SPIE: A Framework For Advanced Database Topics

David Olsen, (E-mail: david.olsen@usu.edu), Utah State University
Karina Hauser, (E-mail: karina.hauser@usu.edu), Utah State University
Zsolt Ugray, (E-mail: zsolt.ugray@usu.edu), Utah State University

ABSTRACT

In the ever expanding universe of database skills and knowledge we propose a framework that can be used to classify advanced database topics. We use the framework to present five advanced database modules that can be successfully incorporated in an advanced database course. These modules were built to illustrate advanced topics and were tested and refined in advanced database courses over several semesters. The skills demonstrated in the modules go beyond what is typically taught in an the introductory level database course but are important in today's highly demanding business environment.

INTRODUCTION

According to the occupational outlook handbook of the Bureau of Labor Statistics the job market for database administrators is rapidly expanding. It is expected to grow faster than most other IT related occupation, with a predicted growth of 18% to 26% during the next 10 years [23]. In a recent study of the most demanded job skills in information technology, one-fifth of all advertised jobs required SQL programming skills [17]. These results are not surprising considering that databases are the foundation of any information system and companies nowadays not only use databases to run their day-to-day operations but also have discovered their strategic value. Data warehousing and data mining are used increasingly by a wide variety of industries to develop a competitive advantage. Major changes have occurred since Codd developed his relational data model in 1970 [3]. Firstly, security and privacy of data is on everybody's mind, with identity theft being in the news nearly daily, and needs to be considered in the development and access of databases. Secondly, databases nowadays easily reach giga- and terabytes sizes [27] and manageability and performance issues have to be considered in the development of data base management systems (DBMS). Thirdly, access of data over the Internet has been continuously growing and this trend is unlikely to change in the near future. Lastly, usability will become more and more important with managers discover the value of data and their use in strategic decision making. No longer does a manager want to have to call the IT department to run a Query and wait days for an answer, they want to do it themselves and they want an answer right away.

Modern database functions have to go significantly beyond the traditional requirements of storing, managing and extracting data.

ADVANCED DATABASE TOPICS IN EDUCATION

The importance of database knowledge and skills is well known in education. In a survey about core course offered at universities, Kung et. al. [11] showed that 92% of all IS programs offer at least one database course. Some advanced database topics, like data warehousing and data mining, are explicitly mentioned in the IS2002 model curriculum [6], whereas others, like preparation of data for publishing on websites or retrieving data directly from websites, are not mentioned at all. Literature regarding advanced database curriculum issues is mainly published in the computer science area [18, 19, 21, 24] with a few exceptions that can be found in information systems oriented outlets [5, 20]. Several universities have recognized the need for coverage of advanced database topic and have adjusted their curricula accordingly. As an example, BYU's Information Technology program has recognized the

importance of the integration of databases and the Internet and has designed a course to incorporate that knowledge early in their curriculum [5]. The University of North Florida has added a second database course to their IS curriculum to cover advanced topic, such as data warehousing, data mining and web databases. In addition, some teaching cases have been developed to support semester long database projects [7] and to include special tools, like XML, into database courses in Information Systems [13]. Given the importance of database skills in the job market there is a need to develop and incorporate further areas of advanced database knowledge and skills into the curriculum.

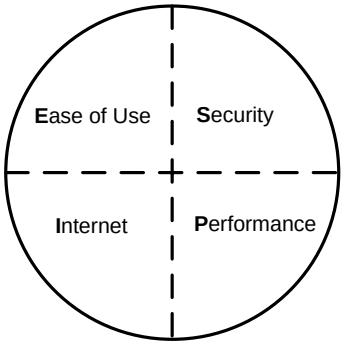
The remainder of the paper is structured as follows: First we provide a framework for classifying advanced database topics and the reasoning behind it. We then provide five detailed examples of advanced database modules that can be easily incorporated into an advanced database class. We close the paper by offering our conclusion and providing directions for future research.

FRAMEWORK

Advanced database course topics need to be relevant, practical, and especially up-to-date. Unfortunately, most textbooks do not cover advanced database topics and are slow in the adoption of emerging technologies. Educators often rely on information found on the Internet to supplement the main textbook they use.

Advanced database topics can be classified into four large categories, enhancements in **S**ecurity, **P**erformance, **I**nternet and **E**ase of use functionality. The framework is illustrated in Figure 1. The dotted lines symbolize the concept that the categories are not exclusive; a topic could cover more than one category.

Figure 1: SPIE Pie



Each of the categories is described in more detail below and we give examples of how the enhancements can be achieved.

Security

Security issues are becoming increasingly important in database design. Provisions have to be taken to ensure that data is stored securely and that only authorized individuals have access rights to the data. Whereas discretionary access control (granting and revoking access rights for data objects) is a standard feature of most database management systems, mandatory access control systems, where rights are determined by the user's and the data object's membership in security classes, need to be designed and developed based on the user's needs [9]. Moreover, tools such as stored procedures must be used to prevent malicious work, like SQL injection attacks.

Performance

Databases that consist of terabytes are becoming increasingly common. The largest commercial database at Walmart exceeds the one-half petabyte size [2]. The need for faster, real-time decision making has brought new challenges to the performance of database systems. Poor performance of database applications might lead to end-user frustration and abandonment of the application [14]. One approach to increase run-time performance of queries is the use of views since part of the query optimization plan has already been performed [8]. Poorly written SQL statements can significantly degrade database performance and a wide variety of suggestions is available in the literature to help with that problem [22]. However, the issues of database performance and tuning are largely overlooked in traditional database courses [16].

Internet

There are three major areas related to advanced database topics and the Internet: data retrieval, data exchange and data publishing.

Much of the information that organizations use for decision making originates from the Internet. Several tools that gather data from the Internet have been developed and incorporated into company databases, research data sets, and knowledge management repositories[e.g. 10, 25]. This process makes it possible to use data from a variety of sources in data warehousing and data mining applications.

Globalization made it necessary that data is accessible and exchangeable worldwide over the Internet. The eXtensible Markup Language (XML) was designed to facilitate data exchange across different systems over the Internet [26]. Educators have responded to the emerging importance of XML with the development of case studies that help integrate XML tools and concepts into database classes[13, 15].

Ease Of Use

Decision makers usually lack the technical skills and knowledge to define complex queries needed for strategic decision making. Several approaches have to be developed to make data accessible with minimal effort to users who have only rudimentary technical skills. Alhajj [1] designed an easy to use, straightforward model for the formulation of complex, object-oriented queries. The model handles aggregation and includes linear recursive queries. They develop a selection and aggregation operator that allows users to locate certain branches or to retrieve a particular subgraph to derive summarized information from a database. Dadashzadeh [4] and Matos and Grasser [12] provide teaching tips for simpler approaches to complex queries. They found that certain concepts (i.e. set comparisons and relational division) are hard to comprehend for students and suggest solutions that are easier to understand.

Most of the above examples used to illustrate the framework are very complex and not suitable for use in the classroom. In the next section we provide 5 detailed examples of advanced database modules that can be easily incorporated into an advanced database course.

ADVANCED DATABASE MODULES

The first example (SQL Injections) illustrates how easily security can be compromised and gives suggestions on how to prevent it. The second example (Daily Sales) describes how to generate large amounts of data for performance testing. The third example (XML and XSL Module for Publishing Data on the Internet) demonstrates how to format data for publishing on the Internet. The forth and fifth (OverLaps and Dynamic SQL) example show different applications that increase ease of use.

Each module starts with the purpose of the module followed by its relationship to standards, i.e. if specialized software is required to run the module.

SQL INJECTIONS

Purpose: A hacker breaks into a password protected system by injecting strategically formed SQL into a query. This attack works because the executed query is formed by the concatenation of a fixed string and values entered by the user. A user may be asked to give a name and a password and instead, gives a partial string of SQL code.

Relationship to standards: This module contains only ANSI-standard SQL and therefore runs in any DBMSenvironment.

1. This example uses a known username (JoeS) and then creates a comment after the valid username so that the WHERE clause never checks for a password.
   Username: JoeS ‘ --
   Password: password

2. This example is used with an unknown user or password. The first time the data string is entered the user is denied access. A new user and password are created and inserted. The second time the values are entered, the user is allowed entrance because the values are valid in the users table.
   Username: JoeH’ ; Insert Users Values (‘JoeH’, ‘password’) - -
   Password: password

   Username: JoeH
   Password: password

3. In this example, no entry is being attempted but rather, all users are going to be deleted. This malicious attack deletes all users and can take significant recovery time.
   Username: JoeH’ ; Delete Users --
   Password: password

Best Practices For Combating Against SQL Injection Attacks:

- Use parameterized queries or stored procedures to access a database, as opposed to using string concatenation.
- Using stored procedures that are executed server side and that disallow non-character fields such as ', -, %, LIKE, etc.
- Limit the numbner of characters to an adequate number using some kind of MAX CHARACTER function or constraint.
- Do not display errors to the user that contain hacking information like table names, attributes, drivers, SQL statements, etc. Use generic web pages and hide all the systems information.

DAILY SALES

Purpose: A common task for database administrations is to populate tables with large amounts of data for performance testing. In this example, we show how to populate a DailySales table by creating sales dates and random numbers for daily sales totals for a company’s three divisions.

Relationship to standards: This module contains only ANSI-standard SQL commands and therefore runs in any DBMS environment.

1. Create a daily sales table as specified below:

```
CREATE TABLE DailySales
(
  SalesDate     datetime          NOT NULL,
  DailyTotal    decimal(21,2)     NOT NULL,
```

```
  CoDivision      int                          NOT NULL,
  CHECK (CoDivision IN (1,2,3))
)
```

2. Make sure no messages are sent when running the Query:

```
SET NOCOUNT ON
```

3. The following Transact-SQL (TSQL) statement populates the DailySales table for an entire year (2007) with random sales data. The RAND() function is used to help create truely random 3 digit numbers.

```
DECLARE @day AS int, @i AS int

SET @day = 0
WHILE @day < 365
BEGIN
  SET @i = 1
  WHILE @i < 4                                  // division#  loop; @i holds division#
  BEGIN
    INSERT INTO DailySales VALUES
    (DATEADD(Day,@day, '20070101'),
    CAST(RIGHT(RAND(),3)  AS DECIMAL(21,2)),
    @i)
  SET @i = @i + 1
  END
  SET @day = @day + 1
END
```

4. Create an SQL statement that creates the following output which includes the sales date and the average daily sales amount for the three divisions for a given day. Do so for the month of January, 2007.

```
SELECT
  CAST(SalesDate AS char(12)) AS sales_date,
  AVG(DailyTotal) AS avg_sales
FROM DailySales
WHERE SalesDate >= '20070101' AND SalesDate < '20070201'
GROUP BY CAST(SalesDate AS char(12))
ORDER BY sales_date
```

The GROUP BY and ORDER BY functions work for a date because a date is stored in the computer as a number such as 39,118 which refers to Wednesday, Feb.7, 2007.
The SQL statement to retrieve that number is SELECT CAST(GETDATE() AS INT)

5. The **format** of the output (not the random numbers) should look exactly like the following for every day in the month of January, 2007.

```
sales_date   avg_sales
------------ -----------------------
Jan  1 2007  222.133333
Jan  2 2007  693.723333
Jan  3 2007  459.563333
Jan  4 2007  876.516666
Jan  5 2007  314.486666
          ...
Jan 31 2007  518.671666
```

6. Assume that the last day of the week is a Saturday. Give the week ending total sales for the company for the entire year of 2007. Be aware that different divisions of the company may choose a different last day of the week so make the query able to easily change the last day of the week. You will likely find the following table useful for your solution:

```
CREATE TABLE WeekDays
(
 dayname VARCHAR(9) NOT NULL PRIMARY KEY,
 dayid   INT       NOT NULL UNIQUE
)

INSERT INTO WeekDays VALUES('Sunday', 1)
INSERT INTO WeekDays VALUES('Monday', 2)
INSERT INTO WeekDays VALUES('Tuesday', 3)
INSERT INTO WeekDays VALUES('Wednesday', 4)
INSERT INTO WeekDays VALUES('Thursday', 5)
INSERT INTO WeekDays VALUES('Friday', 6)
INSERT INTO WeekDays VALUES('Saturday', 7)
```

7. The following SQL statement will use the table WeekDays to calculate the last day of the week and group and order the sales data by week.

```
DECLARE @LastWeekDay AS VARCHAR(9)
SET @LastWeekDay = 'Saturday'

SELECT SalesDate + Diff AS week_end, SUM(DailyTotal) AS week_total
FROM DailySales
 JOIN (SELECT DayName,
       ((SELECT DayId                    /* this DaIid contains the ID of the LastWeekDay (7)*/
        FROM WeekDays
          WHERE DayName = @LastWeekDay) - DayId + 7) % 7 AS Diff     /*this dayid contains the ID of
                                                                       the day from
                                                                       SalesDate*/
        FROM WeekDays) AS W
   ON DATENAME(weekday, SalesDate) = DayName
WHERE YEAR(SalesDate + Diff) = 2007
GROUP BY SalesDate + Diff
ORDER BY SalesDate + Diff
```

Explanation of the algorithm:

Diff contains the difference between current day and the last day of the week. The use of "x-DayId +7" (x=ID of LastWeekDay) and the modulo operator (%) makes it possible to have a variable last day of the week. For example, if the last day of the week would be Sunday (1) and the sales date Friday (6), Diff would be (1-6+7)%7 = 2.

The T-SQL statement DATENAME(weekday,SalesDate) returns the name of the weekday for the SakesDate in the DailySales table.

Any date plus (e.g. SalesDate) its calculated difference to the end of the week (Diff), equals the numerical value of the last day of the week. For example, if the SalesDate is 39,118 (Wednesday, Feb.7, 2007) and the difference is 3 (Wednesday= 4; (7-4+7)%7=3) the last day of the week is 39,121.
This allows the grouping and ordering of the sales data by the end of the week.

8. The output should look similar to the following:

week_end	week_total
2007-01-06 00:00:00.000	5973.91
2007-01-13 00:00:00.000	11013.82
2007-01-20 00:00:00.000	15097.12
2007-01-27 00:00:00.000	11591.05

XML AND XSL MODULE FOR PUBLISHING DATA ON THE INTERNET

Purpose: This example shows how to export a table from SQL Server 2005 to an XML document and attach an XSL Stylesheet for the purpose of publishing it on the Internet. When exporting data from a database using XML and displaying it on a webpage a stylesheet needs to be created with formatting instructions for the browser.

Relationship to standards: SQL Server 2005, and Visual Studio 2005.

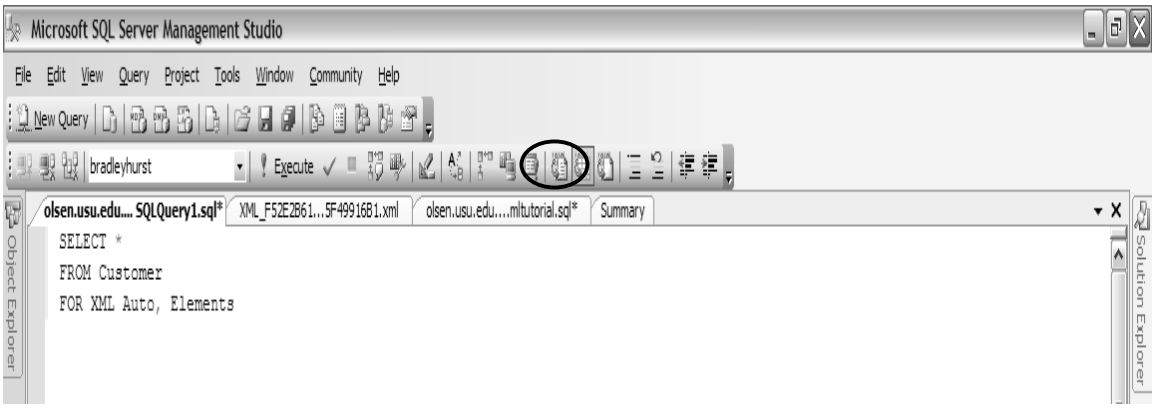
1. Create a root folder called "xmlTutorial" in the c:/ drive.

2. Open SQL Server Management Studio.

3. Open a new query window and copy the following code to create and populate table “Customers”:

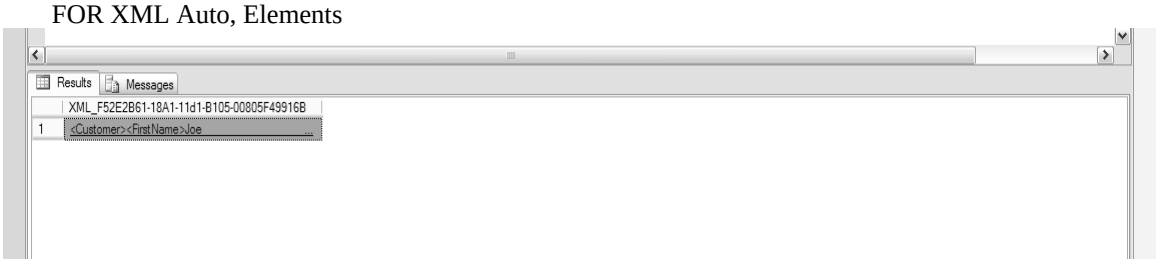
```
CREATE TABLE Customer
(
        FirstName Char(50) NULL,
        LastName Char(50) NULL,
        Address Char(50) NULL,
        City Char(20) NULL,
        State Char(2) NULL,
        Zip char(9) NULL,
)
GO
DELETE Customer

INSERT INTO dbo.Customer (FirstName, LastName, Address, City, State, Zip)
Values('Joe', 'Schmoe', '1125 East 200 N', 'Logan', 'UT', '84321')
GO
INSERT INTO dbo.Customer (FirstName, LastName, Address, City, State, Zip)
Values('Mary', 'Schmoe', '1125 East 200 N', 'Logan', 'UT', '84321')
GO
INSERT INTO dbo.Customer (FirstName, LastName, Address, City, State, Zip)
Values('Ike', 'Schmoe', '1125 East 200 N', 'Logan', 'UT', '84321')
GO
```

4. Open a new query and make sure that “Results to Grid” is selected as shown below:



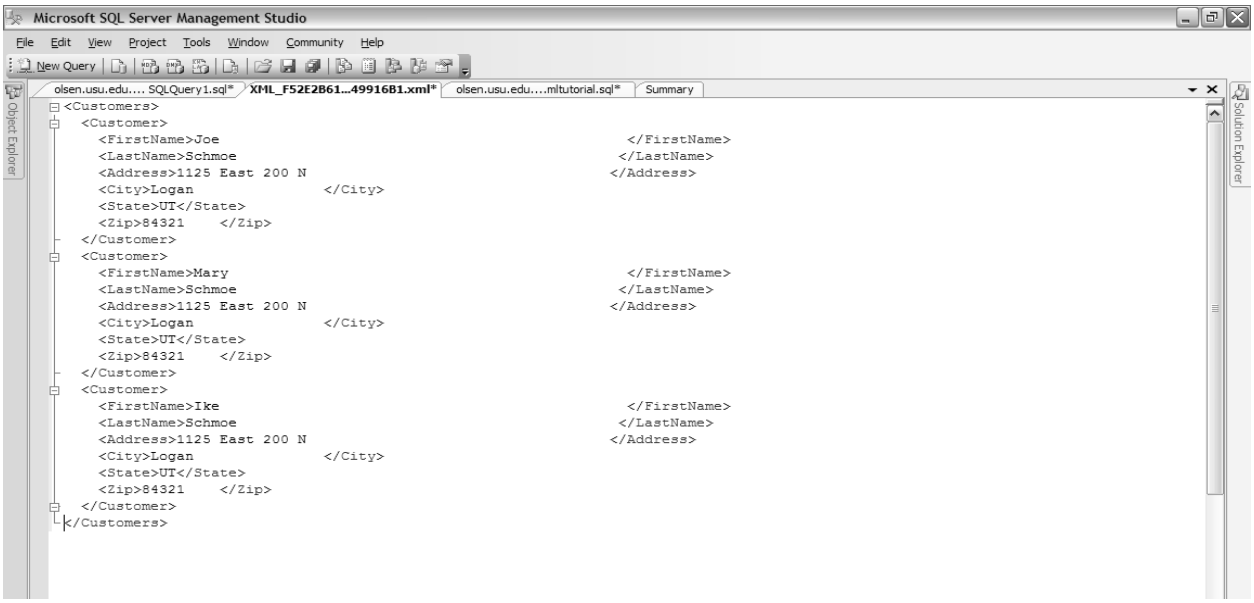
5. Run the following query to transform the table data into XML format:

```
SELECT *
FROM Customer
FOR XML Auto, Elements
```



6. A link appears in the results pane. Click on the link.
   The link opens in a new window and displays the customer data in XML format.

7. Create a root tag for the XML data:
   - Select all of the data and cut it using Ctrl+X.
   - Insert a new tag <Customers>. Typing <Customers> will cause the ending tag</Customers> to appear automatically.
   - Paste the XML data you cut in between the tags as follows:



8. Save it to the "xmlTutorial" Folder as "Customers.xml" and close down SQL Server Management Studio.

9. Open the file in any XML editor and attach an XSLT stylesheet. A stylesheet contains the information that is necessary for the web browser to read or translate the XML code into a format that the browser can display.

10. Open Visual Studio 2005 and go to File, New, File and create a new XSLT file.
Here is an example of an XSLT stylesheet that works with our Customers.xml file. After clearing the existing data from the Visual Studio window enter the following text.

```
<?xml version=”1.0” encoding=”utf-8”?>
<xsl:stylesheet version=”1.0”
xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>
<xsl:template match=”/”>
<!—this is just setting up the html
<html>
<body>
<table width=”100%” cellspacing=”0”  border=”0”>
<!—this is the headers on the table
<tr  bgcolor=”green”>
<td width=”30%”>
<h3>Customer</h3>
</td>
<td width=”30%”>
<h3>Address</h3>
</td>
<td width=”20%”>
<h3>City</h3>
</td>
<td width=”10%”>
<h3>State</h3>
```

12. </td>

```
<td width=”10%”>
<h3>Zip</h3>
```

```
</td>
</tr>
<!—here starts the xslt code, selecting each sql row of information in the xml document
<xsl:for-each select=”Customers/Customer”>
<tr bgcolor=”#dddddd”>
<!—this bit of code alternates row color
<xsl:if test=”position() mod 2 != 1”>
<xsl:attribute name=”style”>background-color:gray</xsl:attribute>
</xsl:if>
<td width=”30%”>
<xsl:value-of select=”FirstName” />
<xsl:value-of select=”LastName” />
</td>
<td width =”30%”>
<xsl:value-of select=”Address”/>
</td>
<td width =”20%”>
<xsl:value-of select=”City” />
</td>
<td width=”10%”>
<xsl:value-of select=”State” />
</td>
<td width=”10%”>
<xsl:value-of select=”Zip” />
</td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

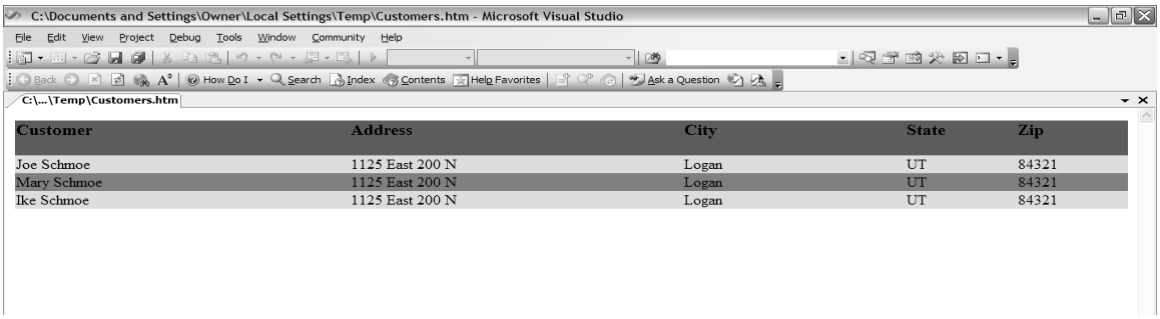
11. Save the XSLT file as Tutorial.xslt in the c:/xmltutorial folder

12. Attach the XSLT to the XML file.
    - Still in Visual Studio, go File, Open File and open “Customers.xml”.
    - In the top two lines add the following two tags:

```
<?xml version=”1.0” encoding=”UTF-8”?>
<?xml-stylesheet type=”text/xsl” href=”C:\xmlTutorial\tutorial.xslt”?>
```

- The second line tells where to look for the XSLT.

13. To view the transformation of the XML document, make sure the "Customers.xml" file is selected in Visual Studio. Then go to the "XML" menu and select "Show XSLT Output". A window opens with the web browser and displays the following:



OVERLAPS MODULE

Purpose: Students often find it difficult to retrieve data with overlapping date information from two different tables. In this example, we have two tables: the first one contains the names of hotel guests and their arrival and departure dates; the second table contains hotel events (e.g. Valentines Day special) and the start and ending dates of the event. These two tables are not related in any way but one might want to know which guests stayed at the hotel during a specific event, or what events attracted the most guests.

Relationship to standards: This module contains only ANSI-standard SQL and therefore runs in any environment.

1. Create and populate a Guests and a Celebrations tables:

```
CREATE TABLE Guests
(GuestName      CHAR(30)      NOT NULL      PRIMARY KEY,
ArrivalDate     DATETIME      NOT NULL,
DepartDate      DATETIME      NOT NULL)

CREATE TABLE Celebrations
(CelebName      CHAR(30)      NOT NULL      PRIMARY KEY,
StartDate       DATETIME      NOT NULL,
FinishDate      DATETIME      NOT NULL)
```

2. Create a report that returns the GuestName and the number of events that the guest attended. Have the guests ordered by the number of events a guest attended in descending order.

2a. One way to get the a report is to create a VIEW. It comes with all the advantages a view provides, namely appropriate access rights and a good performance. The disadvantages of a view are that i) an object is left behind after the query is executed and ii) if the view is accedentally deleted the query can not be executed any more.

```
CREATE VIEW GuestCelebrations
AS
SELECT GuestName, CelebName
        FROM Guests, Celebrations
        WHERE NOT ((DepartDate < StartDate) OR
        (ArrivalDate > FinishDate))

SELECT GuestName, COUNT(CelebName) AS CelebCount
        FROM GuestCelebrations
GROUP BY GuestName
    ORDER BY CelebCount DESC;
```

2b. Another way to get a report is to derive a table in a nested SQL statement. This is more complex and is usually more difficult for students to understand.

```
SELECT GuestName, COUNT(CelebName) AS CelebCount
   FROM (SELECT GuestName, CelebName
              FROM Guests, Celebrations
              WHERE NOT ((DepartDate < StartDate) OR
                        (ArrivalDate > FinishDate))) AS GuestCelebrations
GROUP BY GuestName
ORDER BY CelebCount DESC;
```

2c. Yet another way to get a report is to use Common Table Expressions (CTE) which is a new functionality in SQL Server 2005 but part of ANSI SQL 99. CTE creates tables on the fly but compared to the VIEW example the table is only temporary and no object is left behind.

```
WITH GuestCelebrations (GuestName, CelebName) AS
(
   SELECT GuestName, CelebName
              FROM Guests, Celebrations
              WHERE NOT ((DepartDate < StartDate) OR
                        (ArrivalDate > FinishDate))
)

SELECT GuestName, COUNT(CelebName) AS CelebCount
   FROM GuestCelebrations
GROUP BY GuestName
ORDER BY CelebCount DESC;
```

DYNAMIC SQL MODULE

Purpose: Dynamic SQL provides increased flexibility when writing Queries. It allows the user to put a SQL statement inside a variable and execute the variable.

Environment: This module contains only ANSI-standard SQL and therefore runs in any environment.

1. SQL already provides some flexibility by allowing the passing of parameters. In this example, we create a stored procedure that returns an employee's first and last name given that the employee ID is passed as an input parameter (i.e. entered in a form on a Webpage). In this example the procedure is executed for the employee with the employeeID 3000.

```
CREATE PROCEDURE EmpName @EmpID INT
AS
SELECT FirstName, LastName
        FROM Employees
        WHERE EmployeeID = @EmpID
GO
EXEC EmpName 3000
```

2. This example shows how to create a stored procedure that will list all the tuples and attributes for a given table. The table name must be passed as a parameter to the stored procedure. The procedure is executed for the table Employees.

Following the structure of the first example the SQL statement looks like this:

```
CREATE PROCEDURE AnyTableSelect @TableName VarChar(100)
AS
SELECT *
        FROM @TableName
GO
EXEC AnyTableSelect Employees
```

When executing the procedure the following error message will appear:

Server: Msg 137,Level 15, State 2, Procedure AnyTableSelect, Line 4
Must declare the variable '@TableName'.

This approach does not work because SQL Server will not pass object names as parameters. However, another approach will work which is called 'dynamic SQL.'

```
CREATE PROCEDURE AnyTableSelect @TableName VarChar(100)
AS
Declare @SQL VarChar(1000)
SELECT @SQL = 'SELECT * FROM '
SELECT @SQL = @SQL + @TableName
EXEC (@SQL)
GO
EXEC AnyTableSelect Employees
```

3. In this example, a stored procedure is created that lists all the tuples and all attributes of the Employees table for a given list of employee ID's. The list of ID's must be passed as a parameter to the stored procedure. The procedure is executed for employees 3000 and 3001.

Following the structure of the first example the SQL statement looks like this:

```
CREATE PROCEDURE FindEmps1 @EmpIDs VarChar(100)
AS
SELECT *
        FROM Employees
        WHERE EmployeeID IN (@EmpIDs)
GO
EXEC FindEmps1 "3000,3001"
```

When executing the procedure the following error message will appear:

Server: Msg 245, Level 16, State 1, Procedure FindEmps1, Line 4
Syntax error converting the varchar value '3000,3001' to a column of data type int.

This approach doesn't work because a variable cannot be passed to an IN clause.

The following approach using Dynamic SQL works because the list is included in a string of SQL commands that are executed dynamically.

```
CREATE PROCEDURE FindEmps @EmpIDs VarChar(100)
AS
DECLARE @SQL VARCHAR(1000)
SELECT @SQL = 'SELECT * FROM Employees '
SELECT @SQL = @SQL + ' WHERE EmployeeID IN (' + @EmpIDs +')'
```

```
EXEC (@SQL)
GO
EXEC FindEmps "3000,3001"
```

4. In this last example, a stored procedure is created that lists the tuples of the Employees table for a given list of employee last names. The list of last names must be passed as a parameter to the stored procedure. The attributes to be listed are: EmployeeID, FirstName (only the first 14 characters), LastName, and Employee Comments.

```
CREATE PROCEDURE FindNames @LastName VarChar(100)
AS
EXEC ('SELECT EmployeeID, Firstname=SUBSTRING(FirstName,1,14),LastName, EmployeeComments
        FROM Employees
        WHERE LastName IN (' + @LastName + ')
        Order BY LastName, EmployeeID')
GO
SET QUOTED_IDENTIFIER OFF
EXEC FindNames "'Jackson', 'Adams'"
GO
```

CONCLUSION AND RECOMMENDATIONS

We have developed and presented a framework for integrating advanced database modules in post-introductory, advanced database management courses. The framework can be used to categorize modules into one of the following areas: Security, Performance, Internet or Ease of use functionality enhancements. We have provided classroom-ready examples for each of the framework's categories. Given the importance of database skills in today's business environment, these advanced database skills are very valuable for students in order to successfully compete in the present job market.

Educators need to respond to the changing requirements of the job market and provide students with the appropriate skills. Companies need to be continuously surveyed about the specific areas of the expanding database knowledge set that is currently needed in the workplace and the value of each advanced database knowledge topic. We foresee that future developments in the area of advanced database modules will include the creation of modules and teaching cases that require advanced database skills. Part of the ongoing improvements in teaching database related topics is the inclusion of these emerging database skills into the advanced database related courses. These efforts will have to include the creation of test-banks and measurements of student learning outcomes. The creation of online example databases, with already filled tables, would also greatly help instructors to provide students with a variety of examples for all categories of the framework.

REFERENCES

1. Alhajj, R., Simplifying the Formulation of a Wide Range of Object-Oriented Complex Queries. *Journal of Database Management*. 11(2): p. 20-29. 2000.
2. Babcock, C. *Data, Data, Everywhere*. 2006, Available from: http://www.informationweek.com/shared/printableArticle.jhtml?articleID=175801775 [accessed 12/26/2006].
3. Codd, E.F., A Relational Model of Data for Large Shared Data Banks. *Communications of the ACM* 13(6). 1970.
4. Dadashzadeh, M., A Simpler Approach to Set Comparison Queries in SQL. *Journal of Information Systems Education*. 14(4): p. 345-348. 2003.
5. Ekstrom, J.J., and Renshaw, S.R., Database Curriculum Issues for Four-Year IT Programs, in CIEC Conference: Tucson, Arizona. p. ETD343-1 to ETD343-9. 2003.
6. Gorgone, J.T., Davis, G.B., Valacich, J.S., Topi, H., Fenistein, D.L., and Longnecker, H. E. . IS 2002 Model Curriculum and Guidelines for Undergraduate Degree Programs in Information Systems. 2002, Available from: http://www.acm.org/education/is2002.pdf [accessed January 2005].
7. Green, G.C., Great's Gym: A Teaching Case for Term-Long Database Projects. *Journal of Information systems Education*. 16(4): p. 387-390. 2005.
8. Halevy, Answering queries using views: A survey. *The VLDB Journal*. 10(4): p. 270-294. 2001.
9. Jukic, N., Nestorov, S., Vrbsky, S.V., and Parrish, A., Enhancing Database Access Control by Facilitating Non-Key Related Cover Stories. *Journal of Database Management*. 16(3): p. 1-20. 2005.
10. Kauffman, R.J., March, S.T., and Wood, C.A., Mapping Out Design Aspects for Data-Collecting Agents. *International Journal of Intelligent Systems in Accounting, Finance, and Management*. 9(4): p. 217-236. 2000.
11. Kung, M., Yang, S.C., and Zhang Y., The Changing Information Systems (IS) Curriculum: A Survey of Undergraduate Programs in the United States. *Journal of Education for Business*. 81(6): p. 291-300. 2006.
12. Matos, V.M., and Grasser, R., A Simpler (and Better) SQL Approach to Relational Division. *Journal of Information Systems Education*. 13(2): p. 85-88. 2002.
13. Nicolaou, A.I., P.A. Essex, and M. Raghunathan, E-Hermes: An XML Tool for the Classroom. *Journal of Information Systems Education*. 16(3): p. 351-368. 2005.
14. Nielsen, J., User interface directions for the web. *Communications of the ACM*. 42(1): p. 65-72. 1999.
15. Olsen, D., Cooney, V., Marshall B., and Swart, R., Toward Full Integration of XML And Advanced Database Concepts. *The Review of Business Information Systems*. 9(4): p. 13-22. 2005.
16. Pons, A.P., Database Tuning and Its Role in Information Technology Education. *Journal of Information Systems Education*. 14(4): p. 381-387. 2003.
17. Prabhakar, B., Litecky C.R. and Arnett K. , IT Skills in a Tough Job Market. *Communications of the ACM*. 48(10): p. 91-94. 2005.
18. Robbert, M.A. and C.M. Ricardo, Trends in the Evolution of the Database Curriculum. *SIGCSE Bulletin*. 35(3): p. 139-143. 2003.
19. Robbert, M.A., et al., The database course (panel session): what must be taught *SIGCSE Bulletin*. 32(1): p. 403-404. 2000.
20. Seyed-Abbassi, B., The evolution of an advanced databse course in an information systems curriculum, in ISECON 2002, 19th Annual Conference for Information Systems Educators: San Antonio, TX. 2002.
21. Springsteel, F., M.A. Robbert, and C.M. Ricardo, The next decade of the database course: three decades speak to the next in Proceedings of the thirty-first SIGCSE technical symposium on Computer science education Austin, Texas, United States p. 41-45. 2000.
22. *SQL Server Performance Tuning Articles*. Available from: http://www.sql-server-performance.com/articles_performance.asp [accessed 12/15/06].
23. U.S. Department of Labor: Bureau of Labor Statistics. *Occupational Outlook Quarterly*. 2006-07 Edition, Available from: http://www.bls.gov/oco/ocos042.htm [accessed 12/27/06].
24. Urban, S.D. and S.W. Dietrich, Advanced database concepts for undergraduates: experience with teaching a second course, in Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education Charlotte, NC. p. 357-361. 2001.

25. Wood, C.A.a.O., T.T., WEBVIEW An SQL Extension for Joining Corporate Data to Data Derived from the World Wide Web. *Communications of the ACM*. 48(9): p. 1-13. 2005.
26. *XML*. Available from: http://en.wikipedia.org/wiki/XML
27. Yuhanna, M. *Terabyte-Size Databases Are Still Challenging* 2005, Available from: http://www.forrester.com/Research/Document/Excerpt/0,7211,37729,00.html [accessed 03/01/2007].

NOTES