification of data, and to insure that data remains available when needed.

The concepts related to database security are multifaceted. This makes it challenging to teach the material when database security is included as just one component of a larger course. However, this is how most students are exposed to the topic. This paper suggested a set of sub-topics in a database security course component and introduced a set of interactive software modules mapped to each sub-topic presented. Engaging students in interactive learning activities enhances the learning experience and provides the opportunity for students to further explore database security issues and identify practical implementation methods to database security mechanisms and strategies.

## References

- Baker, W. H., Hutton, A., Hylender, C. D., Novak, C., Porter, C., Sartin, B., Tippett, P., & Valentine, J. A. (2009). *The 2009 data breach investigations report*. Verizon Business. Retrieved January 31, 2010, from [http://www.verizonbusiness.com/resources/security/reports/2009\\_databreach\\_rp.pdf](http://www.verizonbusiness.com/resources/security/reports/2009_databreach_rp.pdf)
- Bertino, E., Byun, J., & Kamra, A. (2007). Database security. In M. Petkovic & W. Jonker (Eds.), *security, privacy, and trust in modern data management (Data-centric systems and applications)* (pp. 87-102). New York: Springer-Verlag.
- Bertino, E., & Sandhu, R. (2005). Database security—concepts, approaches, and challenges. *IEEE Transactions on Dependable and Secure Computing*, 2(1), 2-18.

- Defense Information Systems Agency. (2004). *Database security technical implementation guide*, 7(1). Department of Defense. Retrieved January 31, 2010, from <http://www.databasesecurity.com/dbsec/database-stig-v7r1.pdf>
- Guimaraes, M. (2006). New challenges in teaching database security. *Proceedings of the 3rd Annual Conference on Information Security Curriculum Development*, Kennesaw, GA, USA, 64-67.
- Jaquith, A. (2007). *Security metrics: Replacing fear, uncertainty, and doubt*. Redwood City, CA: Addison-Wesley Professional.
- Knox, D. C. (2004). *Effective Oracle database 10g security by design*. New York: McGraw-Hill/Osborne.
- Phifer, L. (2010). Top ten data breaches and blunders of 2009. *eSecurity Planet*, February 10. Retrieved from <http://www.esecurityplanet.com/features/article.php/3863556/Top-Ten-Data-Breaches-and-Blunders-of-2009.htm>
- Ponemon, L. (2009). *Fourth annual US cost of data breach study*. Ponemon Institute sponsored by PGP Corporation. Retrieved January 31, 2010, from <http://www.ponemon.org/local/upload/fckjail/generalcontent/18/file/2008-2009%20US%20Cost%20of%20Data%20Breach%20Report%20Final.pdf>
- Privacy Rights Clearing House. (2010). *Chronology of data breaches*. Retrieved January 31, 2010, from <http://www.privacyrights.org/>
- Staddon, J. (2003). Dynamic inference control. *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, San Diego, CA, USA, 94-100.
- Steinke, M., Huk, T., & Floto, C. (2003). Helping teachers developing computer animations for improving learning in science education. *Proceedings of the Society for Information Technology and Teacher Education International Conference*, Chesapeake, VA, 3022-3025.
- Yang, L. 2009. Teaching database security and auditing. *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, Chattanooga, TN, USA.

## Biography



**Meg Coffin Murray** is an Associate Professor in the Department of Computer Science and Information Systems at Kennesaw State University. She holds a Ph.D. in Information Systems and has over thirty years of experience in both academe and industry. Dr. Murray specializes in the development and implementation of emerging technologies to meet business and organizational needs. Her current work has focused on web services and using XML as a medium for data exchange. She is also involved in devising strategies to assess and remediate skills needed to leverage IT in innovation, a primary driver of economic growth.